Reinforcement Learning via Gaussian Processes with Neural Network Dual Kernels

Imene R. Goumiri
Physics Division, Lawrence Livermore National Laboratory
Livermore, California, USA
goumiri1@llnl.gov

Benjamin W. Priest
Center for Applied Scientific Computing Lawrence Livermore National Laboratory
Livermore, California, USA
priest2@llnl.gov

Michael D. Schneider
Physics Division, Lawrence Livermore National Laboratory
Livermore, California, USA
schneider42@llnl.gov

***Abstract*—While deep neural networks (DNNs) and Gaussian Processes (GPs) are both popularly utilized to solve problems in reinforcement learning, both approaches feature undesirable drawbacks for challenging problems. DNNs learn complex non-linear embeddings, but do not naturally quantify uncertainty and are often data-inefficient to train. GPs infer posterior distributions over functions, but popular kernels exhibit limited expressivity on complex and high-dimensional data. Fortunately, recently discovered conjugate and neural tangent kernel functions encode the behavior of overparameterized neural networks in the kernel domain. We demonstrate that these kernels can be efficiently applied to regression and reinforcement learning problems by analyzing a baseline case study.**

We apply GPs with neural network dual kernels to solve reinforcement learning tasks for the first time. We demonstrate, using the well understood mountain-car problem, that GPs empowered with dual kernels perform at least as well as those using the conventional radial basis function kernel. We conjecture that by inheriting the probabilistic rigor of GPs and the powerful embedding properties of DNNs, GPs using NN dual kernels will empower future reinforcement learning models on difficult domains.

***Index Terms*—Reinforcement Learning; Gaussian Processes; Deep Neural Networks**

I. Introduction

The traditional approach in optimal control posits a controller with a suite of control signals able to affect a known dynamical system. The problem is to devise a policy for scheduling control signals in order to achieve some given objective. As there is no uncertainty in the model, finding such a policy becomes an optimization problem.

However, many applications involve decision-making challenges where data are limited and the generative models are complex and partially or completely unknown. As such, the reinforcement learning (RL) branch of machine learning arose to develop models for an agent or agents acting on an initially unknown environment. RL algorithms learn a policy to guide agent actions in order to achieve some high-level goal by acting on its environment and using the response to model its dynamics.

Although RL and optimal control are related, these research fields are traditionally separate. Ultimately, both are concerned with sequential decision making to minimize an expected long-term cost. The dynamical system, controller, and control signals of optimal control roughly map onto the environment, agent(s), and actions of RL.

Many RL algorithms [4, 22, 35, 36, 40] address a lack of dynamics knowledge by way of a reliance upon parametric adaptive elements or control policies whose number of parameters or features are fixed and predetermined. These parameters are usually then learned from data. Deep neural networks (DNNs) are also used extensively in RL [26, 27, 41].

Deep reinforcement learning (Deep RL) has become increasingly popular since the demonstration of its super-human performance in playing Atari games [27]. Following this, RL has been successfully applied in many contexts, e.g., games [14], physically-based animations [23, 34], and robotics [11].

DNNs are attractive as they are known to have an excellent representative power [12, 15, 17]. However, tuning and training the parameters is a data-inefficient practice [30]. Moreover, DNNs usually include no natural means of quantifying the uncertainty in their predictions [9]. Thus trained models may overconfidently predict the unknown dynamics when the system operates outside of the observed domain. Such overconfident prediction can lead to system instability, thereby making any controller stability results unachievable.

Nonparametric kernel methods such as Gaussian processes (GPs) [37] have also been applied to reinforcement learning tasks [7, 8, 16, 18, 19, 29, 32]. GPs are popular in many areas of machine learning due to their flexibility, interpretability, and natural uncertainty quantification due to being Bayesian models. However GP-related data-driven methods remain largely unexploited in optimal control.

The choice of GP kernel function encodes our prior beliefs about the distribution of the function of interest and is a key part of modeling. For example, Kuss and Rasmussen use the famous radial basis function (RBF) in a GP to solve the mountain-car problem, implying that the dynamics are believed to be very smooth [18]. However, GPs often struggle to learn the features of complex or high-dimensional data, worrying the researcher interested in extrapolating this approach to such domains.

Recent results have shown a duality between wide, random DNNs and GPs through the use of the conjugate kernels (CK)[5, 6, 20, 25, 31] and neural tangent kernels (NTK)

[1, 2, 13, 39]. These kernels capture, in a sense that will be made explicit in Section II, the nonlinear feature embedding learned by the corresponding DNN architecture. However, these kernels are at present mostly treated as an academic curiosity and have predominantly been applied to image classification problems [1].

Our Contributions. We recreate the mountain-car experiment of [18] using GPs with the NN dual kernels. Our results compare favorably with GPs using the RBF kernel. We also describe our optimized algorithm for this reinforcement learning.

II. Gaussian Processes, Neural Networks, and Dual Kernels

We will briefly review GPs, DNNs, and the correspondence between GPs and infinitely wide Bayesian DNNs. We will focus on the computation of the various models, and largely omit training details.

GPs are flexible, nonparametric Bayesian models that specify a *prior distribution over a function* $f : \mathcal{X} \to \mathcal{Y}$ that can be updated by data $\mathcal{D} \subset \mathcal{X} \times \mathcal{Y}$. Coarsely, a GP is a collection of random variables, any finite subset of which has a multivariate Gaussian distribution. We say that $f \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot))$, where $m : \mathcal{X} \to \mathbb{R}$ is a mean function and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a positive definite covariance function with hyperparameters θ. In practice m is often assumed to be the zero function. For any finite $\mathbf{X} \subset \mathcal{X}$,

$$\mathbf{f} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)]^\top \sim \mathcal{N}(\mathbf{0}, \mathbf{K_{ff}}) \tag{1}$$

is the prior over f at the locations $\mathbf{x}$. Here $\mathbf{K_{ff}}$ is an $n \times n$ matrix whose (i, j)th element is $k(\mathbf{x}_i, \mathbf{x}_j) = \text{cov}(f(\mathbf{x}_i), f(\mathbf{x}_j))$. Such covariance matrices implicitly depend on θ. If we observe $\mathbf{y} = \mathbf{f} + \epsilon$, where $\epsilon_i \sim \mathcal{N}(0, \sigma_\epsilon^2)$ is homoscedastic noise, then the predictive distribution evaluated at finite locations $\mathbf{X}_* \subset \mathcal{X}$ is given by

$$\begin{aligned}\mathbf{f}_* \mid \mathbf{X}, \mathbf{X}_*, \mathbf{y} \sim \mathcal{N}\big(&\mathbf{K}_{*\mathbf{f}} \left(\mathbf{K_{ff}} + \sigma^2 \mathbf{I}\right)^{-1} \mathbf{y}, \\ &\mathbf{K}_{**} - \mathbf{K}_{*\mathbf{f}} \left(\mathbf{K_{ff}} + \sigma^2 \mathbf{I}\right)^{-1} \mathbf{K}_{\mathbf{f}*}\big).\end{aligned} \tag{2}$$

Here $\mathbf{K}_{*\mathbf{f}} = \mathbf{K}_{\mathbf{f}*}{}^\top$ is the cross-covariance matrix between $\mathbf{X}_*$ and $\mathbf{X}$.

The expressiveness of a GP is heavily dependent upon the choice of kernel function k. Most common functions, for example the RBF kernel,

$$k_{\text{RBF}}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{\ell^2}\right), \tag{3}$$

exhibit limited expressiveness on complex data and impose sometimes-inappropriate assumptions such as stationarity. GPs also suffer from cubic scaling in the observation size, although a rich literature of approximations addresses this problem. In exchange, GPs allow fully-Bayesian inference, and exhibit robust uncertainty quantification by way of providing full posterior distributions.

DNNs learn an embedding of inputs into a latent space by way of iteratively applying nonlinear transforms. This embedding transforms highly-nonlinear data relationships into a linear feature space, allowing a final linear regression to produce predictions. In contrast to GPs, DNNs are highly parametric, often utilizing more parameters than observations. For this reason, DNNs often require large amounts of training data, and a vast literature has developed around heuristic training protocols. While DNNs do not, in general, produce posterior distributions, their popularity is due to good empirical performance on complex and high-dimensional data. A DNN with L layers and widths $\{n^\ell\}_{\ell=0}^L$ has parameters consisting of weight matrices $\{W^\ell \in \mathbb{R}^{n^\ell \times n^{\ell-1}}\}_{\ell=1}^L$ and biases $\{\mathbf{b}^\ell \in \mathbb{R}^\ell\}_{\ell=1}^L$. We will assume the NTK parameterization and introduce hyperparameters σ_w and σ_b, whose interpretation we will define in Section II-A. The output of a DNN on input $\mathbf{x}$ is $\mathbf{h}^L(\mathbf{x})$, which is computed recursively as

$$\begin{aligned}\mathbf{h}^1(\mathbf{x}) &= \frac{\sigma_w}{\sqrt{n^0}} W^1 \mathbf{x} + \sigma_b b^1, \\ \mathbf{h}^\ell(\mathbf{x}) &= \frac{\sigma_w}{\sqrt{n^{\ell-1}}} W^\ell \phi\left(\mathbf{h}^{\ell-1}(\mathbf{x})\right) + \sigma_b b^\ell.\end{aligned} \tag{4}$$

Here $\phi(\cdot)$ is an element-wise scalar nonlinear activation function, such as the popular ReLU function:

$$\phi_{\text{ReLU}}(x) = \max\{0, x\}. \tag{5}$$

A. Dual Kernels

As we have noted, GPs and DNNs have different advantages and disadvantages. Many attempts have been made to obtain "the best of both worlds" - the uncertainty quantification and interpretability of GPs along with the computational convenience and expressivity of DNNs. Such efforts include Bayesian neural networks, which apply prior distributions to the weights of neural networks [31], and applying GPs to feature vectors embedded by DNNs [24]. Interestingly, a direct correspondence between GPs and Bayesian DNNs of any depth arises as the hidden layers become sufficiently wide. We will briefly motivate this correspondence, its history and applications.

Initializing all of the parameters in a DNN as $W_{i,j}^\ell \sim \mathcal{N}\left(0, \frac{\sigma_w^2}{n^{\ell-1}}\right)$ and $\mathbf{b}_i^\ell \sim \mathcal{N}\left(0, \sigma_b^2\right)$ for $i \in \left[n^\ell\right]$ and $j \in \left[n^{\ell-1}\right]$ (i.e. Glorot initialization [20]) is common in practice. Note that this is the equivalent of initializing all of the parameters in Eq. (4) as i.i.d. $\mathcal{N}(0, 1)$. In the study of highly overparameterized (wide) models over the last several decades, investigators made two unexpected observations.

1) Random initialization followed by training only the final linear layer often produces high-quality predictions.
2) Training overparameterized models tends to produce weights that differ only slightly from initialization.

The correspondence between infinitely wide single hidden layer neural networks with i.i.d. Gaussian weights and biases was first discovered as far back as the 1990s by Neal by application of the Central Limit Theorem [31]. Recently, others have extended Neal's result to infinitely wide deep neural networks [20, 25] and convolutional neural networks with infinitely many channels [10, 33]. Arora et al. improved

these results by showing that the correspondence holds for *finite* neural networks that are sufficiently wide [1] and showed empirical evidence that the kernel process behavior occurs at lower widths than theoretically guaranteed [2]. The kernel corresponding to wide DNNs is referred to in the literature as the conjugate kernel (CK) [6] or NNGP kernel [20]. Transforming a nonlinear transform of a DNN to kernel form requires obtaining a dual form of the nonlinearity ϕ given positive definite kernel matrix K representing the kernel defined by the lower layers [5, 6]. For nonlinearity ϕ and kernel matrix K the dual form is known to be

$$V_\phi(K)(\mathbf{x}, \mathbf{x}') = \mathop{\mathbb{E}}_{f \sim \mathcal{N}(\mathbf{0}, K)} \phi(f(\mathbf{x}))\phi(f(\mathbf{x}')). \tag{6}$$

Using this dual transform and the notation of Eq. (4) and following the formulation of [39], we can express the CK recursively as

$$\begin{aligned} \Sigma^1(\mathbf{x}, \mathbf{x}') &= \frac{\sigma_w^2}{n^0}\langle \mathbf{x}, \mathbf{x}' \rangle + \sigma_b^2 \\ \Sigma^\ell(\mathbf{x}, \mathbf{x}') &= \sigma_w^2 V_\phi(\Sigma^{\ell-1})(\mathbf{x}, \mathbf{x}') + \sigma_b^2. \end{aligned} \tag{7}$$

The last layer kernel Σ^L is the conjugate kernel for the network. This kernel corresponds exactly to that of the linear model resulting from randomly initializing all weights and training the last layer.

If the CK lends mathematical rigor to observation 1) above, the neural tangent kernel (NTK) does the same with observation 2). Intuitively, the NTK corresponds to a generalization of the CK where we train the whole model, rather than only the last layer. The NTK emerges from the observation that infinitely wide neural networks evolve as linear models under stochastic gradient descent [13, 21] and has also been shown to generalize to convolutional and finite architectures [1]. Evidence suggests that the NTK might be capable of learning more complex features than the CK [39], and the NTK has recently been shown to deliver competitive predictions in an SVM on small data learning benchmarks [2]. We will omit the derivation, which is somewhat involved, and instead recite the form of the NTK Θ^L as given in [39]:

$$\begin{aligned} \Theta^1(\mathbf{x}, \mathbf{x}') &= \Sigma^1(\mathbf{x}, \mathbf{x}') \\ \Theta^\ell(\mathbf{x}, \mathbf{x}') &= \Sigma^\ell(\mathbf{x}, \mathbf{x}') + \sigma_w^2 \Theta^{\ell-1}(\mathbf{x}, \mathbf{x}') V_{\phi'}(\Sigma^{\ell-1})(\mathbf{x}, \mathbf{x}'). \end{aligned} \tag{8}$$

At first blush, the formulations of Eqs. (7) and (8) are unhelpful, as computing Eq. (6) is intractable. Fortunately, closed-form solutions are known for several common activation functions [5, 6], enabling efficient computation. Throughout the rest of this document we will consider only networks utilizing ϕ_{ReLU}, which is known to have analytic dual activations:

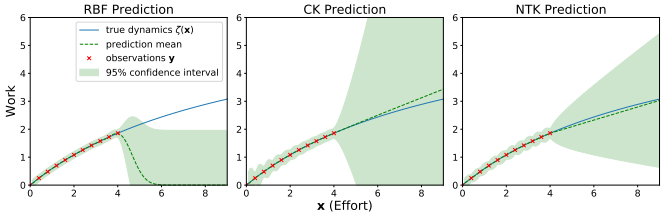

Fig. 1. Predictive distributions of the RBF, CK and NTK trained on observations derived from the dynamics $\zeta(\cdot)$.

$$V_{\phi_{\text{ReLU}}}(K)(\mathbf{x}, \mathbf{x}') = \frac{\sqrt{K(\mathbf{x},\mathbf{x})K(\mathbf{x}',\mathbf{x}')}}{2\pi}\left(\sin c + (\pi - c)\cos c\right) \tag{9}$$

$$V_{\phi'_{\text{ReLU}}}(K)(\mathbf{x}, \mathbf{x}') = \frac{1}{2\pi}(\pi - c) \tag{10}$$

$$c = \arccos\left(\frac{K(\mathbf{x}, \mathbf{x}')}{\sqrt{K(\mathbf{x}, \mathbf{x})K(\mathbf{x}', \mathbf{x}')}}\right). \tag{11}$$

B. A motivating example

We will illustrate the usage of the RBF kernel along with CK and NTK on a model of a simple machine. Consider the central example given in [3] of moving a weight up a slope. We will assume that we are trying to learn a true process driven by the dynamics

$$\zeta(x \mid \theta, a) = \frac{\theta x}{1 - x/a}. \tag{12}$$

Here x is a control parameter modeling the amount of force exerted on the system, while θ and a are unknown. In terms of the model, the numerator of Eq. (12) corresponds to the ideal efficiency of the machine, while the denominator corresponds to inefficiency (such as loss due to friction).

Say that we wish to model Eq. (12) using a GP, and that we have observed a vector of responses $\mathbf{y}$ at 11 locations $\mathbf{x}$ evenly-spaced in $[0.1, 4]$. Then we believe that for each $i \in [11]$,

$$\mathbf{y}_i = f(\mathbf{x}_i) + \epsilon_i \tag{13}$$

where $f \sim \mathcal{GP}(\mathbf{0}, k(\cdot, \cdot))$ for some kernel function k and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ is measurement noise.

We simulate the dynamics

$$\mathbf{y}_i = \zeta(\mathbf{x}_i \mid 0.65, 10) + \epsilon_i \tag{14}$$

for $\epsilon_i \sim \mathcal{N}(0, 0.1^2)$ and fit GP models for each of k_{RBF}, k_{CK}, and k_{NTK} as defined above. We use Eq. (2) to learn the posterior distributions of each GP over a set $\mathbf{x}_*$ uniformly spaced in $[0.2, 9]$ and fit the hyperparameters of each kernel by way of a simple grid search using the loglikelihood. See [37] Chapter 2 for a comprehensive review of GP regression.

Figure 1 plots the means of the resulting distributions and their 95% confidence intervals, along with the true dynamics in blue and the observations in red. Note that the RBF GP returns to the prior mean 0 when extrapolating far from the observed data. This is expected of stationary kernels, as inputs that are

far apart are assumed to have low correlation. As given in Eqs. (7) and (8), both the CK and NTK kernels are functions of $\langle \mathbf{x}, \mathbf{x}' \rangle$, $\|\mathbf{x}\|$, and $\|\mathbf{x}'\|$. Thus, they are *nonstationary* on $\mathbb{R}^{n^0}$. It is worth noting that most extant GP applications of CK and NTK use image data that has been normalized to the unit hypersphere [1, 13, 20, 25]. In this case, CK and NTK are functions of the angle between the unit vectors $\mathbf{x}$ and $\mathbf{x}'$, which maps one-to-one with $\|\mathbf{x} - \mathbf{x}'\|$. Consequently, in the aforementioned applications CK and NTK are isotropic. We do not perform normalization nor do we embed our data in a higher dimensional hypersphere in this work, meaning that in all cases the CK and NTK kernels are nonstationary.

The fact that the posterior means of CK and NTK trend closer to the true dynamics far from the training data does not imply that these kernels are somehow "better" than RBF, but rather that their implicit assumptions about how the data is organized happen to center relatively well on this example. More careful accounting of model discrepancy, such as that demonstrated in [3], can produce much better extrapolation. Note that in all cases, however, the confidence interval grows dramatically as we move further from the observed data. This behavior indicates a low confidence in any projections in these data, which provides a good example of what is desired from uncertainty quantification. The majority of practical problems involve high dimensional transformations that are much harder to visualize. Thankfully, the posterior distributions still allow the investigator to detect where predictions are uncertain due to the presence of high variance. The rest of this document concerns itself with such an application to reinforcement learning.

III. The Mountain-Car Reinforcement Learning Problem

A. Description

The reinforcement learning problem studied in this paper is the mountain-car problem: a car drives along a mountain track and the objective is to drive to the top of the mountain. However gravity is stronger than the engine, and even at full thrust the car cannot accelerate up the steep slope. The only way to solve the problem is to first accelerate backwards, away from the goal, and then apply full thrust forwards, building up enough speed to carry over the steep slope even while slowing down the whole way. Thus, one must initially move away from the goal in order to reach it in the long run. This is a simple example of a task whose optimal solution is unintuitive: things must get worse before they can get better. The problem is fully described in [28, 35] and is illustrated in Figure 2.

The mountain-car dynamical system has two continuous state variables, the position of the car x, and the velocity of the car $\dot{x}$. The state s can be written as $s = (x, \dot{x})$. The mountain surface is described by the altitude

$$H(x) = \begin{cases} x^2 + x & \text{if } x < 0, \\ \frac{x}{\sqrt{1+5x^2}} & \text{if } x \geq 0. \end{cases} \tag{15}$$

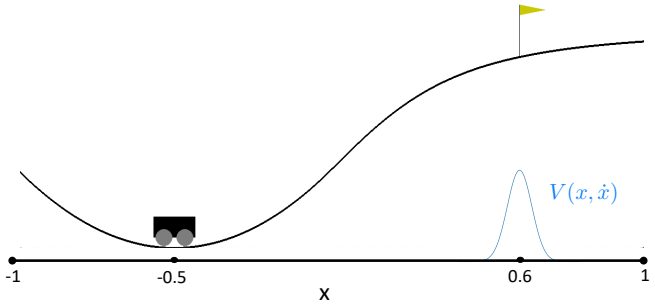

Fig. 2. Illustration of the mountain car problem. The car is initially resting motionless at $x = -0.5$ and the goal is to bring it up and hold it in the region around the flag.

The input that the driver can apply is the horizontal force F. Boundary conditions are imposed for each of the position, velocity and force of the car respectively as follows:

$$\begin{aligned} -1 &\leq x \leq 1 \\ -2 &\leq \dot{x} \leq 2 \\ -4 &\leq F \leq 4. \end{aligned} \tag{16}$$

The initial state $s_0 = [-0.5, 0]^T$ indicates the car is at the unmoving at minimum altitude. This is the also the equilibrium of the dynamics. The target reward R is a multivariate gaussian PDF with mean $(x = 0.6, \dot{x} = 0)$ and covariance $\sigma^2 I_2$ with $\sigma = 0.05$. R is plotted in the top of Fig. 4.

This choice of instantaneous reward function encodes into the model a desire to be as close as possible to the flag at position $x = 6.0$ while remaining as stationary as possible. The agent's goal is to find the optimal trajectory for the car to maneuver towards and remain near the flag, given the dynamics of the environment. The core RL problem here is to find a policy for the decision maker (driver/car): a function π that specifies the action $F = \pi(s)$ that the decision maker will choose when in state s.

The standard family of algorithms to calculate this optimal policy constructs two arrays indexed by state: policy π and value V. Upon completion of the algorithm, $\pi(s)$ specifies the action to be taken in state s, while $V(s)$ is the real-valued discounted sum of the rewards to be earned by following that solution from s.

This RL algorithm has two steps, (1) a value update and (2) a policy update, which are iterated across all the states until π and V converge. The Bellman equation is commonly used to update the value V:

$$V(s) = \int P_{\pi(s)}(s, s') \left[R_{\pi(s)}(s, s') + \gamma V(s') \right] ds'. \tag{17}$$

Here γ is the discount factor and satisfies $0 \leq \gamma \leq 1$, $P_{\pi(s)}$ is the transition probability of going from state s to state s' when applying action $\pi(s)$ and $R_{\pi(s)}$ is the corresponding immediate expected reward. Given a computed value function V

for a given policy π, we can compute an implicitly optimized update policy π' as:

$$\pi(s) = \text{argmax}_a \left\{ \int_{s'} P(s' \mid s, a) \left[R(s' \mid s, a) + \gamma V(s') \right] ds' \right\} \tag{18}$$

Section III-B explains the algorithm in detail. The main idea is that we iterate the process of evaluating V for a given policy π over the continuous state space using Eq. (17) and then recompute the policy using Eq. (18).

B. Algorithmic Implementation

Our algorithm is a generalization of the algorithm described in [18] which is able to accommodate the three different kernels described in the previous section while maintaining computational efficiency. It proceeds by first initializing the dynamics of the model and value function, then iterating over updating the value and policy until convergence. We model the dynamics using GPs. In doing so, we explicitly solve the dynamics for a small number of observed position/velocity states, then train GPs to interpolate the state evolution of unobserved states. Similarly, we use a separate GP to model the value function at a small number of position/velocity states, each with a small number uniform sample forces. We iterate over this GP, applying interpolation to update the learned policy which we in turn use to update the GP.

a) **Initialization of the dynamics:** The first step is to train a GP to predict the dynamics of the system. The dynamical equation is

$$\frac{d}{dt}\begin{pmatrix} x \\ \dot{x} \\ F \end{pmatrix} = \begin{pmatrix} \dot{x} \\ F - G \cdot \sin(\arctan(H'(x))) \\ 0 \end{pmatrix}. \tag{19}$$

Here G is the gravitational constant and H' is the derivative of the altitude given in Eq. (15) with respect to x. Given a state s, we integrate Eq. (19) forward in time over a span Δt of 0.3 s to obtain the corresponding *next* state s'. For training we take $N_d = 128$ random 3D states s_i chosen uniformly in the domain defined by (16) and we compute their corresponding next states s'_i. We use these s - s' pairs as observations to train two GPs, one for x and one for $\dot{x}$. We can then utilize Eq. (2) to interpolate the dynamics evolution at unobserved states. We assume the hyperparameters of both CK and NTK to be distributed according to an inverse-Gamma distribution. We use a Monte Carlo Markov Chain technique to fit them by minimizing the mean square error when predicting the dynamics. See [37] for a nuanced discussion of hyperparameter optimization. Both kernels provide comparable accuracy for predicting the dynamics once trained and tuned.

b) **Initialization of the value function:** Next we must train a GP to predict the value function of any given state. The procedure is iterative so we use the reward R as the initial value function. As with the dynamics, we take a certain number ($N_V = 512$) of random states in the 3D domain of (s_j, F) uniformly from the domain (16) and associate them with their corresponding initial value ($\equiv R$) to provide training samples. Note that contrary to [37], we train that GP using the full 3D state $(x, \dot{x}, F)$ as input rather than omitting F for reasons that will be apparent in the description of the iterations below. We tune the hyperparameters of the value GPs using the same MCMC procedure applied to the dynamics.

c) **Iteration of the value and policy:** Once all the GPs are trained and tuned, we can start iterating to update the value GP and the policy until the value function converges to a fixed point. For each state s_j in the dynamics training set, we generate a sequence of $N_F = 128$ states $s_k = (x_k, \dot{x}_k, F_k)$ where $\forall k$, $x_k = x_j$, $\dot{x}_k = \dot{x}_j$, and the actions F_k are uniformly spaced and cover the entire F domain. Then we use the dynamics GPs to predict their respective next states s'_k as the posterior means of Eq. (2). s'_k then serves as input to the value GP to predict V_k, again as the posterior mean of Eq. (2). We can then compute $V_j^{\max} = \max_k V_k$ and $k_{\max} = \arg\max_k V_k$ and deduce the policy $\pi(s_j) = F_{k_{\max}}$. Finally we can update the value associated with each s_j using $V_j \leftarrow R(s_j) + \gamma \cdot V_j^{\max}$.

Once the value has been updated, we retrain the value GP. We repeat this procedure until the value stops evolving. Once it does we output the optimal policy π.

C. Results

We show that GPs using either both the CK and NTK as their kernel functions are suitable for solving the mountain car reinforcement learning problem. The estimated dynamics are predicted with sufficient accuracy to enable the iterative evolution of the value function.

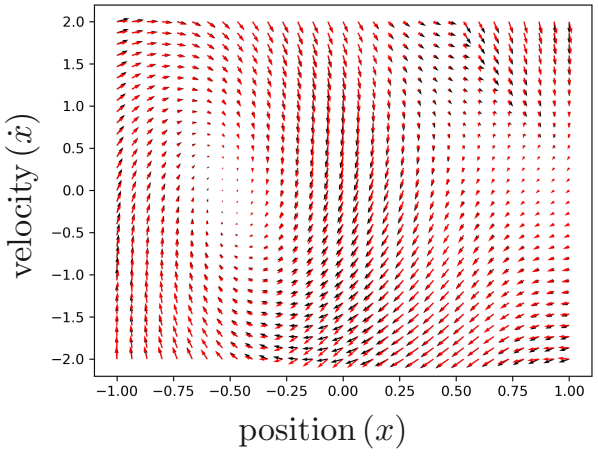

Fig. 3. True (black) and predicted (red) dynamics as a function of x and $\dot{x}$ (for $F = 0$). Each arrow represents a state s (base of the arrow) and points in the direction of its *next* state s' 0.3 s in the future. The arrow lengths are scaled down so as not to overlap. The stable equilibrium at $(-0.5, 0)$ corresponds to the bottom of the valley. The target at (0.6, 0) is unstable requiring a sustained force $F > 0$ to maintain the car at the target. The discontinuity in the upper right of the phase plot is due to boundary conditions: hitting the boundary of the domain brings the velocity to zero.

Figure 3 shows the comparison between the dynamics derived from the physics (the truth) and the dynamics predicted using CK Gaussian process modeling. We can see that our

dynamical model is very close to the reality. It captures the main features and equilibria of the dynamical system.

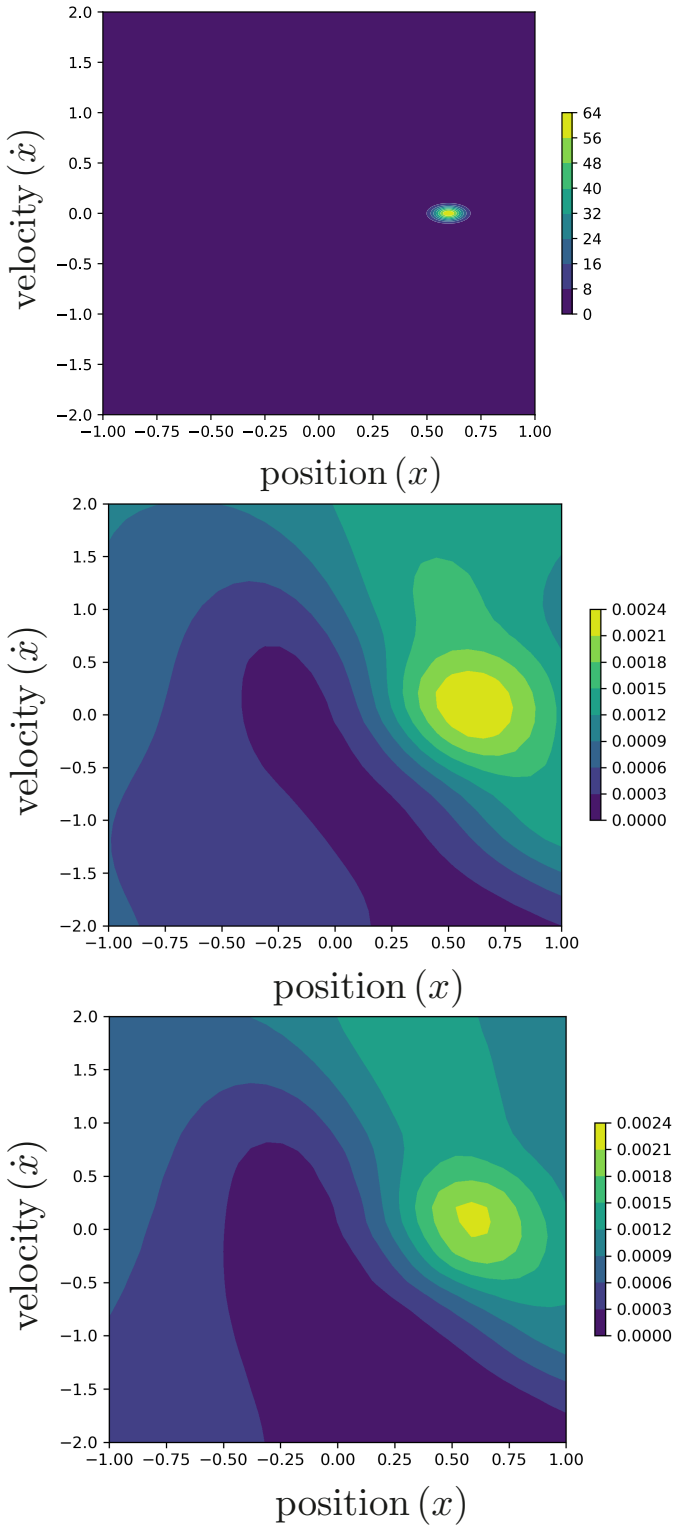

Fig. 4. Initial reward (top) and final value function for CK (middle) and NTK (bottom) as a function of x and $\dot{x}$ for $F = 0$.

Figure 4 shows the initial value (top) which is the instant reward, a Gaussian function centered around the target at $(x, \dot{x}) = (0.6, 0)$ with a small standard deviation of 0.05, and the final value function for CK (middle) and NTK (bottom). For both kernels, the value function converges in six iterations. The value expands diagonally from the target to regions where the velocity is high enough to overcome the steep slope and finally curves back to reach the car's initial position from the left, leading to the non-trivial but correct policy that the car should start by going backward before speeding up the slope. The value function does not increase from zero in the central region of the phase space, which corresponds to the invalid policy of attempting to climb the slope of the mountain in the positive x-direction without sufficient momentum.

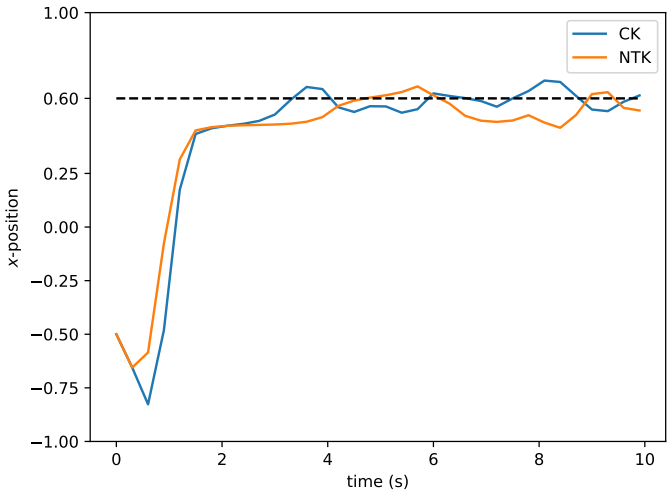

Fig. 5. Evolution of the x position over time. The car goes backward initially then speeds up the hill.

Figure 5 illustrates the learned optimal trajectories of the car along the x-axis over time. Again for both kernels, the car first moves backward then speeds up to quickly reach the target, where it stays indefinitely. The oscillations of the car around the target location ($x = 0.6$) are the result of compounding errors in the GP predictions of the dynamics and the value. These oscillations can most likely be reduced by adding more training points, both for the dynamics and the value, but this comes at the cost of additional computation.

In our training of the GP representation of the value function, we see there are clear *regions of interest* in the value function that change with each training iteration. This indicates that choosing training points uniformly in the entire domain and keeping the same points throughout iterations is suboptimal. It would be better to sample more points where the value is higher so as to achieve better resolution in this region. In other words, we expect to see improved numerical performance by converting the value function into a probability density function for sampling training points followed by resampling the points after each policy iteration step to accommodate the changing value function. This should allow more accurate predictions without the performance cost of adding more training points.

IV. Perspectives and conclusion

We have shown that GP kernels that are dual descriptions of neural networks are suitable for solving a simple reinforcement learning problem. The kernels we use here have been shown to perform well for GP classification tasks [e.g., 20], but we believe our result is the first application of such kernels for GP regression in a non-trivial problem. We have also improved the GP model for the value function from those models presented in the literature [i.e., 18] to increase the computational efficiency of the policy iteration step by decreasing the number of sample points in the combination of phase space and possible actions. We are able to achieve this performance improvement because of the improved expressivity of the GP regression in the combined sample space.

While this simple mountain-car RL problem turns out to be easily soluble with GPs utilizing the classic RBF kernel, we have shown that neural network dual kernels deliver similar performance. Furthermore, we expect that more challenging RL problems that have benefitted from neural networks for modeling the dynamics and the value function may also benefit in the future from the GP dual description of those networks [e.g., 27]. In particular, RL problems relying on computer vision may benefit from application of the convolution version of the CK or NTK [1]. Additionally, the ongoing development of kernels dual to arbitrary architectures opens up the possibility of taking advantage of recurrent neural network expressivity within the GP paradigm [38]. The GP dual to neural networks applied to RL thus offers promise of incorporating recent advances in deep RL with the probabilistic modeling features of GPs. Such applications also elude the grasp of more conventional GP models in the current literature due to the expressivity limitations of known kernels, especially on high-dimensional data.

We also have not fully exploited the value in utilizing dual GPs. All of the predictions given throughout this document utilize only the posterior mean of Eq. (2) for prediction. In this sense, we might as well have actually used overparameterized DNNs for prediction. We expect that knowledge of the posterior variance will be greatly beneficial in more advanced RL problems where dynamics and value function propagation involves more uncertainty. We will incorporate applications of uncertainty quantification into future work.

References

[1] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *arXiv preprint arXiv:1904.11955*, 2019.

[2] Sanjeev Arora, Simon S Du, Zhiyuan Li, Ruslan Salakhutdinov, Ruosong Wang, and Dingli Yu. Harnessing the power of infinitely wide deep nets on small-data tasks. *arXiv preprint arXiv:1910.01663*, 2019.

[3] Jennỳ Brynjarsdóttir and Anthony O'Hagan. Learning about physical parameters: The importance of model discrepancy. *Inverse problems*, 30(11):114007, 2014.

[4] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*. CRC press, 2017.

[5] Youngmin Cho and Lawrence K Saul. Kernel methods for deep learning. In *Advances in neural information processing systems*, pages 342–350, 2009.

[6] Amit Daniely, Roy Frostig, and Yoram Singer. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. In *Advances In Neural Information Processing Systems*, pages 2253–2261, 2016.

[7] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):408–423, 2013.

[8] Marc Peter Deisenroth, Carl Edward Rasmussen, and Jan Peters. Gaussian process dynamic programming. *Neurocomputing*, 72(7-9):1508–1524, 2009.

[9] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Insights and applications. In *Deep Learning Workshop, ICML*, volume 1, page 2, 2015.

[10] Adrià Garriga-Alonso, Carl Edward Rasmussen, and Laurence Aitchison. Deep convolutional networks as shallow gaussian processes. *arXiv preprint arXiv:1808.05587*, 2018.

[11] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[12] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

[13] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[14] Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018.

[15] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, 2013.

[16] Jonathan Ko, Daniel J Kleint, Dieter Fox, and Dirk Haehnelt. Gp-ukf: Unscented kalman filters with gaussian process prediction and observation models. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1901–1907. IEEE, 2007.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural

networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[18] Malte Kuss and Carl E Rasmussen. Gaussian processes in reinforcement learning. In *Advances in neural information processing systems*, pages 751–758, 2004.

[19] Neil Lawrence. Probabilistic non-linear principal component analysis with gaussian process latent variable models. *Journal of machine learning research*, 6(Nov):1783–1816, 2005.

[20] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. In *International Conference on Learning Representations*, 2018.

[21] Jaehoon Lee, Lechao Xiao, Samuel S Schoenholz, Yasaman Bahri, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *arXiv preprint arXiv:1902.06720*, 2019.

[22] Frank L Lewis, Draguna Vrabie, and Kyriakos G Vamvoudakis. Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers. *IEEE Control Systems Magazine*, 32(6):76–105, 2012.

[23] Libin Liu and Jessica Hodgins. Learning to schedule control fragments for physics-based characters using deep q-learning. *ACM Transactions on Graphics (TOG)*, 36(3):1–14, 2017.

[24] Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid. Convolutional kernel networks. In *Advances in neural information processing systems*, pages 2627–2635, 2014.

[25] Alexander G de G Matthews, Mark Rowland, Jiri Hron, Richard E Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks. In *Internation Conference on Learning Representation*, 2018.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[28] Andrew W Moore. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. In *Advances in neural information processing systems*, pages 711–718, 1994.

[29] Roderick Murray-Smith and Daniel Sbarbaro. Nonlinear adaptive control using nonparametric gaussian process prior models. *IFAC Proceedings Volumes*, 35(1):325–330, 2002.

[30] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[31] Radford M Neal. Priors for infinite networks. In *Bayesian Learning for Neural Networks*, pages 29–53. Springer, 1996.

[32] Duy Nguyen-Tuong and Jan Peters. Local gaussian process regression for real-time model-based robot control. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 380–385. IEEE, 2008.

[33] Roman Novak, Lechao Xiao, Jaehoon Lee, Yasaman Bahri, Greg Yang, Dan Abolafia, Jeffrey Pennington, and Jascha Sohl-dickstein. Bayesian deep convolutional neural networks with many channels are gaussian processes. In *International Conference on Learning Representation*, 2019.

[34] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[35] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[36] Kyriakos G Vamvoudakis, Hamidreza Modares, Bahare Kiumarsi, and Frank L Lewis. Game theory-based control system algorithms with real-time reinforcement learning: How to solve multiplayer games online. *IEEE Control Systems Magazine*, 37(1):33–52, 2017.

[37] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT Press Cambridge, MA, 2006.

[38] Greg Yang. Tensor programs i: Wide feedforward or recurrent neural networks of any architecture are gaussian processes. *arXiv preprint arXiv:1910.12478*, 2019.

[39] Greg Yang and Hadi Salman. A fine-grained spectral perspective on neural networks. *arXiv preprint arXiv:1907.10599*, 2019.

[40] Yuanheng Zhu and Dongbin Zhao. Comprehensive comparison of online adp algorithms for continuous-time optimal control. *Artificial Intelligence Review*, 49(4):531–547, 2018.

[41] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016.