Learning Scalable Deep Kernels with Recurrent Structure

Maruan Al-Shedivat
Carnegie Mellon University
alshedivat@cs.cmu.edu

Andrew Gordon Wilson
Cornell University
andrew@cornell.edu

Yunus Saatchi
saatchi@cantab.net

Zhiting Hu
Carnegie Mellon University
zhitingh@cs.cmu.edu

Eric P. Xing
Carnegie Mellon University
epxing@cs.cmu.edu

Abstract

Many applications in speech, robotics, finance, and biology deal with sequential data, where ordering matters and recurrent structures are common. However, this structure cannot be easily captured by standard kernel functions. To model such structure, we propose expressive closed-form kernel functions for Gaussian processes. The resulting model, GP-LSTM, fully encapsulates the inductive biases of long short-term memory (LSTM) recurrent networks, while retaining the non-parametric probabilistic advantages of Gaussian processes. We learn the properties of the proposed kernels by optimizing the Gaussian process marginal likelihood using a new provably convergent semi-stochastic procedure and exploit the structure of these kernels for fast and scalable training and prediction. We demonstrate state-of-the-art performance on several benchmarks, and thoroughly investigate a consequential autonomous driving application, where the predictive uncertainties provided by GP-LSTM are uniquely valuable.

1 Introduction

There exists a vast array of machine learning applications where the underlying datasets are sequential. Applications range from the entirety of robotics, to speech, audio and video processing. While neural-network-based approaches have dealt with the issue of *representation learning* for sequential data, the important question of modeling and propagating uncertainty across time has rarely been addressed by these models. For a robotics application such as a self-driving car, however, it is not just desirable, but essential to have complete predictive densities for variables of interest. When trying to stay in lane and keep a safe following distance from the vehicle front, knowing the uncertainty associated with lanes and lead vehicles is as important as the point estimates.

Recurrent models with long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997] have recently emerged as the leading approach to modeling sequential structure. The LSTM is an efficient gradient-based method for training recurrent networks. It uses a memory cell inside each hidden unit and a special gating mechanism that stabilizes the flow of the back-propagated errors and hence improves the learning process of the model. While the LSTM provides state-of-the-art results on speech and text data [Graves et al., 2013, Sutskever et al., 2014], quantifying uncertainty or extracting full predictive distributions from deep models models is still an area of active research [Gal and Ghahramani, 2016].

In this paper, we quantify the predictive uncertainty of deep models by following a Bayesian nonparametric approach. In particular, we propose kernel functions which fully encapsulate the structural properties of LSTMs, for use with Gaussian processes. The resulting model enables Gaussian processes to achieve state-of-the-art performance on sequential regression tasks, while also allowing for a principled representation of uncertainty, and non-parametric flexibility. For scalability, we use semi-stochastic optimization and exploit the algebraic structure of these kernels, decomposing the relevant covariance matrices into Kronecker products of circulant matrices, for $\mathcal{O}(n)$ training time and $\mathcal{O}(1)$ test predictions [Wilson et al., 2015]. Our model not only can be interpreted as a Gaussian process with a recurrent kernel, but also as a deep recurrent network with probabilistic outputs, infinitely many hidden units, and a utility function robust to overfitting. Hence it naturally fits into the standard computational framework of deep learning models.

Throughout this paper, we assume basic familiarity with Gaussian processes (GPs). We have provided only a brief introduction to GPs in the background section; for a comprehensive reference, see, e.g., Williams and Rasmussen [2006]. In the following sections, we formalize the problem of learning from sequential data, provide background on recurrent networks and the LSTM, and present an extensive empirical evaluation of our model. Specifically, we apply our model to a number of tasks, including system identification, energy forecasting, and self-driving car applications. Quantitatively, the model is assessed on the data ranging in size from hundreds of points to almost a million with various signal-to-noise ratios demonstrating state-of-the-art performance, and the linear scaling of our approach. Qualitatively, the model is tested on consequential self-driving applications: lane estimation and lead vehicle position prediction. Indeed, the main focus of this paper is on achieving state-of-the-art performance on consequential applications involving sequential data, following straightforward and scalable approaches to building highly flexible Gaussian process.

2 Background

We consider the problem of learning a regression function that maps sequences to real-valued targets. Formally, let $\bar{\mathbf{X}} = \{\bar{\mathbf{x}}_i\}_{i=1}^n$ be a collection of sequences, $\bar{\mathbf{x}}_i = [\mathbf{x}_i^1, \mathbf{x}_i^2, \cdots, \mathbf{x}_i^{l_i}]$, each corresponding length l_i, where $\mathbf{x}_i^j \in \mathcal{X}$, and $\mathcal{X}$ is an arbitrary domain. Let $\mathbf{y} = \{y_i\}_{i=1}^n$, $y_i \in \mathbb{R}$, be a collection of corresponding real-valued targets. Assuming that the maximum length of a sequence is L, the goal is to learn a function, $f : \mathcal{X}^L \mapsto \mathbb{R}$, from some family, $\mathcal{F}$, based on the available data.

As our working example, consider the problem of estimating position of the lead vehicle at the next time step from LIDAR, GPS, and gyroscopic measurements of a self-driving car available for a number of previous steps. This task is a classical instance of the *sequence-to-reals regression*, where a temporal sequence of measurements is regressed to the future position estimates. In our notation, the sequences of input vectors of measurements, $\bar{\mathbf{x}}_1 = [\mathbf{x}^1]$, $\bar{\mathbf{x}}_2 = [\mathbf{x}^1, \mathbf{x}^2]$, ..., $\bar{\mathbf{x}}_n = [\mathbf{x}^1, \mathbf{x}^2, \cdots \mathbf{x}^n]$, are indexed by time and would be of growing lengths. However, typically, input sequences are truncated to a finite-time horizon, L, that is assumed to be predictive for the future targets of interest.

Note that the problem of learning a mapping, $f : \mathcal{X}^L \mapsto \mathbb{R}$, is challenging. While considering the whole sequences of observations as input features is necessary for capturing long-term temporal correlations, it virtually blows up the dimensionality of the problem.

If we assume that each measurement is d-dimensional, i.e., $\mathcal{X} \subseteq \mathbb{R}^d$, and consider L previous steps as distinct features, the regression problem will become $(L \times d)$-dimensional. Therefore, to avoid overfitting and be able to extract signal from a finite amount of data, it is crucial to exploit the sequential nature of observations.

Recurrent models. One of the most successful ways to exploit sequential structure of the data is by using a class of recurrent models. In the sequence-to-reals regression scenario, such a model expresses the mapping $f : \mathcal{X}^L \mapsto \mathbb{R}$ in the following general recurrent form:

$$\mathbf{y}^t = \psi(\mathbf{h}^t) + \epsilon^t, \ \mathbf{h}^t = \phi(\mathbf{h}^{t-1}, \mathbf{x}^t) + \delta^t, \tag{1}$$

where $\mathbf{x}^t$ is an input observation at time t, $\mathbf{h}^t$ is a corresponding latent representation, and $\mathbf{y}^t$ is the corresponding target value. Functions $\phi(\cdot)$ and $\psi(\cdot)$ model transition and emission, respectively, and δ^t and ϵ^t are additive noises. While $\phi(\cdot)$ and $\psi(\cdot)$ can be arbitrary, they are typically time-invariant. This strong but realistic assumption incorporated into the structure of the recurrent mapping significantly reduces the complexity of the family of functions, $\mathcal{F}$, regularizes the problem, and prevents overfitting.

Recurrent models can account for various patterns in sequences by *memorizing* internal representations of their dynamics via adjusting ϕ and ψ. Recurrent neural networks (RNNs) model recurrent processes by using linear parametric maps followed by nonlinear activations:

$$\mathbf{y}^t = \psi(\mathbf{W}_{hy}^\top \mathbf{h}^{t-1}), \ \mathbf{h}^t = \phi(\mathbf{W}_{hh}^\top \mathbf{h}^{t-1}, \mathbf{W}_{xh}^\top \mathbf{x}^{t-1}), \tag{2}$$

where $\mathbf{W}_{th}, \mathbf{W}_{tx}, \mathbf{W}_{eh}$ are weight matrices to be learned[1] and $\phi(\cdot)$ and $\psi(\cdot)$ here are some fixed element-wise functions. Importantly and contrary to the standard hidden Markov models (HMMs), *the state* of an RNN at any time t is *distributed* and effectively represented by an entire hidden sequence, $[\mathbf{h}^1, \cdots, \mathbf{h}^{t-1}, \mathbf{h}^t]$. To model complex relationships, it is possible to further stack together multiple recurrent layers [Pascanu et al., 2013]. A major disadvantage of the vanilla RNNs is that their training is nontrivial due to the so-called *vanishing gradient* problem [Bengio et al., 1994]: the error back-propagated through t time steps diminishes exponentially which makes learning long-term relationships nearly impossible.

LSTM. To overcome vanishing gradients, Hochreiter and Schmidhuber [1997] proposed a long short-term memory (LSTM) mechanism that places a *memory cell* into each hidden unit and uses differentiable gating variables. The update rules for the hidden representation at time t have the following form (here $\sigma(\cdot)$ and $\tanh(\cdot)$ are sigmoid and hyperbolic tangent functions, respectively):
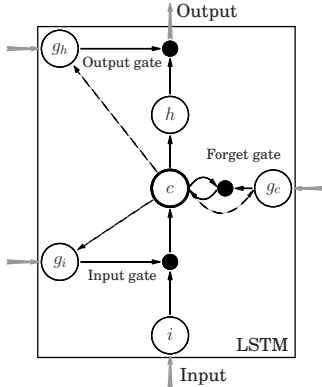
$$
\begin{aligned}
\mathbf{i}^t &= \tanh\left(\mathbf{W}_{xc}^\top \mathbf{x}^t + \mathbf{W}_{hc}^\top \mathbf{h}^{t-1} + \mathbf{b}_c\right), \\
\mathbf{g}_i^t &= \sigma\left(\mathbf{W}_{xi}^\top \mathbf{x}^t + \mathbf{W}_{hi}^\top \mathbf{h}^{t-1} + \mathbf{W}_{ci}^\top \mathbf{c}^{t-1} + \mathbf{b}_i\right), \\
\mathbf{c}^t &= \mathbf{g}_c^t\, \mathbf{c}^{t-1} + \mathbf{g}_i^t\, \mathbf{i}^t, \\
\mathbf{g}_c^t &= \sigma\left(\mathbf{W}_{xf}^\top \mathbf{x}^t + \mathbf{W}_{hf}^\top \mathbf{h}^{t-1} + \mathbf{W}_{cf}^\top \mathbf{c}_{t-1} + \mathbf{b}_f\right), \\
\mathbf{o}^t &= \tanh(\mathbf{c}^t), \\
\mathbf{g}_o^t &= \sigma\left(\mathbf{W}_{xo}^\top \mathbf{x}^t + \mathbf{W}_{ho}^\top \mathbf{h}^{t-1} + \mathbf{W}_{co}^\top \mathbf{c}^t + \mathbf{b}_o\right), \\
\mathbf{h}^t &= \mathbf{g}_o^t\, \mathbf{o}^t.
\end{aligned}
\tag{3}
$$

---

[1]The bias terms are omitted for clarity of presentation.

As illustrated above, $\mathbf{g}_i^t$, $\mathbf{g}_c^t$, and $\mathbf{g}_o^t$ correspond to the input, forget, and output gates, respectively. These variables take their values in $[0, 1]$ and when combined with the internal states, $\mathbf{c}^t$, and inputs, $\mathbf{x}^t$, in a *multiplicative* fashion, they play the role of *soft gating*. The gating mechanism not only improves the flow of errors through time, but also, allows the the network to decide whether to keep, erase, or overwrite certain memorized information based on the forward flow of inputs and the backward flow of errors. This mechanism adds stability to the network's memory.

Gaussian processes. The Gaussian process (GP) is a Bayesian nonparametric model that generalizes the Gaussian distributions to functions. We say that a random function f is drawn from a GP with a mean function μ and a covariance kernel k, $f \sim \mathcal{GP}(\mu, k)$, if for any vector of inputs, $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$, the corresponding vector of function values is Gaussian

$$[f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n)] \sim \mathcal{N}(\mu, K_{X,X}),$$

with mean μ, such that $\mu_i = \mu(\mathbf{x}_i)$, and covariance matrix $K_{X,X}$ that satisfies $(K_{X,X})_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. GPs can be seen as distributions over the reproducing kernel Hilbert space (RKHS) of functions which is uniquely defined by the kernel function, k [Schölkopf and Smola, 2002]. GPs with RBF kernels are known to be universal approximators with prior support to within an arbitrarily small epsilon band of any continuous function [Micchelli et al., 2006].

Assuming additive Gaussian noise, $y \mid \mathbf{x} \sim \mathcal{N}(f(\mathbf{x}), \sigma^2)$, and a GP prior on $f(\mathbf{x})$, given training inputs $\mathbf{X}$ and training data $\mathbf{y}$, the predictive distribution of the GP evaluated at arbitrary test points $\mathbf{X}_*$ is:

$$\mathbf{f}_* \mid \mathbf{X}_*, \mathbf{X}, \mathbf{Y}, \sigma^2 \sim \mathcal{N}(\mathbb{E}[\mathbf{f}_*], \mathrm{Cov}[\mathbf{f}_*]), \tag{4}$$

where

$$\begin{aligned}
\mathbb{E}[\mathbf{f}_*] &= \mu_{X_*} + K_{X_*,X}[K_{X,X} + \sigma^2 I]^{-1}\mathbf{y}, \\
\mathrm{Cov}[\mathbf{f}_*] &= K_{X_*,X_*} - K_{X_*,X}[K_{X,X} + \sigma^2 I]^{-1}K_{X,X_*}.
\end{aligned} \tag{5}$$

Here, $K_{X_*,X}$, K_{X,X_*}, $K_{X,X}$, and K_{X_*,X_*} are matrices that consist of the covariance function, k, evaluated at the corresponding points, $\mathbf{x} \in X$ and $\mathbf{x}_* \in X_*$, and μ_* is the mean function evaluated at $\mathbf{x}_* \in X_*$. GPs are fit to the data by optimizing *the evidence*—the marginal probability of the data given the model—with respect to kernel hyperparameters. The evidence has the form:

$$\log p(\mathbf{y} \mid \mathbf{X}) = - \left[\mathbf{y}^\top (K + \sigma^2 I)^{-1}\mathbf{y} + \log \det(K + \sigma^2 I) \right] + \mathrm{const}, \tag{6}$$

where we use a shorthand K for $K_{X,X}$, and K implicitly depends on the kernel hyperparameters. This objective function consists of a *model fit* and a *complexity penalty* term that results in an automatic Occam's razor for realizable functions [Rasmussen and Ghahramani, 2001]. By optimizing the evidence with respect to the kernel hyperparameters, we effectively learn the the structure of the space of functional relationships between the inputs and the targets. For further details on Gaussian processes and relevant literature we refer interested readers to the classical book by Williams and Rasmussen [2006].

Turning back to the problem of learning from sequential data, it seems natural to apply the powerful GP machinery to modeling complicated relationships. However, GPs are limited to learning only pairwise correlations between the inputs and are unable to account for long-term dependencies, often dismissing complex temporal structures. Combining GPs with recurrent models has potential to addresses this issue.

3   Related work

The problem of learning from sequential data, especially from temporal sequences, is well known in the control and dynamical systems literature. Stochastic temporal processes are usually described with generative *autoregressive models* (AM) or with *state-space models* (SSM) [Van Overschee and De Moor, 2012]. The former approach includes nonlinear autoregressive models with exogenous inputs (NARX) that are constructed by using, *e.g.*, neural networks [Lin et al., 1996] or Gaussian processes [Kocijan et al., 2005]. The latter approach additionally introduces unobservable variables, *the state*, and constructs an autoregressive dynamics in the latent space. This construction allows to explicitly model the signal (via the state evolution) and the noise. Alternatively, generative SSMs can be also used in conjunction with discriminative models via the Fisher kernel [Jaakkola and Haussler, 1999].

Modeling time series with GPs is equivalent to using *linear-Gaussian* autoregressive or SSM models [Box et al., 1994]. Learning and inference are efficient in such models, but they are not designed to capture long-term dependencies or correlations beyond pairwise. Wang et al. [2005] introduced GP-based state-space models (GP-SSM) that use GPs for transition and/or observation functions. These models appear to be more general and flexible as they account for uncertainty in the state dynamics, though require complicated approximate training and inference, which are hard to scale [Turner et al., 2010, Frigola et al., 2014].

Perhaps the most recent relevant work to our approach is *recurrent Gaussian processes* (RGP) [Mattos et al., 2015]. RGP extends the GP-SSM framework to regression on sequences by using a recurrent architecture with GP-based activation functions. The structure of the RGP model mimics the standard RNN, where every parametric layer is substituted with a Gaussian process. This procedure allows one to propagate uncertainty throughout the network for an additional cost. Inference is intractable in RGP, and efficient training requires a sophisticated *approximation* procedure, the so-called *recurrent variational Bayes*. In addition, the authors have to turn to RNN-based approximation of the variational mean functions to battle the growth of the number of variational parameters with the size of data. While technically beautiful and promising, RGP seems problematic from the application perspective, especially in its implementation and scalability aspects.

Our model, while aiming to solve the same problem, has a few distinctive differences from the previous work. Firstly, one of our goals is to keep the model as simple as possible while being able to represent and quantify predictive uncertainty. We maintain an analytical objective function and refrain from complicated and difficult-to-diagnose inference schemes. This simplicity is achieved by giving up the idea of trying to connect GPs in a recurrent fashion. Instead, we propose to directly *learn kernels* with recurrent structure via joint optimization of a simple functional composition of a standard GP with a recurrent model (*e.g.*, LSTM), as described in detail in the following section. A similar approach has been previously explored and proved to be fruitful for non-recurrent deep networks [Wilson et al., 2016]. Additionally, uncertainty over the recurrent parts of our model can always be represented via dropout, which is computationally cheap and turns out to be equivalent to approximate Bayesian inference in a deep Gaussian process [Damianou and Lawrence, 2013] with particular intermediate kernels [Gal and Ghahramani, 2016]. Finally, one can also view GP-LSTM as a standalone flexible Gaussian process, which leverages learning techniques that scale to massive datasets [Wilson et al., 2015].
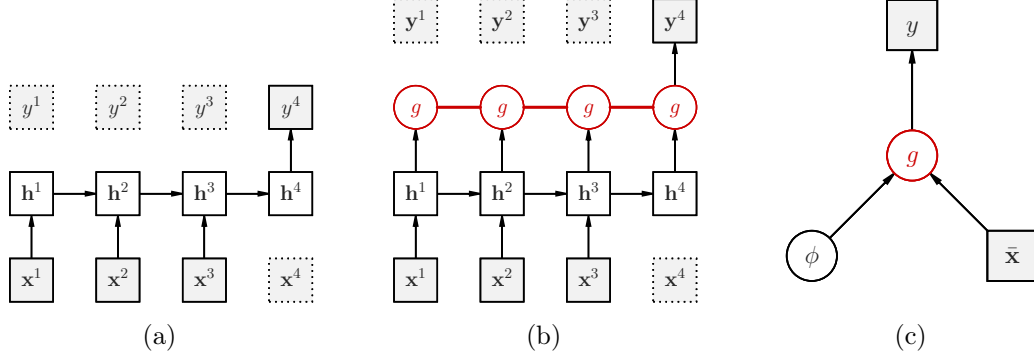
Figure 1: (a) Graphical representation of a recurrent model (RNN/LSTM) that maps input sequences to an output. Dotted boxes represent data instances unused at the given time step. Shaded variables are observable. (b) Graphical model for GP-LSTM. Latent representations are mapped to the outputs through a Gaussian field (denoted in red). (c) Graphical representation of a GP with structured kernel. Squares represent finite-dimensional random variables while circles represent random functions.

4   Learning recurrent kernels

Gaussian processes with different kernel functions correspond to different structured probabilistic models. For example, the Matérn class of kernels induces models with Markovian structure [Stein, 1999]. To construct deep kernels with recurrent structure we transform the original input space with an LSTM network and build a kernel directly in the transformed space, as shown in Figure 1b.

In particular, let $\phi : \mathcal{X}^L \mapsto \mathcal{H}$ be an arbitrary deterministic transformation of the input sequences into some latent space, $\mathcal{H}$. Next, let $k : \mathcal{H}^2 \mapsto \mathbb{R}$ be a real-valued kernel defined on $\mathcal{H}$. The decomposition of the kernel and the transformation, $\tilde{k} = k \circ \phi$, is defined as

$$\tilde{k}(\bar{\mathbf{x}}, \bar{\mathbf{x}}') = k(\phi(\bar{\mathbf{x}}), \phi(\bar{\mathbf{x}}')), \text{ where } \bar{\mathbf{x}}, \bar{\mathbf{x}}' \in \mathcal{X}^L, \text{ and } \tilde{k} : (\mathcal{X}^L)^2 \mapsto \mathbb{R}. \tag{7}$$

It is trivial to prove that $\tilde{k}(\bar{\mathbf{x}}, \bar{\mathbf{x}}')$ is a valid kernel defined on $\mathcal{X}^L$. In addition, the resulting model can be viewed as simply adding a GP as layer to a neural architecture that represents $\phi(\cdot)$ and changing the standard objective function—mean squared error (MSE)—to the negative log marginal likelihood (NLML) of the resultant GP.

This input transformation technique is certainly not new. Recently, Wilson et al. [2016] successfully used it with feedforward and convolutional architectures for regression. In this paper, we apply the same technique to learn kernels with recurrent structure by transforming input sequences with a (stochastic) recurrent network[2], as shown in Figure 1c.

Unfortunately, once the the MSE objective is substituted with NLML, it no longer factorizes over the data. This prevents us from using the well-established stochastic optimization techniques for training our recurrent model. In the case of feedforward and convolutional networks, Wilson et al. [2016] proposed to pre-train the input transformations and then fine-tune them by jointly optimizing GP hyperparameters and the network weights using full-batch algorithms. When the transformation is recurrent, stochastic updates play a key role. Therefore, we propose a semi-stochastic block-gradient optimization procedure which allows stochastic weight updates and fully joint training of the model from scratch.

---

[2]Generally, recurrent transformation is not necessarily deterministic, e.g., when dropout is used.

4.1 Optimization

The negative log marginal likelihood of the Gaussian process has the following form:

$$\mathcal{L}(K) = - \left[\mathbf{y}^\top (K_y + \sigma^2 I)^{-1} \mathbf{y} + \log \det(K_y + \sigma^2 I) \right] + \text{const}, \tag{8}$$

where $K \triangleq K_y + \sigma^2 I$, the Gram kernel matrix, K_y, is computed on $\{\phi(\bar{\mathbf{x}}_i)\}_{i=1}^N$ and implicitly depends on the base kernel hyperparameters, θ, and the parameters of the recurrent neural transformation, $\phi(\cdot)$, denoted $\mathbf{w}$ and further referred as the *transformation hyperparameters*. Our goal is to optimize $\mathcal{L}$ with respect to both θ and $\mathbf{w}$.

The derivative of the NLML objective with respect to θ is standard and takes the following form [Williams and Rasmussen, 2006]:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{1}{2} \text{tr} \left[\left(K^{-1} \mathbf{y} \mathbf{y}^\top K^{-1} - K^{-1} \right) \frac{\partial K}{\partial \theta} \right], \tag{9}$$

where $\partial K / \partial \theta$ is depends on the kernel function, $k(\cdot, \cdot)$, and usually has an analytic form. The derivative with respect to the l-th transformation hyperparameter, w_l, is as follows:[3]

$$\frac{\partial \mathcal{L}}{\partial w_l} = \frac{1}{2} \sum_{i,j} \left(K^{-1} \mathbf{y} \mathbf{y}^\top K^{-1} - K^{-1} \right)_{ij} \left\{ \left(\frac{\partial k(\mathbf{h}_i, \mathbf{h}_j)}{\partial \mathbf{h}_i} \right)^\top \frac{\partial \mathbf{h}_i}{\partial w_l} + \left(\frac{\partial k(\mathbf{h}_i, \mathbf{h}_j)}{\partial \mathbf{h}_j} \right)^\top \frac{\partial \mathbf{h}_j}{\partial w_l} \right\}, \tag{10}$$

where $\mathbf{h}_i = \phi(\bar{\mathbf{x}}_i)$ corresponds to the latent representation of the the the i-th data instance. Once the derivatives are computed, the model can be trained with first-order or quasi-Newton optimization routine. However, application of the stochastic gradient method—the *de facto* standard optimization routine for deep recurrent networks—is not straightforward: neither the objective, nor its derivatives factorize over the data[4] due to the kernel matrix inverses, and hence convergence is not guaranteed.

Semi-stochastic alternating gradient descent. Observe that once the kernel matrix, K, is *fixed*, the expression, $\left(K^{-1} \mathbf{y} \mathbf{y}^\top K^{-1} - K^{-1} \right)$, can be precomputed on the full data and fixed. Subsequently, Eq. (10) turns into a weighted sum of independent functions of each data point. This observation suggests that, given a *fixed kernel matrix*, one could compute a stochastic update for $\mathbf{w}$ on a mini-batch of training points by only using the corresponding sub-matrix of K. Hence, we propose to optimize GPs with recurrent kernels in a semi-stochastic fashion, alternating between updating the kernel hyperparameters, θ, on the full data first, and then updating the weights of the recurrent network, $\mathbf{w}$, using stochastic steps. The procedure is given in Algorithm 1.

Semi-stochastic alternating gradient descent is a special case of block-stochastic [Xu and Yin, 2015] gradient iteration. While the latter splits all the variables into arbitrary blocks and applies Gauss–Seidel type stochastic gradient updates to each of them, our optimization procedure alternates between applying deterministic updates to θ and stochastic updates to $\mathbf{w}$ of the form $\theta^{(t+1)} \leftarrow \theta^{(t)} + \lambda_t \mathbf{g}_\theta^{(t)}$ and $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \lambda_t \mathbf{g}_\mathbf{w}^{(t)}$. The corresponding Algorithm 1 is provably convergent for convex and non-convex problems under certain conditions. The following theorem adapts the work of Xu and Yin [2015] to our optimization scheme.

---

[3]Step-by-step derivations are given in Appendix B.
[4]Cannot be represented as sums of independent functions of each data point.

Algorithm 1 Semi-stochastic alternating gradient descent.

---

input Data – $(\mathbf{X}, \mathbf{y})$, kernel – $k_\theta(\cdot, \cdot)$, recurrent transformation – $\phi_\mathbf{w}(\cdot)$.

  1: Initialize θ and $\mathbf{w}$; compute initial K.
  2: **repeat**
  3:     **for all** mini-batches $\mathbf{X}_b$ in $\mathbf{X}$ **do**
  4:       $\theta \leftarrow \theta + \text{update}_\theta(\mathbf{X}, \theta, K)$. and $\mathbf{w} \leftarrow \mathbf{w} + \text{update}_\mathbf{w}(\mathbf{X}_b, \mathbf{w}, K)$.
  5:       Update the kernel matrix, K.
  6:     **end for**
  7: **until** Convergence
output Optimal θ^* and $\mathbf{w}^*$

Algorithm 2 Semi-stochastic asynchronous gradient descent with delayed kernel updates.

---

input Data – $(\mathbf{X}, \mathbf{y})$, kernel – $k_\theta(\cdot, \cdot)$, recurrent transformation – $\phi_\mathbf{w}(\cdot)$.

  1: Initialize θ and $\mathbf{w}$; compute initial K.
  2: **repeat**
  3:     $\theta \leftarrow \theta + \text{update}_\theta(\mathbf{X}, \mathbf{w}, K)$.
  4:     **for all** mini-batches $\mathbf{X}_b$ in $\mathbf{X}$ **do**
  5:       $\mathbf{w} \leftarrow \mathbf{w} + \text{update}_\mathbf{w}(\mathbf{X}_b, \mathbf{w}, K^{\text{stale}})$.
  6:     **end for**
  7:     Update the kernel matrix, K.
  8: **until** Convergence
output Optimal θ^* and $\mathbf{w}^*$

Theorem 1 (informal) *Semi-stochastic alternating gradient descent converges to a fixed point when the learning rate, λ_t, decays as $\Theta(1/t^{\frac{1+\delta}{2}})$ for any $\delta \in (0, 1]$.*

Applying alternating gradient to our case has a catch: the kernel matrix (and its inverse) has to be updated each time $\mathbf{w}$ and θ are changed, *i.e.*, on every mini-batch iteration (marked red in Algorithm 1). Computationally, this updating strategy defeats the purpose of stochastic gradients because we have to use the entire data on each step. To deal with the issue of computational efficiency, we use ideas from asynchronous optimization.

Asynchronous techniques. One of the recent trends in parallel and distributed optimization is applying updates in an asynchronous [Agarwal and Duchi, 2011] fashion. Such strategies naturally require some tolerance to delays in parameter updates [Langford et al., 2009]. In our case, we modify Algorithm 1 to allow *delayed kernel matrix updates*.

The key observation is very intuitive: when the stochastic updates of $\mathbf{w}$ are small enough, K does not change much between mini-batches, and hence we can perform multiple stochastic steps for $\mathbf{w}$ before re-computing the kernel matrix, K, and still converge. For example, K may be updated once at the end of each pass through the entire data (see Algorithm 2). To ensure convergence of the algorithm, it is important to strike the balance between (a) the learning rate for $\mathbf{w}$ and (b) the frequency of the kernel matrix updates. The following theorem provides convergence results under certain conditions.

Theorem 2 (informal) *Semi-stochastic gradient descent with τ-delayed kernel updates converges to a fixed point when the learning rate, λ_t, decays as $\Theta(1/\tau t^{\frac{1+\delta}{2}})$ for any $\delta \in (0, 1]$.*

Formal statements, conditions, and proofs for Theorems 1 and 2 are given in Appendix C.2.

Why stochastic optimization? GPs with recurrent kernels can be also trained with full-batch gradient descent, as proposed by Wilson et al. [2016]. However, stochastic gradient methods have been proved to attain better generalization [Hardt et al., 2015] and are often demonstrate superior performance in deep and recurrent architectures [Wilson and Martinez, 2003]. Moreover, stochastic methods are 'online' and can scale to large datasets. In our experiments, we demonstrate that GPs with recurrent kernels trained with Algorithm 2 converge faster and attain better performance than if trained with full-batch techniques.

Table 1: Statistics for the data used in experiments. SNR was determined by assuming a certain degree of smoothness of the signal, fitting kernel ridge regression with RBF kernel to predict the targets from the input time series, and regarding the residuals as the noise. Tasks with low average correlation between inputs and targets and lower SNR are naturally harder prediction problems.

Dataset	Task	# time steps	# dim	# outputs	Abs. corr.	SNR
Drives	system ident.	500	1	1	0.7994	25.72
Actuator		1,024	1	1	0.0938	12.47
GEF	power load	38,064	11	1	0.5147	89.93
	wind power	130,963	16	1	0.1731	4.06
Car	speed		6	1	0.1196	159.33
	gyro yaw		6	1	0.0764	3.19
	lane sequence	932,939	26	16	0.0816	—
	lead vehicle pos.		9	2	0.1099	—

Stochastic variational inference. Stochastic variational inference (SVI) in Gaussian processes [Hensman et al., 2013] is another viable approach to enabling stochastic optimization for GPs with recurrent kernels. Such method would optimize a variational lower bound on the original objective that factorizes over the data by construction. Note that unlike all previous existing work, our proposed approach does not require a variational approximation to the marginal likelihood to perform mini-batch training of Gaussian processes.

4.2 Scalability

Training and inference in Gaussian processes requires inverting the kernel matrix, K, which scales as $\mathcal{O}(n^3)$ for n training data points. For scalability, we replace all instances of the covariance matrix K with $WK_{U,U}W^\top$ [Wilson and Nickisch, 2015, Wilson et al., 2015], where W is a sparse interpolation matrix, and $K_{U,U}$ is the recurrent covariance matrix evaluated over m latent inducing points, which decomposes into a Kronecker product of circulant matrices. Inference and learning are then $\mathcal{O}(n)$ and test predictions are $\mathcal{O}(1)$, while preserving model structure. For completeness, we provide an overview of the underlying algebraic machinery in Appendix A.

5 Experiments

We compare the proposed Gaussian processes with recurrent kernels based on RNN and LSTM architectures (GP-RNN/LSTM) with a number of baselines on datasets of various complexity and ranging in size from hundreds to almost a million of time points. For the datasets with more than a few thousand points, we use a massively scalable version of GPs (see Section 4.2) and demonstrate its scalability during inference and learning. We carry out a number of experiments that help to gain empirical insights about the convergence properties of the proposed optimization procedure with delayed kernel updates. Additionally, we analyze the regularization properties of GP-RNN/LSTM and compare them with other techniques, such as dropout. Finally, we apply the model to the problem of lane estimation and lead vehicle position prediction, both critical in autonomous driving applications.
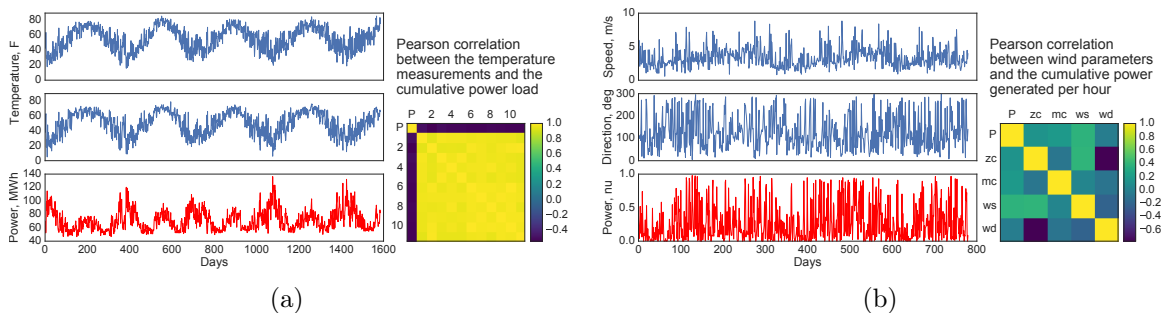
Figure 2: (a) Visualization of the *GEF-power* time series for two zones and the cumulative load with the time resolution of 1 day. Cumulative power load is generally negatively correlated with the temperature measurements on all the zones. (b) Visualization of the *GEF-wind* time series with the time resolution of 1 day.

5.1 Data and the setup

Below, we describe each dataset we used in our experiments and the associated prediction tasks. Essential statistics for the datasets are summarized in Table 1.

System identification. In some of the experiments, we used publicly available nonlinear system identification datasets: *Actuator*[5] [Sjöberg et al., 1995] and *Drives*[6] [Wigren, 2010]. Both datasets had one dimensional input and output time series. *Actuator* had the size of the valve opening as the input and the resulting change in oil pressure as the output. *Drives* is from a system with two electric motors that drive a pulley using a flexible belt; the input is the sum of voltages applied to the motors and the output is the speed of the belt.

Smart grid data[7]**.** We considered the problem of forecasting for the smart grid that consisted of two tasks. The first task was to predict power load from the historical temperature data (Figure 2a). The data had 11 input time series coming from hourly measurements of temperature on 11 zones and an output time series that represented the cumulative hourly power load on a U.S. utility. The second task was to predict power generated by wind farms from the wind forecasts (Figure 2b). The data consisted of 4 different hourly forecasts of the wind and hourly values of the generated power by a wind farm. Each wind forecast was a 4-element vector that corresponds to *zonal component*, *meridional component*, *wind speed* and *wind angle*. In our experiments, we concatenated the 4 different 4-element forecasts, which resulted in a 16-dimensional input time series.

Self-driving car dataset[8]**.** One of the main target applications of the proposed model is autonomous driving. We considered a large dataset coming from sensors of a self-driving car that was recorded on two car trips with discretization of $10\,ms$. The data featured two sets of GPS ECEF locations, ECEF velocities, measurements from a fiber-optic gyro compass, LIDAR, and a few more time series from a variety of IMU sensors. Additionally, locations of the left and right lanes were extracted from a video stream for each time step. We consider the data from the first trip for training and from the second trip for validation and testing. A

---

[5] http://www.iau.dtu.dk/nnbook/systems.html

[6] http://www.it.uu.se/research/publications/reports/2010-020/NonlinearData.zip.

[7] The smart grid data were taken from Global Energy Forecasting Kaggle competitions organized in 2012.

[8] The dataset is proprietary. It was released in part for public use under the Creative Commons Attribution 3.0 license: http://archive.org/details/comma-dataset.
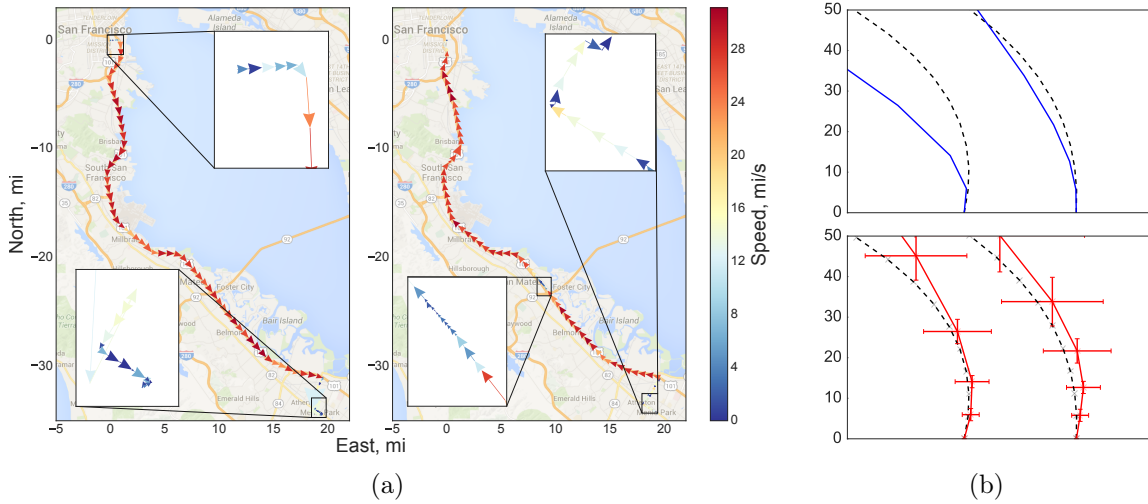
Figure 3: (a) Train and test routes of the self-driving car in the ENU coordinates with the origin at the starting location. Arrows point in the direction of motion; color encodes the speed. Insets zoom selected regions of the routes. Best viewed in color. (b) Point-wise predictions of the lanes made by LSTM (upper) and by GP-LSTM (lower). Dashed lines correspond to the ground truth; solid lines are predictions. Error-bars denote one standard deviation.

visualization of the car routes with 25 second discretization in the ENU coordinates are given in Figure 3a. We consider four tasks, the first two of which are more of proof-of-concept type variety, while the final two are fundamental to good performance for a self-driving car:

1. Speed prediction from noisy GPS velocity estimates and gyroscopic inputs.

2. Prediction of the angular acceleration of the car from the estimates of its speed and steering angle.

3. Point-wise prediction of the lanes from the estimates at the previous time steps, and estimates of speed, gyroscopic and compass measurements.

4. Prediction of the lead vehicle location from its location at the previous time steps, and estimates of speed, gyroscopic and compass measurements.

We provide more specific details on the smart grid data and self-driving data in Appendix D.

Metrics and baselines. We used a number of classical baselines: NARX [Lin et al., 1996], GP-NARX models [Kocijan et al., 2005], and classical RNN and LSTM architectures. The kernels of our models, GP-RNN and GP-LSTM, used the ARD base kernel and were structured by RNN and LSTM baselines. Our comparison also includes performance of the RGP model [Mattos et al., 2015] on the system identification data. As the primary metric, we used root mean squared error (RMSE) on a held out set and additionally negative log marginal likelihood (NLML) on the training set for the GP-based models.

Implementation. Recurrent parts of each model were implemented using *Keras*[9] library. We extended *Keras* with the Gaussian process layer and developed a backed engine based on the *GPML* library[10]. Our approach allows us to take full advantage of the functionality

---

[9]http://www.keras.io
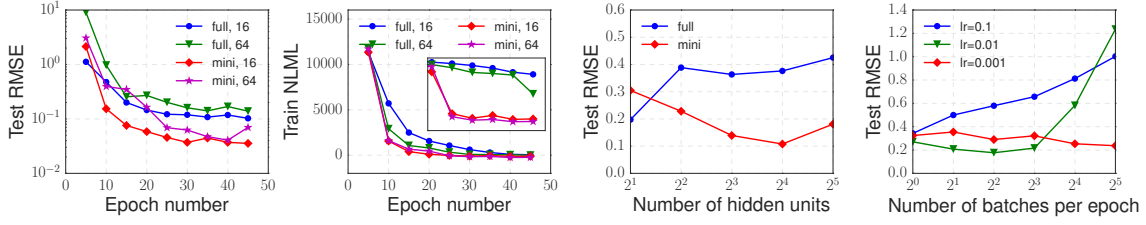[10]http://www.gaussianprocess.org/gpml/code/matlab/doc/

Figure 4: *Two charts on the left:* Convergence of the optimization in terms of RMSE on test and NLML on train. The inset zooms the region of the plot right beneath it using log scale for the vertical axis. *full* and *mini* denote full-batch and mini-batch optimization procedures, respectively, while 16 and 64 refer to models with the respective number of units per hidden layer. *Two charts on the right:* Test RMSE error for a given architecture trained with a specified method and/or learning rate.

available in *Keras* and *GPML*, *e.g.*, use automatic differentiation for the recurrent part of the model. Our code is publicly available at `http://github.com/alshedivat/kgp/`.

5.2 Analysis

This section discusses quantitative and qualitative experimental results. Additional details on all the architectures used in our experiments are given in Appendix E.

5.2.1 Convergence of the optimization

To address the question of whether stochastic optimization of GPs with recurrent kernels is necessary and to assess the behavior of the proposed optimization scheme with delayed kernel updates, we performed a number of experiments on the Actuator dataset. The results are presented on Figure 4.

First, we constructed two GP-LSTM models with 1 recurrent hidden layer and 16 or 64 hidden units and trained them with (non-stochastic) full-batch iterative procedure (similar to the proposal of Wilson et al. [2016]) and with our semi-stochastic optimizer with delayed kernel updates (Algorithm 2). The convergence results are given on the first two charts. Both in terms of the error on a held out set and the NLML on the training set, the models trained with mini-batches converged faster and demonstrated better final performance.

Next, we compared the two optimization schemes on the same GP-LSTM architecture with different sizes of the hidden layer ranging from 2 to 32. It is clear from the third chart that, even though full-batch approach seemed to find a better optimum when the number of hidden units was small, the stochastic approach was clearly superior for larger hidden layers.

Finally, we compared the behavior of Algorithm 2 with different number of mini-batches used for each epoch (equivalently, the number of steps between the kernel matrix updates) and different learning rates, and the results are give on the last chart. As expected, there is a fine balance between the number of mini-batches and the learning rate: if the number of mini-batches is large (*i.e.*, the delay between the kernel updates becomes too long) while the learning rate is high enough, optimization does not converge; at the same time, an appropriate combination of the learning rate and the mini-batch size leads better generalization than the default batch approach of Wilson et al. [2016].

Table 2: Performance of the models in terms of RMSE. Results for recurrent Gaussian processes (RGP) are based on [Mattos et al., 2015] and were available only for the system identification data.

Data	Task	NARX	RNN	LSTM	RGP	GP-NARX	GP-RNN	GP-LSTM
Drives *Actuator*	system ident.	0.423 0.482	0.408 0.771	0.382 0.381	0.249 0.368	0.403 0.891	0.332 0.492	**0.225** **0.347**
GEF	power pred. wind est.	0.529 0.837	0.622 0.818	0.465 0.781	— —	0.780 0.835	0.242 0.792	**0.158** **0.764**
Car	speed gyro yaw lane seq. lead vehicle pos.	0.114 0.189 0.128 0.410	0.152 0.223 0.331 0.452	0.027 0.121 0.078 0.400	— — — —	0.125 0.242 0.101 0.341	0.088 0.238 0.472 0.412	**0.019** **0.076** **0.055** **0.312**

5.2.2 Comparison with RGP

In this set of experiments, our main goal was to provide a comparison with the most recent RGP approach on Actuator and Drives datasets. RGP architecture is the classical RNN with every parametric layer substituted with a Gaussian process. We replicated the experimental setup of [Mattos et al., 2015] and additionally trained a number of other baselines. The results are given in the first section of Table 2. GP-based architectures consistently yielded improved predictive performance compared to their vanilla deep learning counterparts. Given the small size of the datasets, we could attribute such behavior to better regularization properties of the negative log marginal likelihood as a loss function. We also found out that when GP-based models were initialized with weights of pre-trained neural networks, they tended to overfit and give overly confident predictions on these tasks. The best performance was achieved when the models were trained from a random initialization.

5.2.3 Prediction for smart grid and self-driving car applications

For both smart grid prediction tasks we used LSTM and GP-LSTM models with 48 hour time lags and were predicting the target values one hour ahead. LSTM and GP-LSTM were trained with one or two layers and 32 to 256 hidden units per layer and batch normalization. The best models were selected on 25% of the training data used for validation. For autonomous driving prediction tasks, we used the same architectures but with 128 time steps of lag (1.28 s). Dropout [Srivastava et al., 2014] and batch normalization [Ioffe and Szegedy, 2015] were used to additionally regularize both neural networks and GPs with recurrent kernels. On both GEF and self-driving car datasets, we used the scalable version of Gaussian process (MSGP) [Wilson et al., 2015]. Given the scale of the data and the challenge of nonlinear optimization of the recurrent models, we initialized the recurrent parts of GP-RNN and GP-LSTM with pre-trained weights of the corresponding neural networks. Fine-tuning of the models was performed with Algorithm 2. The quantitative results are provided in the second and third sections of Table 2 and demonstrate that GPs with recurrent kernels attained the state-of-the-art performance on these tasks.

Additionally, we investigated convergence and regularization properties of LSTM and GP-LSTM models on the GEF-power dataset. The first two charts of Figure 5 demonstrate that GP-based models are less prone to overfitting, even when the data is not enough. The
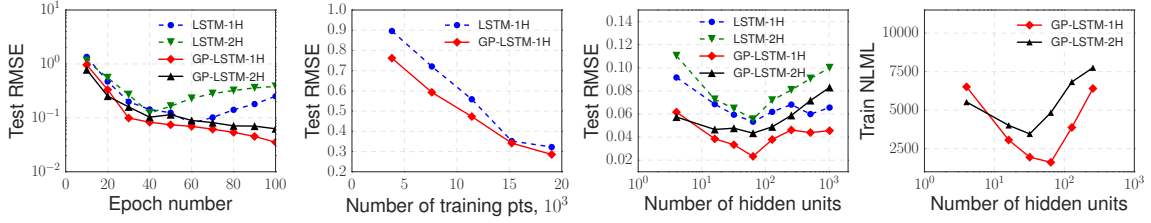
Figure 5: *Left to right*: RMSE vs. the number of training points; RMSE vs. the number model parameters per layer; NLML vs. the number model parameters per layer for GP-based models. All metrics are computed on a held-out set.
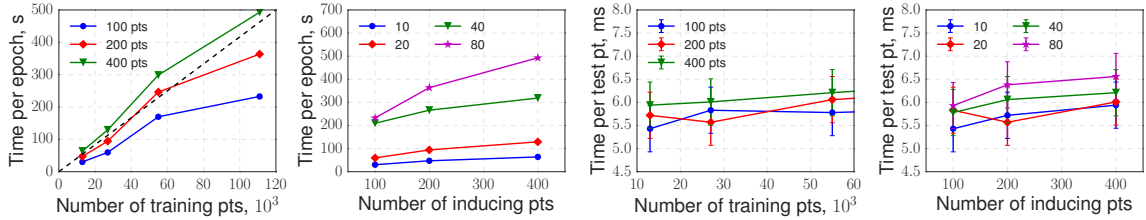

Figure 6: The charts demonstrate scalability of learning and inference of MSGP with an LSTM-based recurrent kernel. Legends with points denote the number of inducing points used. Legends with percentages denote the percentage of the training dataset used learning the model.

third panel shows that architectures with a particular number of hidden units per layer attain the best performance on the power prediction task. The advantage of the GPs over the standard recurrent networks is that the best architecture could be identified solely based on the negative log likelihood of the model as shown on the last chart.

Finally, Figure 3b qualitatively demonstrates the difference between point-wise prediction of lanes with LSTM and GP-LSTM models. GP-LSTM not only provides a more robust fit, but also estimates the uncertainty of its predictions. Such information can be further used in downstream prediction-based decision making, *e.g.*, such as whether a self-driving car should slow down and switch to a more cautious driving style when the uncertainty is too high.

5.2.4   Scalability of the model

Following Wilson et al. [2015], we performed a generic scalability analysis of the MSGP-LSTM model on the car sensors data. The LSTM architecture was the same as described in the previous section: it was transforming multi-dimensional sequences of inputs to a two-dimensional representation. We trained the model for 10 epochs on 10%, 20%, 40%, and 80% of the training set with 100, 200, and 400 inducing points per dimension and measured the average training time per epoch and the average prediction time per testing point. The measured time was the total time spent on both LSTM optimization and MSGP computations. The results are presented in Figure 6.

The training time per epoch (one full pass through the entire training data) grows linearly with the number of training examples and depends linearly on the number of inducing points (Figure 6, two left charts). Thus, given a fixed number of inducing points per dimension, the

14

time complexity of MSGP-LSTM learning and inference procedures is linear in the number of training examples. The prediction time per testing data point is virtually constant and does not depend on neither on the number of training points, nor on the number of inducing points (Figure 6, two right charts).

6   Conclusion

We proposed a method for learning kernels with recurrent long short-term memory structure on sequences. Gaussian processes with such kernels, termed the GP-LSTM, have the structure and learning biases of LSTMs, while retaining a probabilistic Bayesian nonparametric representation. Not only does GP-LSTM attain the highest performance on a number of sequence-to-sequence regression tasks, it works on data with low and high signal-to-noise ratio, can be scaled to very large datasets, all with a straightforward, practical, and generally applicable model specification. In short, the GP-LSTM provides an elegant way to quantify predictive uncertainty while using the standard deep learning toolbox. Predictive uncertainty is of high value and importance in robotics applications, such as autonomous driving, and could also be applied to other areas such as financial modeling and computational biology.

Additionally, we proposed a semi-stochastic iterative scheme with delayed kernel updates for training GP-LSTM models. We have shown that the procedure is convergent in theory and is efficient in practical settings. Our optimization approach is reminiscent of the ideas from the asynchronous stochastic optimization literature and holds a promise to allow kernel learning in a truly distributed fashion, which avoids strong independence assumptions of the previous work [Deisenroth and Ng, 2015]. We envision that by combining provably convergent asynchronous techniques with stochastic variational inference for learning kernels of scalable structure, Gaussian processes may continue to grow in their relevance to modern machine learning approaches, harmonizing with the progress in deep learning. For now, whether it is possible to train a Gaussian process in a completely data- and/or model-parallel regime with minimum synchronization and with convergence guarantees is an interesting open question that we leave for future research.

7   Acknowledgements

References

A. Agarwal and J. C. Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems*, pages 873–881, 2011.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.

D. P. Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999.

G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 1994.

A. C. Damianou and N. D. Lawrence. Deep gaussian processes. In *AISTATS*, pages 207–215, 2013.

M. P. Deisenroth and J. W. Ng. Distributed gaussian processes. In *International Conference on Machine Learning (ICML)*, volume 2, page 5, 2015.

R. Frigola, Y. Chen, and C. Rasmussen. Variational gaussian process state-space models. In *Advances in Neural Information Processing Systems*, pages 3680–3688, 2014.

Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 1050–1059, 2016.

A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE, 2013.

M. Hardt, B. Recht, and Y. Singer. Train faster, generalize better: Stability of stochastic gradient descent. *arXiv preprint arXiv:1509.01240*, 2015.

J. Hensman, N. Fusi, and N. D. Lawrence. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*, 2013.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

T. S. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. *Advances in neural information processing systems*, pages 487–493, 1999.

R. Keys. Cubic convolution interpolation for digital image processing. *IEEE transactions on acoustics, speech, and signal processing*, 29(6):1153–1160, 1981.

J. Kocijan, A. Girard, B. Banko, and R. Murray-Smith. Dynamic systems identification with gaussian processes. *Mathematical and Computer Modelling of Dynamical Systems*, 11(4): 411–424, 2005.

J. Langford, A. Smola, and M. Zinkevich. Slow learners are fast. *arXiv preprint arXiv:0911.0491*, 2009.

T. Lin, B. G. Horne, P. Tiňo, and C. L. Giles. Learning long-term dependencies in narx recurrent neural networks. *Neural Networks, IEEE Transactions on*, 7(6):1329–1338, 1996.

C. L. C. Mattos, Z. Dai, A. Damianou, J. Forth, G. A. Barreto, and N. D. Lawrence. Recurrent gaussian processes. *arXiv preprint arXiv:1511.06644*, 2015.

C. A. Micchelli, Y. Xu, and H. Zhang. Universal kernels. *The Journal of Machine Learning Research*, 7:2651–2667, 2006.

R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

J. Quinonero-Candela and C. E. Rasmussen. A unifying view of sparse approximate gaussian process regression. *The Journal of Machine Learning Research*, 6:1939–1959, 2005.

C. E. Rasmussen and Z. Ghahramani. Occam's razor. *Advances in neural information processing systems*, pages 294–300, 2001.

B. Schölkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P.-Y. Glorennec, H. Hjalmarsson, and A. Juditsky. Nonlinear black-box modeling in system identification: a unified overview. *Automatica*, 31(12):1691–1724, 1995.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

M. Stein. Interpolation of spatial data, 1999.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

R. D. Turner, M. P. Deisenroth, and C. E. Rasmussen. State-space inference and learning with gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 868–875, 2010.

P. Van Overschee and B. De Moor. *Subspace identification for linear systems: Theory — Implementation — Applications*. Springer Science & Business Media, 2012.

J. Wang, A. Hertzmann, and D. M. Blei. Gaussian process dynamical models. In *Advances in neural information processing systems*, pages 1441–1448, 2005.

T. Wigren. Input-output data sets for development and benchmarking in nonlinear identification. *Technical Reports from the department of Information Technology*, 20:2010–020, 2010.

C. K. Williams and C. E. Rasmussen. *Gaussian processes for machine learning*. the MIT Press, 2006.

A. G. Wilson and R. P. Adams. Gaussian process kernels for pattern discovery and extrapolation. *arXiv preprint arXiv:1302.4245*, 2013.

A. G. Wilson and H. Nickisch. Kernel interpolation for scalable structured gaussian processes (kiss-gp). *arXiv preprint arXiv:1503.01057*, 2015.

A. G. Wilson, C. Dann, and H. Nickisch. Thoughts on massively scalable gaussian processes. *arXiv preprint arXiv:1511.01870*, 2015.

A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing. Deep kernel learning. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, 2016.

D. R. Wilson and T. R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, 2003.

Y. Xu and W. Yin. Block stochastic gradient iteration for convex and nonconvex optimization. *SIAM Journal on Optimization*, 25(3):1686–1716, 2015.

A   Massively scalable Gaussian processes

Massively scalable Gaussian processes (MSGP) [Wilson et al., 2015] is a significant extension of the kernel interpolation framework originally proposed by Wilson and Nickisch [2015]. The core idea of the framework is to improve scalability of the inducing point methods [Quinonero-Candela and Rasmussen, 2005] by (1) placing the virtual points on a regular grid, (2) exploiting the resulting Kronecker and Toeplitz structures of the relevant covariance matrices, and (3) do local cubic interpolation to go back to the kernel evaluated at the original points. This combination of techniques brings the complexity down to $\mathcal{O}(n)$ for training and $\mathcal{O}(1)$ for each test prediction. Below, we overview the methodology. We remark that a major difference in philosophy between MSGP and many classical inducing point method is that the points are *select and fixed* rather than optimized over. This allows to use a much higher number of virtual points which typically results in a better approximation of the true kernel.

A.1   Structured kernel interpolation

Given a set of m inducing points, the $n \times m$ cross-covariance matrix, $K_{X,U}$, between the training inputs, X, and the inducing points, U, can be approximated as $\tilde{K}_{X,U} = W_X K_{U,U}$ using a (potentially sparse) $n \times m$ matrix of interpolation weights, W_X. This allows to approximate $K_{X,Z}$ for an arbitrary set of inputs Z as $K_{X,Z} \approx \tilde{K}_{X,U} W_Z^\top$. For any given kernel function, K, and a set of inducing points, U, *structured kernel interpolation* (SKI) procedure [Wilson and Nickisch, 2015] gives rise to the following approximate kernel:

$$k_{\text{SKI}}(\mathbf{x}, \mathbf{z}) = \mathbf{w}_x K_{U,U} \mathbf{w}_z^\top, \tag{A.1}$$

which allows to approximate $K_{X,X} \approx W_X K_{U,U} W_X^\top$. Wilson and Nickisch [2015] note that standard inducing point approaches, such as subset of regression (SoR) or fully independent training conditional (FITC), can be reinterpreted from the SKI perspective. Importantly, the efficiency of SKI-based MSGP methods comes from, first, a clever choice of a set of inducing points that allows to exploit algebraic structure of $K_{U,U}$, and second, from using very sparse local interpolation matrices. In practice, local cubic interpolation is used [Keys, 1981].

A.2   Kernel approximations

If inducing points, U, form a regularly spaced P-dimensional grid, and we use a stationary product kernel (e.g., the RBF kernel), then $K_{U,U}$ decomposes as a Kronecker product of Toeplitz matrices:

$$K_{U,U} = T_1 \otimes T_2 \otimes \cdots \otimes T_P. \tag{A.2}$$

The Kronecker structure allows to compute the eigendecomposition of $K_{U,U}$ by separately decomposing $T_1, \ldots, T_P$, each of which is much smaller than $K_{U,U}$. Further, to efficiently eigendecompose a Toeplitz matrix, it can be approximated by a circulant matrix[11] which eigendecomposes by simply applying discrete Fourier transform (DFT) to its first column. Therefore, an approximate eigendecomposition of each $T_1, \ldots, T_P$ is computed via the fast Fourier transform (FFT) and requires only $\mathcal{O}(m \log m)$ time.

---

[11] Wilson et al. [2015] explored 5 different approximation methods known in the numerical analysis literature.

A.3 Structure exploiting inference

To perform inference, we need to solve $(K_{\text{SKI}} + \sigma^2 I)^{-1}\mathbf{y}$; kernel learning requires evaluating $\log \det(K_{\text{SKI}} + \sigma^2 I)$. The first task can be accomplished by using an iterative scheme—linear conjugate gradients—which depends only on matrix vector multiplications with $(K_{\text{SKI}} + \sigma^2 I)$. The second is done by exploiting the Kronecker and Toeplitz structure of $K_{U,U}$ for computing an approximate eigendecomposition, as described above.

A.4 Fast Test Predictions

To achieve constant time prediction, we approximate the latent mean and variance of $\mathbf{f}_*$ by applying the same SKI technique. In particular, for a set of n_* testing points, X_*, we have

$$
\begin{aligned}
\mathbb{E}[\mathbf{f}_*] &= \mu_{X_*} + K_{X_*,X}\left[K_{X,X} + \sigma^2 I\right]^{-1}\mathbf{y} \\
&\approx \mu_{X_*} + \tilde{K}_{X_*,X}\left[\tilde{K}_{X,X} + \sigma^2 I\right]^{-1}\mathbf{y},
\end{aligned}
\tag{A.3}
$$

where $\tilde{K}_{X,X} = WK_{U,U}W^\top$ and $\tilde{K}_{X_*,X} = W_*K_{U,U}W^\top$, and W and W_* are $n \times m$ and $n_* \times m$ sparse interpolation matrices, respectively. Since $K_{U,U}W^\top[\tilde{K}_{X,X} + \sigma^2 I]^{-1}\mathbf{y}$ is precomputed at training time, at test time, we only multiply the latter with W_* matrix which results which costs $\mathcal{O}(n_*)$ operations leading to $\mathcal{O}(1)$ operations per test point. Similarly, approximate predictive variance can be also estimated in $\mathcal{O}(1)$ operations [Wilson et al., 2015].

Note that the fast prediction methodology can be readily applied to any trained Gaussian process model as it is agnostic to the way inference and learning were performed.

B Gradients for GPs with recurrent kernels

GPs with deep recurrent kernels are trained by minimizing the negative log marginal likelihood objective function. Below we derive the update rules.

By applying the chain rule, we get the following first order derivatives:

$$
\frac{\partial\mathcal{L}}{\partial\gamma} = \frac{\partial\mathcal{L}}{\partial K}\cdot\frac{\partial K}{\partial\gamma}, \quad \frac{\partial\mathcal{L}}{\partial\mathbf{w}} = \frac{\partial\mathcal{L}}{\partial K}\cdot\frac{\partial K}{\partial\phi}\cdot\frac{\partial\phi}{\partial\mathbf{w}}.
\tag{B.4}
$$

The derivative of the log marginal likelihood w.r.t. to the kernel hyperparameters, θ, and the parameters of the recurrent map, $\mathbf{w}$, are generic and take the following form [Williams and Rasmussen, 2006, Ch. 5, Eq. 5.9]:

$$
\frac{\partial\mathcal{L}}{\partial\theta} = \tfrac{1}{2}\operatorname{tr}\left[\left(K^{-1}\mathbf{y}\mathbf{y}^\top K^{-1} - K^{-1}\right)\tfrac{\partial K}{\partial\theta}\right],
\tag{B.5}
$$

$$
\frac{\partial\mathcal{L}}{\partial\mathbf{w}} = \tfrac{1}{2}\operatorname{tr}\left[\left(K^{-1}\mathbf{y}\mathbf{y}^\top K^{-1} - K^{-1}\right)\tfrac{\partial K}{\partial\mathbf{w}}\right].
\tag{B.6}
$$

The derivative $\partial K/\partial\theta$ is also standard and depends on the form of a particular chosen kernel function, $k(\cdot,\cdot)$. However, computing each part of $\partial\mathcal{L}/\partial\mathbf{w}$ is a bit subtle, and hence we elaborate these derivations below.

Consider the ij-th entry of the kernel matrix, K_{ij}. We can think of K as a matrix-valued function of all the data vectors in d-dimensional transformed space which we denote by

$H \in \mathbb{R}^{N \times d}$. Then K_{ij} is a scalar-valued function of H and its derivative w.r.t. the l-th parameter of the recurrent map, w_l, can be written as follows:

$$\frac{\partial K_{ij}}{\partial w_l} = \text{tr} \left[\left(\frac{\partial K_{ij}}{\partial H} \right)^\top \frac{\partial H}{\partial w_l} \right]. \tag{B.7}$$

Notice that $\partial K_{ij}/\partial H$ is a derivative of a scalar w.r.t. to a matrix and hence is a matrix; $\partial H/\partial w_l$ is a derivative of a matrix w.r.t. to a scalar which is taken element-wise and also gives a matrix. Also notice that K_{ij} is a function of H, but it only depends the i-th and j-th elements for which the kernel is being computed. This means that $\partial K_{ij}/\partial H$ will have only non-zero i-th row and j-th column and allows us to re-write (B.7) as follows:

$$\begin{aligned}
\frac{\partial K_{ij}}{\partial w_l} &= \left(\frac{\partial K_{ij}}{\partial \mathbf{h}_i} \right)^\top \frac{\partial \mathbf{h}_i}{\partial w_l} + \left(\frac{\partial K_{ij}}{\partial \mathbf{h}_j} \right)^\top \frac{\partial \mathbf{h}_j}{\partial w_l} \\
&= \left(\frac{\partial k(\mathbf{h}_i, \mathbf{h}_j)}{\partial \mathbf{h}_i} \right)^\top \frac{\partial \mathbf{h}_i}{\partial w_l} + \left(\frac{\partial k(\mathbf{h}_i, \mathbf{h}_j)}{\partial \mathbf{h}_j} \right)^\top \frac{\partial \mathbf{h}_j}{\partial w_l}.
\end{aligned} \tag{B.8}$$

Since the kernel function has two arguments, the derivatives must be taken with respect of each of them and evaluated at the corresponding points in the hidden space, $\mathbf{h}_i = \phi(\mathbf{x}_i)$ and $\mathbf{h}_j = \phi(\mathbf{x}_j)$. When we plug this into (B.6), we arrive at the following expression:

$$\frac{\partial \mathcal{L}}{\partial w_l} = \frac{1}{2} \sum_{i,j} \left(K^{-1} \mathbf{y} \mathbf{y}^\top K^{-1} - K^{-1} \right)_{ij} \left\{ \left(\frac{\partial k(\mathbf{h}_i, \mathbf{h}_j)}{\partial \mathbf{h}_i} \right)^\top \frac{\partial \mathbf{h}_i}{\partial w_l} + \left(\frac{\partial k(\mathbf{h}_i, \mathbf{h}_j)}{\partial \mathbf{h}_j} \right)^\top \frac{\partial \mathbf{h}_j}{\partial w_l} \right\}. \tag{B.9}$$

The same expression can be written in a more compact form using the *Einstein notation*:

$$\frac{\partial \mathcal{L}}{\partial w_l} = \frac{1}{2} \left(K^{-1} \mathbf{y} \mathbf{y}^\top K^{-1} - K^{-1} \right)_i^j \left(\left[\frac{\partial K}{\partial \mathbf{h}} \right]_i^{jd} + \left[\frac{\partial K}{\partial \mathbf{h}} \right]_j^{id} \right) \left[\frac{\partial \mathbf{h}}{\partial \mathbf{w}} \right]_i^{dl} \tag{B.10}$$

where d indexes the dimensions of the $\mathbf{h}$ and l indexes the dimensions of $\mathbf{w}$.

In practice, deriving a computationally efficient analytical form of $\partial K/\partial \mathbf{h}$ might be too complicated for some kernels (*e.g.*, the spectral mixture kernels [Wilson and Adams, 2013]), especially if grid-based approximations of the kernel are enabled. In such cases, we can simply use a finite difference approximation of this derivative. As we remark in the following section, numerical errors that result from this approximation do not affect convergence of the algorithm.

C Convergence results

Convergence results for the semi-stochastic alternating gradient schemes with and without delayed kernel matrix updates are based on [Xu and Yin, 2015]. There are a few notable differences between the original setting and the one considered in this paper:

1. Xu and Yin [2015] consider a stochastic program that minimizes the expectation of the objective w.r.t. some distribution underlying the data:

$$\underset{x \in \mathcal{X}}{\text{minimize}} \, f(x) := \mathbb{E}_\xi F(x; \xi), \tag{C.11}$$

where every iteration a new ξ is sampled from the underlying distribution. In our case, the goal is to minimize the negative log marginal likelihood on a particular given dataset. This is equivalent to the original formulation (C.11), but with the expectation taken w.r.t. the empirical distribution that corresponds to the given dataset.

2. The optimization procedure of Xu and Yin [2015] has access to only a single random point generated from the data distribution at each step. Our algorithm requires having access to the entire training data each time the kernel matrix is computed.

3. For a given sample, Xu and Yin [2015] propose to loop over a number of coordinate blocks and apply Gauss–Seidel type gradient updates to each block. Our semi-stochastic scheme has only two parameter blocks, θ and $\mathbf{w}$, where θ is updated deterministically on the entire dataset while $\mathbf{w}$ is updated with stochastic gradient on samples from the empirical distribution.

Noting these differences, we first adapt convergence results for the smooth non-convex case [Xu and Yin, 2015, Theorem 2.10] to our scenario, and then consider the variant with delaying kernel matrix updates.

C.1 Semi-stochastic alternating gradient

As shown in Algorithm 1, we alternate between updating θ and $\mathbf{w}$. At step t, we get a mini-batch of size N_t, $\mathbf{X}_t \equiv \{\bar{\mathbf{x}}_i\}_{i \in \mathcal{I}_t}$, which is just a selection of points from the full set, $\mathbf{X}$. Define the gradient errors for θ and $\mathbf{w}$ at step t as follows:

$$\delta_\theta^t := \tilde{\mathbf{g}}_\theta^t - \mathbf{g}_\theta^t, \quad \delta_\mathbf{w}^t := \tilde{\mathbf{g}}_\mathbf{w}^t - \mathbf{g}_\mathbf{w}^t, \tag{C.12}$$

where $\mathbf{g}_\theta^t$ and $\mathbf{g}_\mathbf{w}^t$ are the true gradients and $\tilde{\mathbf{g}}_\theta^t$ and $\tilde{\mathbf{g}}_\mathbf{w}^t$ are estimates[12]. We first update θ and then $\mathbf{w}$, and hence the expressions for the gradients take the following form:

$$\tilde{\mathbf{g}}_\theta^t \equiv \mathbf{g}_\theta^t = \frac{1}{N} \nabla_\theta \mathcal{L}(\theta^t, \mathbf{w}^t) = \frac{1}{2N} \sum_{i,j} \left(K_t^{-1} \mathbf{y}\mathbf{y}^\top K_t^{-1} - K_t^{-1} \right)_{ij} \left(\frac{\partial K_t}{\partial \theta} \right)_{ij} \tag{C.13}$$

$$\mathbf{g}_\mathbf{w}^t = \frac{1}{N} \nabla_\mathbf{w} \mathcal{L}(\theta^{t+1}, \mathbf{w}^t) = \frac{1}{2N} \sum_{i,j} \left(K_{t+1}^{-1} \mathbf{y}\mathbf{y}^\top K_{t+1}^{-1} - K_{t+1}^{-1} \right)_{ij} \left(\frac{\partial K_{t+1}}{\partial \mathbf{w}} \right)_{ij} \tag{C.14}$$

$$\tilde{\mathbf{g}}_\mathbf{w}^t = \frac{1}{2N_t} \sum_{i,j \in \mathcal{I}_t} \left(K_{t+1}^{-1} \mathbf{y}\mathbf{y}^\top K_{t+1}^{-1} - K_{t+1}^{-1} \right)_{ij} \left(\frac{\partial K_{t+1}}{\partial \mathbf{w}} \right)_{ij} \tag{C.15}$$

Note that as we have shown in Appendix B, when the kernel matrix is fixed, $\mathbf{g}_\mathbf{w}$ and $\tilde{\mathbf{g}}_\mathbf{w}$ factorize over $\mathbf{X}$ and $\mathbf{X}_t$, respectively. Further, we denote all the mini-batches sampled before t as $\mathbf{X}_{[t-1]}$.

Lemma 1 *For any step t, $\mathbb{E}[\delta_\mathbf{w}^t \mid \mathbf{X}_{[t-1]}] = \mathbb{E}[\delta_\theta^t \mid \mathbf{X}_{[t-1]}] = 0$.*

Proof First, $\delta_\theta^t \equiv 0$, and hence $\mathbb{E}[\delta_\theta^t \mid \mathbf{X}_{[t-1]}] = 0$ is trivial. Next, by definition of $\delta_\mathbf{w}^t$, we have $\mathbb{E}[\delta_\mathbf{w}^t \mid \mathbf{X}_{[t-1]}] = \mathbb{E}[\tilde{\mathbf{g}}_\mathbf{w}^t - \mathbf{g}_\mathbf{w}^t \mid \mathbf{X}_{[t-1]}]$. Consider the following:

---

[12]Here, we consider the gradients and their estimates scaled by the number of full data points, N, and the mini-batch size, N_t, respectively. These constant scaling is introduced for sake of having cleaner proofs.

- Consider $\mathbb{E}[\mathbf{g}_{\mathbf{w}}^t \mid \mathbf{X}_{[t-1]}]$: $\mathbf{g}_{\mathbf{w}}^t$ is a deterministic function of θ^{t+1} and $\mathbf{w}^t$. θ^{t+1} is being updated deterministically using $\mathbf{g}_\theta^{t-1}$, and hence only depends on $\mathbf{w}^t$ and θ^t. Therefore, it is independent of $\mathbf{X}_t$, which means that $\mathbb{E}[\mathbf{g}_{\mathbf{w}}^t \mid \mathbf{X}_{[t-1]}] \equiv \mathbf{g}_{\mathbf{w}}^t$.

- Now, consider $\mathbb{E}[\tilde{\mathbf{g}}_{\mathbf{w}}^t \mid \mathbf{X}_{[t-1]}]$: Noting that the expectation is taken w.r.t. the empirical distribution and K_{t+1} does not depend on the current mini-batch, we can write:

$$
\begin{aligned}
\mathbb{E}[\tilde{\mathbf{g}}_{\mathbf{w}}^t \mid \mathbf{X}_{[t-1]}] &= \mathbb{E}\left[\frac{1}{2N_t} \sum_{i,j \in \mathcal{I}_t} \left(K_{t+1}^{-1} \mathbf{y}\mathbf{y}^\top K_{t+1}^{-1} - K_{t+1}^{-1} \right)_{ij} \left(\frac{\partial K_{t+1}}{\partial \mathbf{w}} \right)_{ij} \right] \\
&= \frac{N_t}{2N N_t} \sum_{i,j} \left(K_{t+1}^{-1} \mathbf{y}\mathbf{y}^\top K_{t+1}^{-1} - K_{t+1}^{-1} \right)_{ij} \left(\frac{\partial K_{t+1}}{\partial \mathbf{w}} \right)_{ij} \\
&\equiv \mathbf{g}_{\mathbf{w}}^t.
\end{aligned}
\tag{C.16}
$$

Finally, $\mathbb{E}[\delta_{\mathbf{w}}^t \mid \mathbf{X}_{[t-1]}] = \mathbf{g}_{\mathbf{w}}^t - \mathbf{g}_{\mathbf{w}}^t = 0$. ∎

In other words, semi-stochastic gradient descent that alternates between updating θ and $\mathbf{w}$ computes unbiased estimates of the gradients on each step.

Remark 1 *Note that in case if $(\partial K_{t+1}/\partial \mathbf{w})$ is computed approximately, Lemma 1 still holds since both $\mathbf{g}_{\mathbf{w}}^t$ and $\mathbb{E}[\tilde{\mathbf{g}}_{\mathbf{w}}^t \mid \mathbf{X}_{[t-1]}]$ will contain exactly the same numerical errors.*

Assumption 1 *For any step t, $\mathbb{E}\|\delta_\theta^t\|^2 \leq \sigma_t^2$ and $\mathbb{E}\|\delta_{\mathbf{w}}^t\|^2 \leq \sigma_t^2$.*

Lemma 1 and Assumption 1 result into a stronger condition than the original assumption given by Xu and Yin [2015]. This is due to the semi-stochastic nature of the algorithm, it simplifies the analysis, though it is not critical. Assumptions 2 and 3 are straightfordwardly adapted from the original paper.

Assumption 2 *The objective function, $\mathcal{L}$, is lower bounded and its partial derivatives w.r.t. θ and $\mathbf{w}$ are uniformly Lipschitz with constant $L > 0$:*

$$
\|\nabla_\theta \mathcal{L}(\theta, \mathbf{w}) - \nabla_\theta \mathcal{L}(\tilde{\theta}, \mathbf{w})\| \leq L\|\theta - \tilde{\theta}\|, \quad \|\nabla_{\mathbf{w}} \mathcal{L}(\theta, \mathbf{w}) - \nabla_{\mathbf{w}} \mathcal{L}(\theta, \tilde{\mathbf{w}})\| \leq L\|\mathbf{w} - \tilde{\mathbf{w}}\|. \tag{C.17}
$$

Assumption 3 *There exists a constant ρ such that $\|\theta^t\|^2 + \mathbb{E}\|\mathbf{w}^t\|^2 \leq \rho^2$ for all t.*

Lemma 2 *Under Assumptions 2 and 3,*

$$
\mathbb{E}\|\mathbf{w}^t\|^2 \leq \rho^2, \quad \mathbb{E}\|\theta^t\|^2 \leq \rho^2, \quad \mathbb{E}\|\mathbf{g}_{\mathbf{w}}^t\|^2 \leq M_\rho^2, \quad \mathbb{E}\|\mathbf{g}_\theta^t\|^2 \leq M_\rho^2 \quad \forall t,
$$

where $M_\rho^2 = 4L^2\rho^2 + 2\max\{\nabla_\theta \mathcal{L}(\theta^0, \mathbf{w}^0), \nabla_{\mathbf{w}} \mathcal{L}(\theta^0, \mathbf{w}^0)\}$.

Proof The inequalities for $\mathbb{E}\|\theta\|^2$ and $\mathbb{E}\|\mathbf{w}\|^2$ are trivial, and the ones for $\mathbb{E}\|\mathbf{g}_\theta^t\|^2$ and $\mathbb{E}\|\mathbf{g}_{\mathbf{w}}^t\|^2$ are merely corollaries from Assumption 2. ∎

Negative log marginal likelihood of a Gaussian process with a structured kernel is a nonconvex function of its arguments. Therefore, we can only shown that the algorithm converges to a stationary point, *i.e.*, a point at which the gradient of the objective is zero.

Theorem 1 *Let $\{(\theta^t, \mathbf{w}^t)\}$ be a sequence generated from Algorithm 1 with learning rates for θ and $\mathbf{w}$ being $\lambda_\theta^t = c_\theta^t \alpha_t$ and $\lambda_\mathbf{w}^t = c_\mathbf{w}^t \alpha_t$, respectively, where $c_\theta^t, c_\mathbf{w}^t, \alpha_t$ are positive scalars, such that*

$$0 < \inf_t c_{\{\theta, \mathbf{w}\}}^t \leq \sup_t c_{\{\theta, \mathbf{w}\}}^t < \infty, \quad \alpha_t \geq \alpha_{t+1}, \quad \sum_{t=1}^\infty \alpha_t = +\infty, \quad \sum_{t=1}^\infty \alpha_t^2 < +\infty, \quad \forall t. \tag{C.18}$$

Then, under Assumptions 1 through 3 and if $\sigma = \sup_t \sigma_t < \infty$, we have

$$\lim_{t \to \infty} \mathbb{E}\|\nabla \mathcal{L}(\theta^t, \mathbf{w}^t)\| = 0. \tag{C.19}$$

Proof The proof is an adaptation of the one given by Xu and Yin [2015] with the following three adjustments: (1) we have only two blocks of coordinates and (2) updates for θ are deterministic which zeros out their variance terms, (3) the stochastic gradients are unbiased.

- From the Lipschitz continuity of $\nabla_\mathbf{w} \mathcal{L}$ and $\nabla_\theta \mathcal{L}$ (Assumption 2), we have:

$$
\begin{aligned}
&\mathcal{L}(\theta^{t+1}, \mathbf{w}^{t+1}) - \mathcal{L}(\theta^{t+1}, \mathbf{w}^t) \\
\leq\ & \langle \mathbf{g}_\mathbf{w}^t, \mathbf{w}^{t+1} - \mathbf{w}^t \rangle + \frac{L}{2}\|\mathbf{w}^{t+1} - \mathbf{w}^t\|^2 \\
=\ & -\lambda_\mathbf{w}^t \langle \mathbf{g}_\mathbf{w}^t, \tilde{\mathbf{g}}_\mathbf{w}^t \rangle + \frac{L}{2}(\lambda_\mathbf{w}^t)^2 \|\tilde{\mathbf{g}}_\mathbf{w}^t\|^2 \\
=\ & -\left(\lambda_\mathbf{w}^t - \frac{L}{2}(\lambda_\mathbf{w}^t)^2\right)\|\mathbf{g}_\mathbf{w}^t\|^2 + \frac{L}{2}(\lambda_\mathbf{w}^t)^2\|\delta_\mathbf{w}^t\|^2 - \left(\lambda_\mathbf{w}^t - L(\lambda_\mathbf{w}^t)^2\right)\langle \mathbf{g}_\mathbf{w}^t, \delta_\mathbf{w}^t \rangle \\
=\ & -\left(\lambda_\mathbf{w}^t - \frac{L}{2}(\lambda_\mathbf{w}^t)^2\right)\|\mathbf{g}_\mathbf{w}^t\|^2 + \frac{L}{2}(\lambda_\mathbf{w}^t)^2\|\delta_\mathbf{w}^t\|^2 \\
& - \left(\lambda_\mathbf{w}^t - L(\lambda_\mathbf{w}^t)^2\right)\left(\langle \mathbf{g}_\mathbf{w}^t - \nabla_\mathbf{w}\mathcal{L}(\theta^t, \mathbf{w}^t), \delta_\mathbf{w}^t \rangle + \langle \nabla_\mathbf{w}\mathcal{L}(\theta^t, \mathbf{w}^t), \delta_\mathbf{w}^t \rangle\right) \\
\leq\ & -\left(\lambda_\mathbf{w}^t - \frac{L}{2}(\lambda_\mathbf{w}^t)^2\right)\|\mathbf{g}_\mathbf{w}^t\|^2 + \frac{L}{2}(\lambda_\mathbf{w}^t)^2\|\delta_\mathbf{w}^t\|^2 - \left(\lambda_\mathbf{w}^t - L(\lambda_\mathbf{w}^t)^2\right)\langle \nabla_\mathbf{w}\mathcal{L}(\theta^t, \mathbf{w}^t), \delta_\mathbf{w}^t \rangle \\
& + \frac{L}{2}\lambda_\theta^t\left(\lambda_\mathbf{w}^t + L(\lambda_\mathbf{w}^t)^2\right)\left(\|\delta_\mathbf{w}^t\|^2 + \|\mathbf{g}_\theta^t\|^2\right),
\end{aligned}
$$

where the last inequality comes from the following (note that $\mathbf{g}_\mathbf{w}^t := \nabla_\mathbf{w}\mathcal{L}(\theta^{t+1}, \mathbf{w}^t)$):

$$
\begin{aligned}
& -\left(\lambda_\mathbf{w}^t - L(\lambda_\mathbf{w}^t)^2\right)\langle \nabla_\mathbf{w}\mathcal{L}(\theta^{t+1}, \mathbf{w}^t) - \nabla_\mathbf{w}\mathcal{L}(\theta^t, \mathbf{w}^t), \delta_\mathbf{w}^t \rangle \\
\leq\ & \left|\lambda_\mathbf{w}^t - L(\lambda_\mathbf{w}^t)^2\right| \|\delta_\mathbf{w}^t\| \|\nabla_\mathbf{w}\mathcal{L}(\theta^{t+1}, \mathbf{w}^t) - \nabla_\mathbf{w}\mathcal{L}(\theta^t, \mathbf{w}^t)\| \\
\leq\ & L\lambda_\theta^t \left|\lambda_\mathbf{w}^t - L(\lambda_\mathbf{w}^t)^2\right| \|\delta_\mathbf{w}^t\| \|\mathbf{g}_\theta^t\| \\
\leq\ & \frac{L}{2}\lambda_\theta^t \left(\lambda_\mathbf{w}^t + L(\lambda_\mathbf{w}^t)^2\right)\left(\|\delta_\mathbf{w}^t\|^2 + \|\mathbf{g}_\theta^t\|^2\right).
\end{aligned}
$$

Analogously, we derive a bound on $\mathcal{L}(\theta^{t+1}, \mathbf{w}^t) - \mathcal{L}(\theta^t, \mathbf{w}^t)$:

$$
\begin{aligned}
&\mathcal{L}(\theta^{t+1}, \mathbf{w}^t) - \mathcal{L}(\theta^t, \mathbf{w}^t) \\
\leq\ & -\left(\lambda_\theta^t - \frac{L}{2}(\lambda_\theta^t)^2\right)\|\mathbf{g}_\theta^t\|^2 + \frac{L}{2}(\lambda_\theta^t)^2\|\delta_\theta^t\|^2 - \left(\lambda_\theta^t - L(\lambda_\theta^t)^2\right)\langle \nabla_\theta\mathcal{L}(\theta^t, \mathbf{w}^t), \delta_\theta^t \rangle
\end{aligned}
$$

24

- Since $(\theta^t, \mathbf{w}^t)$ is independent from the current mini-batch, $\mathbf{X}_t$, using Lemma 1, we have that $\mathbb{E}\langle \nabla_\theta \mathcal{L}(\theta^t, \mathbf{w}^t), \delta_\theta^t \rangle = 0$ and $\mathbb{E}\langle \nabla_{\mathbf{w}} \mathcal{L}(\theta^t, \mathbf{w}^t), \delta_{\mathbf{w}}^t \rangle = 0$.

  Summing the two obtained inequalities and applying Assumption 1, we get:

$$
\begin{aligned}
&\mathbb{E}[\mathcal{L}(\theta^{t+1}, \mathbf{w}^{t+1}) - \mathcal{L}(\theta^t, \mathbf{w}^t)] \\
\leq\ & -\left(\lambda_{\mathbf{w}}^t - \frac{L}{2}(\lambda_{\mathbf{w}}^t)^2\right)\mathbb{E}\|\mathbf{g}_{\mathbf{w}}^t\|^2 + \frac{L}{2}(\lambda_{\mathbf{w}}^t)^2\sigma + \frac{L}{2}\lambda_\theta^t\left(\lambda_{\mathbf{w}}^t + L(\lambda_{\mathbf{w}}^t)^2\right)(\sigma + \mathbb{E}\|\mathbf{g}_\theta^t\|^2) \\
& -\left(\lambda_\theta^t - \frac{L}{2}(\lambda_\theta^t)^2\right)\mathbb{E}\|\mathbf{g}_\theta^t\|^2 + \frac{L}{2}(\lambda_\theta^t)^2\sigma \\
\leq\ & -\left(c\alpha_t - \frac{L}{2}C^2\alpha_t^2\right)\mathbb{E}\|\mathbf{g}_{\mathbf{w}}^t\|^2 + LC^2\sigma\alpha_t + \frac{L}{2}C^2\alpha_t^2\left(1 + LC\alpha_t\right)(\sigma + \mathbb{E}\|\mathbf{g}_\theta^t\|^2) \\
& -\left(c\alpha_t - \frac{L}{2}C^2\alpha_t^2\right)\mathbb{E}\|\mathbf{g}_\theta^t\|^2 + \frac{L}{2}C^2\alpha_t^2\sigma,
\end{aligned}
$$

  where we denoted $c = \min\{\inf_t c_{\mathbf{w}}^t, \inf_t c_\theta^t\}$, $C = \max\{\sup_t c_{\mathbf{w}}^t, \sup_t c_\theta^t\}$. Note that from Lemma 2, we also have $\mathbb{E}\|\mathbf{g}_\theta^t\|^2 \leq M_\rho^2$ and $\mathbb{E}\|\mathbf{g}_{\mathbf{w}}^t\|^2 \leq M_\rho^2$. Therefore, summing the right hand side of the final inequality over t and using (C.18), we have:

$$
\lim_{t\to\infty} \mathbb{E}\mathcal{L}(\theta^{t+1}, \mathbf{w}^{t+1}) - \mathbb{E}\mathcal{L}(\theta^0, \mathbf{w}^0) \leq -c\sum_{t=1}^{\infty}\alpha_t\left(\mathbb{E}\|\mathbf{g}_\theta^t\|^2 + \mathbb{E}\|\mathbf{g}_{\mathbf{w}}^t\|^2\right). \qquad \text{(C.20)}
$$

  Since the objective function is lower bounded, this effectively means:

$$
\sum_{t=1}^{\infty}\alpha_t\mathbb{E}\|\mathbf{g}_\theta^t\|^2 < \infty, \quad \sum_{t=1}^{\infty}\alpha_t\mathbb{E}\|\mathbf{g}_{\mathbf{w}}^t\|^2 < \infty.
$$

- Finally, using Lemma 2, our assumptions and Jensen's inequality, it follows that

$$
\left|\mathbb{E}\|\mathbf{g}_{\mathbf{w}}^{t+1}\|^2 - \mathbb{E}\|\mathbf{g}_{\mathbf{w}}^t\|^2\right| \leq 2LM_\rho C\alpha_t\sqrt{2(M_\rho^2 + \sigma^2)}.
$$

According to Proposition 1.2.4 of [Bertsekas, 1999], we have $\mathbb{E}\|\mathbf{g}_\theta^t\|^2 \to 0$ and $\mathbb{E}\|\mathbf{g}_{\mathbf{w}}^t\|^2 \to 0$ as $t \to \infty$, and hence

$$
\begin{aligned}
\mathbb{E}\|\nabla\mathcal{L}(\theta^t, \mathbf{w}^t)\| &\leq \mathbb{E}\|\nabla_{\mathbf{w}}\mathcal{L}(\theta^t, \mathbf{w}^t) - \mathbf{g}_\theta^t\| + \mathbb{E}\|\mathbf{g}_\theta^t\| + \mathbb{E}\|\nabla_{\mathbf{w}}\mathcal{L}(\theta^t, \mathbf{w}^t) - \mathbf{g}_\theta^t\| + \mathbb{E}\|\mathbf{g}_{\mathbf{w}}^t\| \\
&\leq 2LC\sqrt{2(M_\rho^2 + \sigma^2)}\alpha + \mathbb{E}\|\mathbf{g}_\theta^t\| + \mathbb{E}\|\mathbf{g}_{\mathbf{w}}^t\| \to 0 \text{ as } t \to \infty,
\end{aligned}
$$

where the first term of the last inequality follows from Lemma 2 and Jensen's inequality. $\blacksquare$

C.2 Semi-stochastic gradient with delayed kernel matrix updates

We show that given a bounded delay on the kernel matrix updates, the algorithm is still convergent. Our analysis is based on computing the change in δ_θ^t and $\delta_{\mathbf{w}}^t$ and applying the same argument as in Theorem 1. The only difference is that we need to take into account the perturbations of the kernel matrix due to the introduced delays, and hence we have to impose certain assumptions on its spectrum.

Assumption 4 *Recurrent transformations, $\phi_{\mathbf{w}}(\bar{\mathbf{x}})$, is L-Lipschitz w.r.t. $\mathbf{w}$ for all $\bar{\mathbf{x}} \in \mathcal{X}^L$:*

$$\|\phi_{\tilde{\mathbf{w}}}(\bar{\mathbf{x}}) - \phi_{\mathbf{w}}(\bar{\mathbf{x}})\| \leq L\|\tilde{\mathbf{w}} - \mathbf{w}\|.$$

Assumption 5 *The kernel function, $k(\cdot, \cdot)$, is uniformly G-Lipschitz and its first partial derivatives are uniformly J-Lipschitz:*

$$\left\|k(\tilde{\mathbf{h}}_1, \mathbf{h}_2) - k(\mathbf{h}_1, \mathbf{h}_2)\right\| \leq G\|\tilde{\mathbf{h}}_1 - \mathbf{h}_1\|, \ \left\|k(\mathbf{h}_1, \tilde{\mathbf{h}}_2) - k(\mathbf{h}_1, \mathbf{h}_2)\right\| \leq G\|\tilde{\mathbf{h}}_2 - \mathbf{h}_2\|,$$

$$\left\|\partial_1 k(\mathbf{h}_1, \mathbf{h}_2) - \partial_1 k(\tilde{\mathbf{h}}_1, \mathbf{h}_2)\right\| \leq J\|\tilde{\mathbf{h}}_1 - \mathbf{h}_1\|, \ \left\|\partial_2 k(\mathbf{h}_1, \mathbf{h}_2) - \partial_2 k(\mathbf{h}_1, \tilde{\mathbf{h}}_2)\right\| \leq J\|\tilde{\mathbf{h}}_2 - \mathbf{h}_2\|.$$

Assumption 6 *For any collection of data representations, $\{\mathbf{h}_i\}_{i=1}^N$, the smallest eigenvalue of the corresponding kernel matrix, K, is lower bounded by a positive constant $\gamma > 0$.*

Note that not only the assumptions are relevant to practice, Assumptions 5 and 6 can be also controlled by choosing the class of kernel functions used in the model. For example, the smallest eigenvalue of the kernel matrix, K, can be controlled by the smoothing properties of the kernel [Williams and Rasmussen, 2006].

Consider a particular stochastic step of Algorithm 2 at time t for a given mini-batch, $\mathbf{X}_t$, assuming that the kernel was last updated τ steps ago. The stochastic gradient will take the following form:

$$\hat{\mathbf{g}}_{\mathbf{w}}^t = \frac{1}{N_t}\nabla_{\mathbf{w}}\mathcal{L}(\theta^{t+1}, \mathbf{w}^t, K_{t-\tau}) = \frac{1}{2N_t}\sum_{i,j\in\mathcal{I}_t}\left(K_{t-\tau}^{-1}\mathbf{y}\mathbf{y}^\top K_{t-\tau}^{-1} - K_{t-\tau}^{-1}\right)_{ij}\left(\frac{\partial K_{t-\tau}}{\partial \mathbf{w}}\right)_{ij}.$$

(C.21)

We can define $\hat{\delta}_{\mathbf{w}}^t = \hat{\mathbf{g}}_{\mathbf{w}}^t - \mathbf{g}_{\mathbf{w}}^t$ and uniformly bound $\|\hat{\delta}_{\mathbf{w}}^t - \delta_{\mathbf{w}}^t\|$ in order to enable the same argument as in Theorem 1. To do that, we simply need to understand effect of perturbation of the kernel matrix on $\tilde{\mathbf{g}}_{\mathbf{w}}^t$.

Lemma 3 *Under the given assumptions, the following bound holds for all $i, j = 1, \ldots, N$:*

$$\left|\left(K_{t-\tau}^{-1}\mathbf{y}\mathbf{y}^\top K_{t-\tau}^{-1} - K_{t-\tau}^{-1}\right)_{ij} - \left(K_t^{-1}\mathbf{y}\mathbf{y}^\top K_t^{-1} - K_t^{-1}\right)_{ij}\right| \leq D^2\|y\|^2 + D,$$ (C.22)

where $D = \gamma^{-1} + \left(\gamma - 2GL\tau\lambda\sigma\sqrt{N}\right)^{-1}$.

Proof The difference between $K_{t-\tau}^{-1}$ and K_t^{-1} is simply due to that the former has been computed for $\mathbf{w}^{t-\tau}$ and the latter for $\mathbf{w}^t$. To prove the bound, we need multiple steps.

- First, we need to bound the element-wise difference between $K_{t-\tau}$ and K_t. This is done by using Assumptions 4 and 5 and the triangular inequality:

$$\begin{aligned}
|(K_t)_{ij} - (K_{t-\tau})_{ij}| &\leq G\left(\|\phi_{\mathbf{w}^t}(\bar{\mathbf{x}}_i) - \phi_{\mathbf{w}^{t-\tau}}(\bar{\mathbf{x}}_i)\| + \|\phi_{\mathbf{w}^t}(\bar{\mathbf{x}}_j) - \phi_{\mathbf{w}^{t-\tau}}(\bar{\mathbf{x}}_j)\|\right) \\
&\leq 2GL\|\mathbf{w}^t - \mathbf{w}^{t-\tau}\| \\
&= 2GL\|\sum_{s=1}^{\tau}\lambda_{\mathbf{w}}^{t-\tau+s}\hat{\mathbf{g}}_{\mathbf{w}}^{t-\tau+s}\| \\
&\leq 2GL\tau\lambda\sigma
\end{aligned}$$

26

- Next, since each element of the perturbed matrix is bounded by $2GL\tau\lambda\sigma$, we can bound its spectral norm as follows:

$$\|K_t - K_{t-\tau}\| \leq \|K_t - K_{t-\tau}\|_F \leq 2GL\tau\lambda\sigma\sqrt{N},$$

which means that the minimal singular value of the perturbed matrix is at least $\sigma_1 = \gamma - 2GL\tau\lambda\sigma\sqrt{N}$ due to Assumption 6.

- The spectral norm of the expression of interest can be bounded (quite pessimistically!) by summing up together the largest eigenvalues of the matrix inverses:

$$
\begin{aligned}
&\left|\left(K_{t-\tau}^{-1}\mathbf{y}\mathbf{y}^\top K_{t-\tau}^{-1} - K_{t-\tau}^{-1}\right)_{ij} - \left(K_t^{-1}\mathbf{y}\mathbf{y}^\top K_t^{-1} - K_t^{-1}\right)_{ij}\right| \\
&\leq \left\|\left(K_{t-\tau}^{-1} - K_t^{-1}\right)\mathbf{y}\mathbf{y}^\top\left(K_{t-\tau}^{-1} - K_t^{-1}\right) - \left(K_{t-\tau}^{-1} - K_t^{-1}\right)\right\| \\
&\leq \left(\gamma^{-1} + \left(\gamma - 2GL\tau\lambda\sigma\sqrt{N}\right)^{-1}\right)^2 \|\mathbf{y}\|^2 + \gamma^{-1} + \left(\gamma - 2GL\tau\lambda\sigma\sqrt{N}\right)^{-1}.
\end{aligned}
$$

Each element of a matrix is bounded by the largest eigenvalue. ∎

Using Lemma 3, it is straightforward to extend Theorem 1 to Algorithm 2.

Theorem 2 *Let $\{(\theta^t, \mathbf{w}^t)\}$ be a sequence generated from Algorithm 2 with learning rates for θ and $\mathbf{w}$ being $\lambda_\theta^t = c_\theta^t \alpha_t$ and $\lambda_\mathbf{w}^t = c_\mathbf{w}^t \alpha_t$, respectively, where $c_\theta^t, c_\mathbf{w}^t, \alpha_t$ are positive scalars, such that*

$$0 < \inf_t c_{\{\theta, \mathbf{w}\}}^t \leq \sup_t c_{\{\theta, \mathbf{w}\}}^t < \infty, \quad \alpha_t \geq \alpha_{t+1}, \quad \sum_{t=1}^\infty \alpha_t = +\infty, \quad \sum_{t=1}^\infty \alpha_t^2 < +\infty, \quad \forall t. \tag{C.23}$$

Then, under Assumptions 1 through 6 and if $\sigma = \sup_t \sigma_t < \infty$, we have

$$\lim_{t\to\infty} \mathbb{E}\|\nabla\mathcal{L}(\theta^t, \mathbf{w}^t)\| = 0. \tag{C.24}$$

Proof The proof is identical to the proof of Theorem 1. The only difference is in the following upper bound on the expected gradient error: $\mathbb{E}\|\delta_\mathbf{w}^t\| \leq \sigma + 2ND(D\|y\|^2 + 1)J\rho$, where D is as given in Lemma 3. ∎

Remark 1 *Even though the provided bounds are crude due to pessimistic estimates of the perturbed kernel matrix spectrum[13], we still see a fine balance between the delay, τ, and the learning rate, λ, as given in the expression for the D constant.*

D   Details on the datasets

The datasets varied in the number of time steps (from hundreds to a million), input and output dimensionality, and the nature of the estimation tasks.

---

[13]Tighter bounds can be derived by inspecting the effects of perturbations on specific kernels as well as using more specific assumptions about the data distribution.

D.1 Self-driving car

The following is description of the input and target time series used in each of the autonomous driving tasks (dimensionality is given in parenthesis).

- **Car speed estimation:**

    - *Features:* GPS velocity (3), fiber gyroscope (3).
    - *Targets:* speed measurements from the car speedometer.

- **Car yaw estimation:**

    - *Features:* acceleration (3), compass measurements (3).
    - *Targets:* yaw in the car-centric frame.

- **Lane sequence prediction:** Each lane was represented by 8 cubic polynomial coefficients [4 coefficients for x (front) and 4 coefficients for y (left) axes in the car-centric frame].

    - *Features:* lanes at a previous time point (16), GPS velocity (3), fiber gyroscope (3), compass (3), steering angle (1).
    - *Targets:* coefficients of a cubic polynomial fit.

- **Position estimation of a nearest vehicle:** x and y coordinates in the car-centric frame.

    - *Features:* x and y at a previous time point (2), GPS velocity (3), fiber gyroscope (3), compass (3), steering angle (1).
    - *Targets:* x and y coordinates.

E   Neural architectures

Details on the best neural architectures used for each of the datasets are given in Table 3.

Table 3: Summary of the feedforward and recurrent neural architectures and the corresponding hyperparameters used in the experiments. GP-based models used the same architectures as their non-GP counterparts.

Name	Data	Time lag	# layers	# units*	type	Regularizer	Optimizer
NARX	Actuator	10	1	16	ReLU	dropout(0.5)	Adagrad(0.1)
	Drives	10	1	16			
	GEF-power	48	1	256			
	GEF-wind	48	1	16			
	Car	100	1	128			
RNN	Actuator	10	2	8	tanh	dropout(0.5)	Adam(0.01)
	Drives	10	1	16			
	GEF-power	48	1	16			
	GEF-wind	48	1	32			
	Car	100	1	128			
LSTM	Actuator	10	2	32	tanh	dropout(0.5)	Adam(0.01)
	Drives	10	1	16			
	GEF-power	48	2	256			
	GEF-wind	48	1	64			
	Car	100	2	32			

*Each layer consisted of the same number of units given in the table.