
Probabilistic Programming with Gaussian Process Memoization

Anonymous Author(s)

Affiliation

Address

email

Abstract

This paper describes the *Gaussian process memoizer*, a probabilistic programming technique that uses Gaussian processes to provide a statistical alternative to memorization. Memoizing a target procedure results in a self-caching wrapper that remembers previously computed values. Gaussian process memoization additionally produces a statistical emulator based on a Gaussian process whose predictions automatically improve whenever a new value of the target procedure becomes available. This paper also introduces an efficient implementation, named `gpmem`, that can use kernels given by a broad class of probabilistic programs. The flexibility of `gpmem` is illustrated via three applications: (i) GP regression with hierarchical hyper-parameter learning, (ii) Bayesian structure learning via compositional kernels generated by a probabilistic grammar, and (iii) a bandit formulation of Bayesian optimization with automatic inference and action selection. All applications share a single 50-line Python library and require fewer than 20 lines of probabilistic code each.

1 Introduction

Probabilistic programming could be revolutionary for machine intelligence due to universal inference engines and the rapid prototyping for novel models (Ghahramani, 2015). This levitates the design and testing of new models as well as the incorporation of complex prior knowledge which currently is a difficult and time consuming task. Probabilistic programming languages aim to provide a formal language to specify probabilistic models in the style of computer programming and can represent any computable probability distribution as a program. In this work, we will introduce new features of Venture, a recently developed probabilistic programming language. We consider Venture the most compelling of the probabilistic programming languages because it is the first probabilistic programming language suitable for general purpose use (Mansinghka et al., 2014). Venture comes with scalable performance on hard problems and with a general purpose inference engine. The inference engine deploys Markov Chain Monte Carlo (MCMC) methods (for an introduction, see Andrieu et al. (2003)). MCMC lends itself to models with complex structures such as probabilistic programs or hierarchical Bayesian non-parametric models since they can provide a vehicle to express otherwise intractable integrals necessary for a fully Bayesian representation. MCMC is scalable, often distributable and also compositional. That is, one can arbitrarily chain MCMC kernels to infer over several hierarchically connected or nested models as they will emerge in probabilistic programming.

One very powerful model yet unseen in probabilistic programming languages are Gaussian Processes (GPs). GPs are gaining increasing attention for representing unknown functions by posterior probability distributions in various fields such as machine learning, signal processing, computer vision and bio-medical data analysis. Making GPs available in probabilistic programming is crucial to allow a language to solve a wide range of problems. Hard problems include but are not limited

054 to hierarchical prior construction (Neal, 1997), Bayesian Optimization Snoek et al. (2012) and sys-
055 tems for inductive learning of symbolic expressions such as the one introduced in the Automated
056 Statistician project Duvenaud et al. (2013); Lloyd et al. (2014). Learning such symbolic expressions
057 is a hard problem that requires careful design of approximation techniques since standard inference
058 method do not apply.

059 In the following, we will present `gpmem` as a novel probabilistic programming technique that solves
060 such hard problems. `gpmem` introduces a statistical alternative to standard memoization. Our con-
061 tribution is threefold:

- 063 • we introduce an efficient implementation of `gpmem` in form of a self-caching wrapper that
064 remembers previously computed values;
- 065 • we illustrate the statistical emulator that `gpmem` produces and how it improves with every
066 data-point that becomes available; and
- 067 • we show how one can solve hard problems of state-of-the-art machine learning related to
068 GP using `gpmem` in a Bayesian fashion and with only a few lines of Venture code.

070 We evaluate the contribution on problems posed by the GP community using real world and syn-
071 thetic data by assessing quality in terms of posterior distributions of symbolic outcome and in terms
072 of the residuals produced by our probabilistic programs. The paper is structured as follows, we will
073 first provide some background on memoization. We will explain programming in Venture and pro-
074 vide a brief introduction to GPs. We introduce `gpmem` and its use in probabilistic programming and
075 Bayesian modeling. Finally, we will show how we can apply `gpmem` on problems of causally struc-
076 tured hierarchical priors for hyper-parameter inference, structure discovery for Gaussian Processes
077 and Bayesian Optimization including experiments with real world and synthetic data.

078 2 Background

079 2.1 Memoization

080 Memoization is the practice of storing previously computed values of a function so that future calls
081 with the same inputs can be evaluated by lookup rather than recomputation. Research on the Church
082 language (Goodman et al., 2008) pointed out that although memoization does not change the se-
083 mantics of a deterministic program, it does change that of a stochastic program. In fact, there is an
084 infinite range of possible caching policies (specifications of when to use a stored value and when
085 to recompute), each potentially having a different semantics. Any particular caching policy can
086 be understood by random world semantics (Poole, 1993; Sato, 1995) over the stochastic program:
087 each possible world corresponds to a mapping from function input sequence to function output se-
088 quence (McAllester et al., 2008). In Venture, these possible worlds are first-class objects, known as
089 *traces* (Mansinghka et al., 2014).

090 2.2 Venture

091 Venture is a compositional language for custom inference strategies that comes with a Scheme-
092 like and a JavaScript-like front-end syntaxes. Its implementation is based on on three concepts:
093 (i) *stochastic procedures* that specify and encapsulate random variables, analogously to conditional
094 probability tables in a Bayesian network; (ii) *execution traces* that represent (partial) execution his-
095 tories and track the conditional dependencies of the random variables occurring therein; and (iii)
096 *scaffolds* that partition execution histories and factor global inference problems into sub-problems.
097 These building blocks provide a powerful and concise way to represent probability distributions,
098 including distributions with a dynamically determined and unbounded set of random variables. In
099 this paper we will use only the four basic Venture directives: ASSUME, OBSERVE, SAMPLE and
100 INFER.

- 104 • ASSUME induces a hypothesis space for (probabilistic) models including random variables
105 by binding the result of a supplied expression to a supplied symbol.
- 106 • Whereas in Scheme an expression is evaluated within an environment, in Venture an ex-
107 pression is evaluated within a (partial) trace of the model program. Thus, the value of an

108 expression within a model program is a random variable, whose randomness comes from
 109 the distribution on possible execution traces of the program. The SAMPLE directive sam-
 110 plesthe value of the supplied expression within the current model program.

- 111 • OBSERVE constrains the supplied expression to have the supplied value. In other words,
 112 all samples taken after an OBSERVE are conditioned on the observed data.
- 113 • INFER uses the supplied inference program to mutate the execution trace. For a correct in-
 114 ference program, this will result approximate sampling from the true posterior on execution
 115 traces, conditioned on the model and constraints introduced by ASSUME and OBSERVE.
 116 The posterior on any random variable can then be approximately sampled by calling SAM-
 117 PLE to extract values from the trace.

119 INFER is commonly done using the Metropolis–Hastings algorithm (MH) (Metropolis et al., 1953).
 120 Many of the most popular MCMC algorithms can be interpreted as special cases of MH (Andrieu
 121 et al., 2003). We can outline the MH algorithm as follows. The following two-step process is
 122 repeated as long as desired (say, for T steps): First we sample x^* from a proposal distribution q :

$$123 \quad x^* \sim q(x^* | x^t); \quad (1)$$

124 then we accept this proposal ($x^{t+1} \leftarrow x^*$) with probability

$$125 \quad \alpha = \min \left\{ 1, \frac{p(x^*)q(x^t | x^*)}{p(x^t)q(x^* | x^t)} \right\}; \quad (2)$$

126 if the proposal is not accepted then we take $x^{t+1} \leftarrow x^t$.

127 Venture includes a built-in generic MH inference program which performs the above steps on any
 128 specified set of random variables in the model program. In that inference program, probabilistic
 129 execution traces play the role of x above.

132 2.3 Gaussian Processes

134 We now introduce GP related theory and notations. We work exclusively with two-variable regres-
 135 sion problems. Let the data be pairs of real-valued scalars $\{(x_i, y_i)\}_{i=1}^n$ (complete data will be de-
 136 noted by column vectors \mathbf{x}, \mathbf{y}). In regression, one tries to learn a functional relationship $y_i = f(x_i)$,
 137 where the function f is to be learned. GPs present a non-parametric way to express prior knowledge
 138 on the space of possible functions f . Formally, a GP is an infinite-dimensional extension of the
 139 multivariate Gaussian distribution. For any finite set of inputs \mathbf{x} , the marginal prior on $f(\mathbf{x})$ is the
 140 multivariate Gaussian

$$141 \quad f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x})),$$

142 where $m(\mathbf{x}) = \mathbb{E}_f[f(\mathbf{x})]$ is the mean function and $k(\mathbf{x}, \mathbf{x}') = \text{Cov}_f(f(\mathbf{x}), f(\mathbf{x}'))$ is the covariance
 143 function, a.k.a. kernel.¹ In all examples below, our prior mean function m is identically zero; this is
 144 the most common choice. The marginal likelihood can be expressed as:

$$145 \quad p(f(\mathbf{x}) = \mathbf{y} | \mathbf{x}) = \int p(f(\mathbf{x}) = \mathbf{y} | f, \mathbf{x}) p(f|\mathbf{x}) df \quad (3)$$

147 where here $p(f|\mathbf{x}) = p(f) \sim \mathcal{GP}(m, k)$ since we assume no dependence of f on \mathbf{x} . We can sample
 148 a vector of unseen data $\mathbf{y}^* = f(\mathbf{x}^*)$ from the predictive posterior with

$$149 \quad \mathbf{y}^* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad (4)$$

151 a multivariate normal with mean vector

$$152 \quad \boldsymbol{\mu} = k(\mathbf{x}^*, \mathbf{x}) k(\mathbf{x}, \mathbf{x})^{-1} \mathbf{y} \quad (5)$$

153 and covariance matrix

$$154 \quad \boldsymbol{\Sigma} = k(\mathbf{x}^*, \mathbf{x}^*) - k(\mathbf{x}^*, \mathbf{x}) k(\mathbf{x}, \mathbf{x})^{-1} k(\mathbf{x}, \mathbf{x}^*). \quad (6)$$

156 Often one assumes the values \mathbf{y} are noisily measured, that is, one only sees the values of $\mathbf{y}_{\text{noisy}} =$
 157 $\mathbf{y} + \mathbf{w}$ where \mathbf{w} is Gaussian white noise with variance σ_{noise}^2 . In that case, the log-likelihood is

$$158 \quad \log p(\mathbf{y}_{\text{noisy}} | \mathbf{x}) = -\frac{1}{2} \mathbf{y}^\top (\boldsymbol{\Sigma} + \sigma_{\text{noise}}^2 \mathbf{I})^{-1} \mathbf{y} - \frac{1}{2} \log |\boldsymbol{\Sigma} + \sigma_{\text{noise}}^2 \mathbf{I}| - \frac{n}{2} \log 2\pi \quad (7)$$

160 ¹ Note that $m(\mathbf{x}) = (m(x_i))_{i=1}^n$ and $k(\mathbf{x}, \mathbf{x}') = (k(x_i, x'_{i'}))_{1 \leq i \leq n, 1 \leq i' \leq n'}$, where n' is the number of entries in
 161 \mathbf{x}' .

162 where n is the number of data points. Both log-likelihood and predictive posterior can be computed
 163 efficiently in a Venture SP with an algorithm that resorts to Cholesky factorization(Rasmussen and
 164 Williams, 2006, chap. 2) resulting in a computational complexity of $\mathcal{O}(n^3)$ in the number of data
 165 points.

166 The covariance function governs high-level properties of the observed data such as linearity, periodicity
 167 and smoothness. The most widely used form of covariance function is the squared exponential:
 168

$$k(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right), \quad (8)$$

171 where σ and ℓ are hyperparameters: σ is a scaling factor and ℓ is the typical length-scale.
 172

173 Adjusting hyperparameters results in a new covariance function with the same qualitative human-
 174 interpretation; more drastically different covariance functions are achieved by changing the structure
 175 of the covariance function. Note that covariance function structures are compositional: adding or
 176 multiplying two valid covariance functions results in another valid covariance function. We suggest
 177 (**TODO** cite or point to later in paper) that adding covariance structures k_1, k_2 together,
 178

$$k_3(x, x') = k_1(x, x') + k_2(x, x'), \quad (9)$$

179 corresponds to combining global structures, while multiplying covariance functions,
 180

$$k_4(x, x') = k_1(x, x') \times k_2(x, x'), \quad (10)$$

181 corresponds to combining local structures. Note that both k_3 and k_4 are valid covariance function
 182 structures.
 183

184 3 Venture GPs

185 The Venture procedure `make-gp` takes as input a mean function and a covariance function, and
 186 outputs a procedure for sampling from a Gaussian process. In effect, each call to this procedure
 187 samples from (4) conditioned on the return values of all previous samples. `make-gp` allows us to
 188 perform GP inference in Venture with only a few lines of code. We can concisely express a wide
 189 variety of GPs: simple smoothing with fixed hyper-parameters, or a prior on hyper-parameters, or
 190 a custom covariance function. Inference on hyper-parameters can be performed using Venture's
 191 built-in MH operator or a custom inference strategy.
 192

193 Venture code to create and sample from a GP with a
 194 smoothing kernel and hyperparameters is shown in Listing 3.

```
// HYPER-PARAMETERS
assume log_sf = tag('hyper, log(gamma(1,1)))
assume log_l = tag('hyper, log(gamma(1,1)))

// COVARIANCE FUNCTION
assume se = make_squaredexp(log_sf, log_l)

// MAKE GAUSSIAN PROCESS
assume gp = make_gp( 0, se)

// INCORPORATE OBSERVATIONS
observe gp(array x[1],...x[n])= array(y[1],...,y[n])

// INFER HYPER-PARAMETERS
infer mh('hyper, one, 1))
```

209 The first two lines declare the hyper-parameters. We tag both of them to belong to the “scope”
 210 ‘hyper. These tags are supplied to the inference program (in this case, MH) to specify on which
 211 random variables inference should be done. In this paper, we use MH inference throughout. Scopes
 212 may be further subdivided into blocks, on which block proposals can be made. In this paper we do
 213 not use block proposals; MH inference is done on one variable at a time.
 214

215 The `ASSUME` directives describe the GP model: `sf` and `l` (corresponding to σ and ℓ) are drawn
 216 from independent $\Gamma(1, 3)$ distributions. The squared exponential covariance function can be defined

outside the Venture code in a conventional programming language (e.g. Python) and imported as a foreign SP. In that way, the user can define custom covariance functions using his or her language and libraries of choice, without having to port existing code into Venture’s modelling language. In the above, the factory function `f`, which produces a squared exponential function with the supplied hyperparameters, is imported from Python (we have omitted the Python code). In the next line `f` is used to produce a covariance function `SE`, whose (random) hyperparameters are `l` and `s f`. Finally, we declare `GP` to be a Gaussian process with mean zero and covariance function `SE`.

I don’t know what these two paragraphs mean –Ben

In the case where hyper-parameters are unknown they can be found deterministically by optimizing the marginal likelihood using a gradient based optimizer. Non-deterministic, Bayesian representations of this case are also known (Neal, 1997).

We have already implemented this in listing 3. We draw the hyper-parameters from a Γ -prior for a Bayesian treatment of hyper-parameters. This is simple using the build in stochastic procedure that simulates drawing samples from a gamma distribution. The program gives rise to a Bayesian representation of GPs, which we will explore in the following.

3.1 Gaussian process memoization: `gpmem`

TODO write

3.2 A Bayesian interpretation

3.2.1 Data modelling as a special case of `gpmem`

From the standpoint of computation, a data set of the form $\{(x_i, y_i)\}$ can be thought of as a function $y = f_{\text{restr}}(x)$, where f_{restr} is restricted to only allow evaluation at a specific set of inputs x . Modelling the data set with a GP then amounts to trying to learn a smooth function f_{emu} (“emu” stands for “emulator”) which extends f to its full domain. Indeed, if f_{restr} is a foreign procedure made available as a black-box to Venture, whose secret underlying source code is:

```

245 def f_restr(x):
246     if x in D:
247         return D[x]
248     else:
249         raise Exception('Illegal input')
  
```

Then the `OBSERVE` code in Listing 3 can be rewritten using `gpmem` as follows (where here the data set `D` has keys $x[1], \dots, x[n]$):

```

252 [ASSUME (list f_compute f_emu) (gpmem f_restr)]
253 for i=1 to n:
254   [PREDICT (f_compute x[i])]
255   [INFER (MH {hyper-parameters} one 100)]
256   [SAMPLE (f_emu (array 1 2 3))]
  
```

This rewriting has at least two benefits: (i) readability (in some cases), and (ii) amenability to active learning. As to (i), the statistical code of creating a Gaussian process is replaced with a memoization-like idiom, which will be more familiar to programmers. As to (ii), when using `gpmem`, it is quite easy to decide incrementally which data point to sample next: for example, the loop from $x[1]$ to $x[n]$ could be replaced by a loop in which the next index i is chosen by a supplied decision rule. In this way, we could use `gpmem` to perform online learning using only a subset of the available data.

3.2.2 The efficacy of learning hyperparameters

The probability of the hyper-parameters of a GP with assumptions as above and given covariance function structure \mathbf{K} can be described as:

$$P(\boldsymbol{\theta} | \mathbf{D}, \mathbf{K}) = \frac{P(\mathbf{D} | \boldsymbol{\theta}, \mathbf{K})P(\boldsymbol{\theta} | \mathbf{K})}{P(\mathbf{D} | \mathbf{K})}. \quad (11)$$

Let the \mathbf{K} be the sum of a smoothing and a white noise (WN) kernel. For this case, Neal suggested the problem of outliers in data as a use-case for a hierarchical Bayesian treatment of Gaussian processes (1997)². The work suggests a hierarchical system of hyper-parameterization (Fig. 1a). Here, we draw hyper-parameters from a Γ distributions:

$$\ell^{(t)} \sim \Gamma(\alpha_1, \beta_1), \sigma^{(t)} \sim \Gamma(\alpha_2, \beta_2) \quad (12)$$

and in turn sample the α and β from Γ distributions as well:

$$\alpha_1^{(t)} \sim \Gamma(\alpha_\alpha^1, \beta_\alpha^1), \alpha_2^{(t)} \sim \Gamma(\alpha_\alpha^2, \beta_\alpha^2), \dots \quad (13)$$

Assuming the covariance structure is an additive comprised of a smoothing and a white noise kernel, one can represent this kind of model using `gpmem` with only a few lines of code:

```

281 //////////////////////////////////////////////////////////////////
282 // SETTING UP THE MODEL
283 assume alpha_sf = tag('hyperhyper, gamma(7, 1))
284 assume beta_sf = tag('hyperhyper, gamma(7, 1))
285 assume alpha_l = tag('hyperhyper, gamma(7, 1))
286 assume beta_l = tag('hyperhyper, gamma(7, 1))

287 // Parameters of the covariance function
288 assume log_sf = tag('hyper, log(gamma(alpha_sf, beta_sf)))
289 assume log_l = tag('hyper, log(gamma(alpha_l, beta_l)))
290 assume log_sigma = tag('hyper, log(uniform_continuous(0, 2)))

291 // The covariance function
292 assume se = make_squaredexp(log_sf, log_l)
293 assume wn = make_whitenoise(log_sigma)
294 assume composite_covariance = add_funcs(se, wn)

295 //////////////////////////////////////////////////////////////////
296 // PERFORMING INFERENCE
297 // Create a prober and emulator using gpmem
298 assume f_restr = get_neal_blackbox()
299 assume (f_compute, f_emu) = gpmem(f_restr, composite_covariance)

300 // Probe all data points
301 predict mapv(f_compute, get_neal_data_xs())

303 // Infer hypers and hyperhypers
304 infer repeat(100, do(
305     mh('hyperhyper, one, 2),
306     mh('hyper, one, 1)))
307

```

Neal provides a custom inference algorithm setting and evaluates it using the following synthetic data problem. Let f be the underlying function that generates the data:

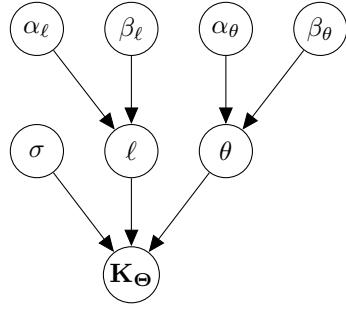
$$f(x) = 0.3 + 0.4x + 0.5 \sin(2.7x) + \frac{1.1}{(1+x^2)} + \eta \quad \text{with } \eta \sim \mathcal{N}(0, \sigma) \quad (14)$$

We synthetically generate outliers by setting $\sigma = 0.1$ in 95% of the cases and to $\sigma = 1$ in the remaining cases. `gpmem` can capture the true underlying function within only 100 MH steps on the hyper-parameters to get a good approximation for their posterior (see Fig. 1). Note that Neal devices an additional noise model and performs large number of Hybrid-Monte Carlo and Gibbs steps. We illustrate the hyper-parameter by showing the shift of the distribution on the noise parameter σ (Fig. 2). We see that `gpmem` learns the posterior distribution well, the posterior even exhibits a bimodal histogram when sampling σ 100 times reflecting the two modes of data generation, that is normal noise and outliers³.

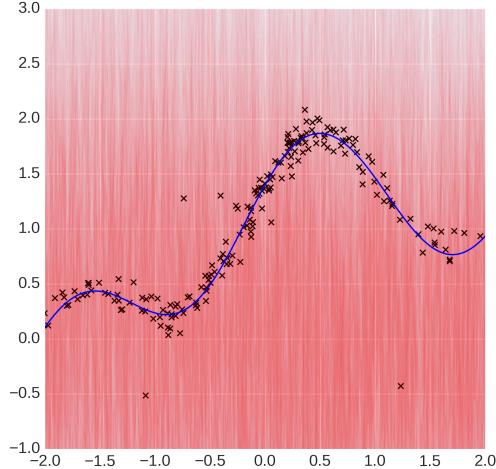
²In (Neal, 1997) the sum of an SE plus a constant kernel is used. We stick to the WN kernel for illustrative purposes.

³For this pedagogical example we have increased the probability for outliers in the data generation slightly from 0.05 to 0.2

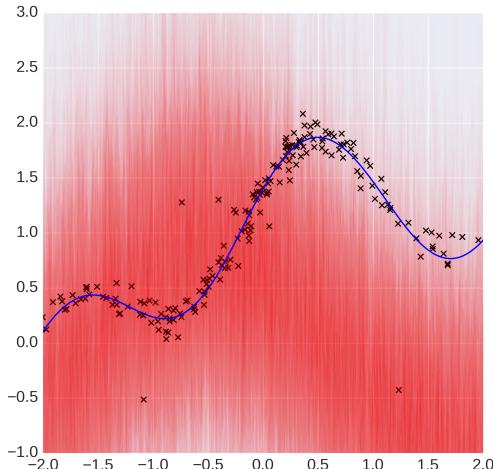
324
 325
 326
 327
 328
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 340
 341
 342
 343
 344
 345
 346



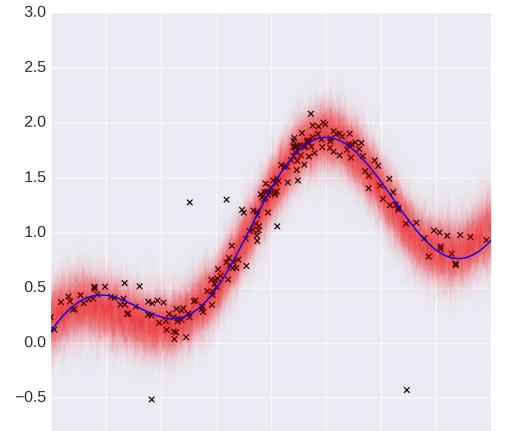
(a) Hierarchical Prior



(b) Prior Inference



(c) Observed



(d) Inferred

368
 369
 370
 371
 372
 373
 374
 375
 376
 377

Figure 1: (a) depicts the hierarchical structure of the hyper-parameter as constructed in the work by Neal as a Bayesian Network. (b)-(d) shows gpmem on Neal's example. We see that prior renders functions all over the place (a). After gpmem observes a some data-points an arbitrary smooth trend with a high level of noise is sampled. After running inference on the hierarchical system of hyper-parameters we see that the posterior reflects the actual curve well. Outliers are treated as such and do not confound the GP.

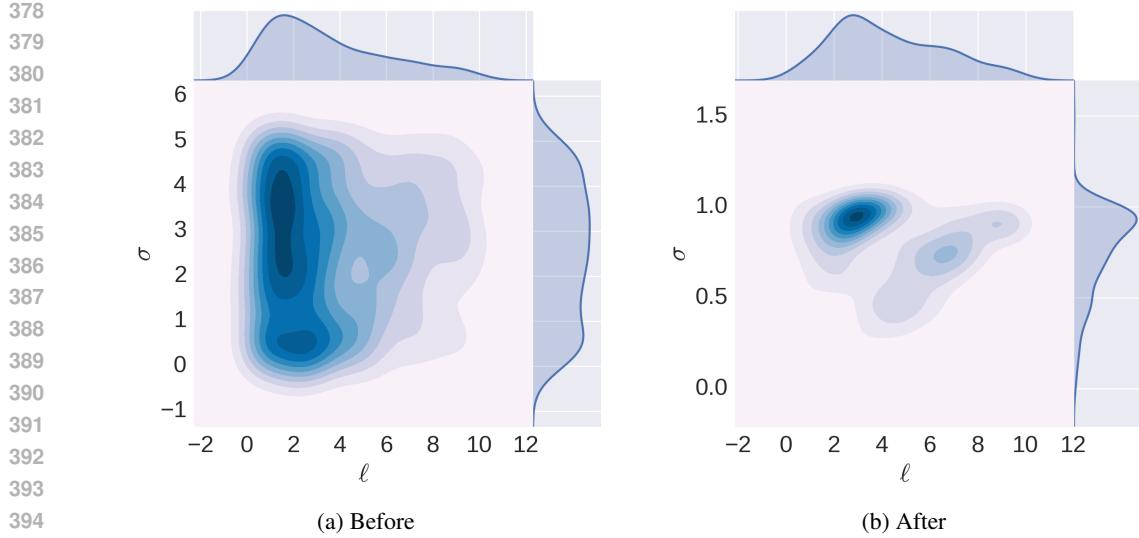


Figure 2: Hyper-parameter inference on the parameter of the noise kernel. We show 100 samples drawn from the distribution on σ . One can clearly recognise the shift from the uniform prior $\mathcal{U}(0, 5)$ to a double peak distribution around the two modes - normal and outlier.

3.2.3 Broader applicability of gpmem

More generally, gpmem is relevant not just when a data set is available, but also whenever we have at hand a function f_{restr} which is expensive or impractical to evaluate many times. gpmem allows us to model f_{restr} with a GP-based emulator f_{emu} , and also to use f_{emu} during the learning process to choose, in an online manner, an effective set of probe points $\{x_i\}$ on which to use our few evaluations of f_{restr} . This idea is illustrated in detail in Section 4. Before doing this, we will illustrate another benefit of having a probabilistic programming apparatus for GP modelling: the linguistically unified treatment of inference over structure and inference over parameters. This unification makes interleaved joint inference over structure and parameters very natural, and allows us to give a short, elegant description of what it means to ‘‘learn the covariance function,’’ both in prose and in code. Furthermore, the example in Section 3.3 below recovers the performance of current state-of-the-art GP-based models.

3.3 Structure Learning

The space of possible kernel composition is infinite. Combining inference over this space with the problem of finding a good parameterization that could potentially explain the observed data best poses a hard problem. The natural language interpretation of the meaning of a kernel and its composition renders this a problem of symbolic computation. Duvenaud and colleagues note that a sum of kernels can be interpreted as logical OR operations and kernel multiplication as logical AND (2013). This is due to the kernel rendering two points similar if k_1 OR k_2 outputs a high value in the case of a sum. Respectively, multiplication of two kernels results in high values only if k_1 AND k_2 have high values (see Fig. 3 exemplifies how to interpret global vs. local aspects and its symbolic analog respectively). In the following, we will refer to covariance functions that are not composite as base covariance functions.

Knowledge about the composite nature of covariance functions is not new, however, until recently, the choice and the composition of covariance functions were done ad-hoc. The Automated Statistician Project came up with an approximate search over the possible space of kernel structures (Duvenaud et al., 2013; Lloyd et al., 2014). However, a fully Bayesian treatment of this was not done before. The case where the covariance structure is not given is even more interesting. Our probabilistic programming based MCMC framework approximates the following intractable integrals of

432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450

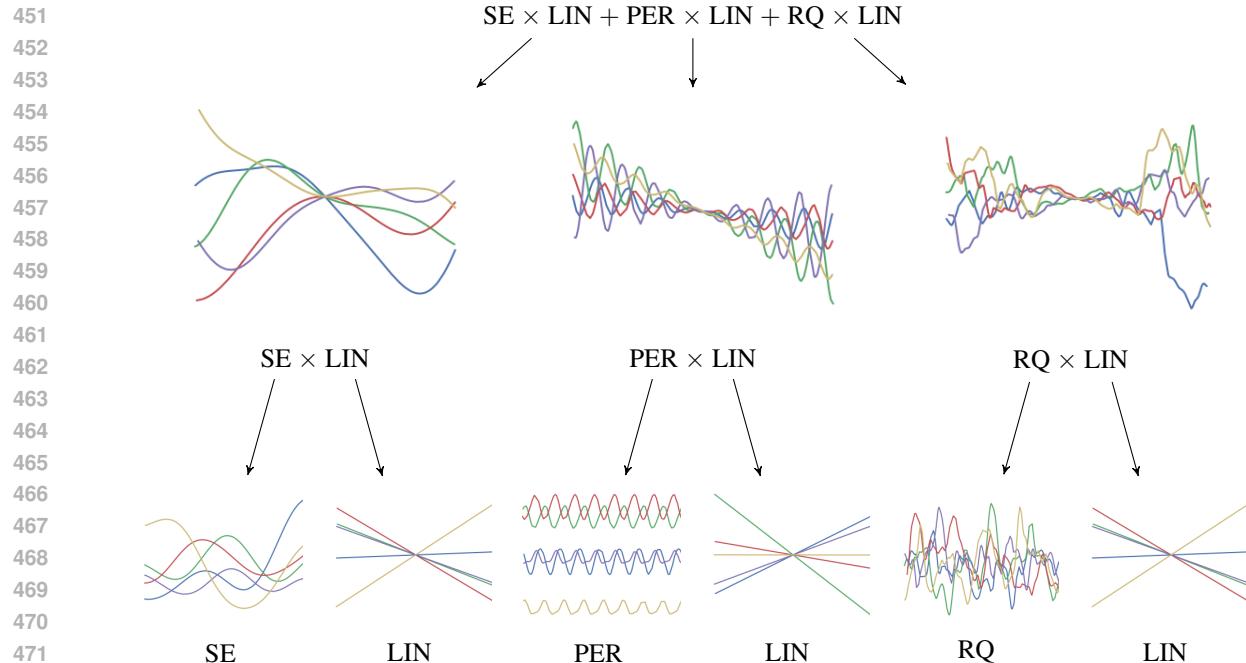


Figure 3: Composition of covariance function parsed using the example $\text{SE} \times \text{LIN} + \text{PER} \times \text{LIN} + \text{RQ} \times \text{LIN}$

476
477
478
479
480
481
482
483
484
485

486 the expectation for the prediction:
 487
 488
 489
 490

$$491 \quad \mathbb{E}[y^* | x^*, \mathbf{D}, \mathbf{K}] = \iint f(x^*, \boldsymbol{\theta}, \mathbf{K}) P(\boldsymbol{\theta} | \mathbf{D}, \mathbf{K}) P(\mathbf{K} | \boldsymbol{\Omega}, s, n) d\boldsymbol{\theta} d\mathbf{K}. \quad (15)$$

492
 493
 494
 495
 496

This is done by sampling from the posterior probability distribution of the hyper-parameters and the possible kernel:

$$499 \quad y^* \approx \frac{1}{T} \sum_{t=1}^T f(x^* | \boldsymbol{\theta}^{(t)}, \mathbf{K}^{(t)}). \quad (16)$$

500
 501

In order to provide the sampling of the kernel, we introduce a stochastic process to the SP that simulates the grammar for algebraic expressions of covariance function algebra:

$$507 \quad \mathbf{K}^{(t)} \sim P(\mathbf{K} | \boldsymbol{\Omega}, s, n) \quad (17)$$

508
 509
 510
 511

512 Here, we start with a set of possible kernels and draw a random subset. For this subset of size n , we
 513 sample a set of possible operators that operate on the base kernels.

514 The marginal probability of a kernel structure which allows us to sample is characterized by the
 515 probability of a uniformly chosen subset of the set of n possible covariance functions times the
 516 probability of sampling a global or a local structure which is given by a binomial distribution:
 517

$$518 \quad P(\mathbf{K} | \boldsymbol{\Omega}, s, n) = P(\boldsymbol{\Omega} | s, n) \times P(s | n) \times P(n), \quad (18)$$

519 with

$$521 \quad P(\boldsymbol{\Omega} | s, n) = \binom{n}{r} p_{+ \times}^k (1 - p_{+ \times})^{n-k} \quad (19)$$

522 and

$$523 \quad P(s | n) = \frac{n!}{|s|!} \quad (20)$$

524 where $P(n)$ is a prior on the number of base kernels used which can sample from a discrete uniform
 525 distribution. This will strongly prefer simple covariance structures with few base kernels since
 526 individual base kernels are more likely to be sampled in this case due to (20). Alternatively, we
 527 can approximate a uniform prior over structures by weighting $P(n)$ towards higher numbers. It is
 528 possible to also assign a prior for the probability to sample global or local structures, however, we
 529 have assigned complete uncertainty to this with the probability of a flip $p = 0.5$.

530

531 Many equivalent covariance structures can be sampled due to covariance function algebra and equivalent representations with different parameterization (Lloyd et al., 2014). Certain covariance functions can differ in terms of the hyper-parameterization but can be absorbed into a single covariance function with a different parameterization. To inspect the posterior of these equivalent structures we convert each kernel expression into a sum of products and subsequently simplify. Rules for this simplification can be found in appendix B.

532 For reproducing results from the Automated Statistician Project in a Bayesian fashion we first define
 533 a prior on the hypothesis space. Note that, as in the implementation of the Automated Statistician,
 534 we upper-bound the complexity of the space of covariance functions we want to explore. We also
 535 put vague priors on hyper-parameters.
 536

10

```

540 // GRAMMAR FOR KERNEL STRUCTURE
541 assume base_kernels = list(se, wn, lin, per, rq) // defined as above
542
543 // prior on the number of kernels
544 assume p_number_k = tag('number_kernels, uniform_structure(n))
545 assume s = tag('choice_subset, subset(base_kernels, p_number_k))
546
547 assume cov_compo = proc(l) {
548     // kernel composition
549     if (size(l) <= 1)
550     then { first(l)
551         else { if (flip()) then { add_funcs(first(l), cov_compo(rest(l))) }
552                 else { mult_funcs(first(l), cov_compo(rest(l))) }
553             }
554         }
555
556 assume K = tag('composit, cov_compo(s))
557
558 // PERFORMING INFERENCE
559 infer repeat(2000, do(
560
561 mh('number_kernel, one, 1),
562 mh('choice_subset, one, 1),
563 mh('composit, one, 1),
564 mh('hyper, one, 10)))
565

```

564 We defined the space of covariance structures in a way allowing us to reproduce results for covariance
565 function structure learning as in the Automated Statistician. This lead to coherent results, for
566 example for the airline data set describing monthly totals of international airline passengers (Box
567 et al., 1997, according to Duvenaud et al., 2013. We will elaborate the result using a sample from the
568 posterior (Fig. ??). The sample is identical with the highest scoring result reported in previous work
569 using a search-and-score method (Duvenaud et al., 2013) for the CO₂ data set (see Rasmussen and
570 Williams, 2006 for a description) and the predictive capability is comparable. However, the components
571 factor in a different way due to different parameterization of the individual base kernels.

572 We further investigated the quality of our stochastic processes by running a leave one out cross-
573 validation to gain confidence on the posterior. This resulted in 545 independent runs of the Markov
574 chain that produced a coherent posterior: our Bayesian interpretation of GP structure and GPs pro-
575 duced a posterior of structures that is in line with previous results on this data set (Duvenaud et al.,
576 2013; see Fig. ??).

577 We ran similar evaluation on the airline data set resulting in a similar structure to what was previously
578 reporte (Fig. ??, residuals and log-score along the Markov chain see Fig. ??).

579 We found the final sample of multiple runs to be most informative. This kind of Markov Chain
580 seems to produce samples that are highly auto-correlated.
581

582 Given two additive components $\mathbf{K} = \mathbf{K}_a + \mathbf{K}_b$, one can compute the marginal of a global com-
583 ponent of a composite kernel structure (Benavoli and Mangili, 2015) with a gaussian posterior
584 $\mathcal{N}(f_a | \hat{\mu}_a, \hat{\mathbf{K}}_a)$ where:

585

$$\hat{\mu}_a = \mathbf{K}_a(\mathbf{x}, \mathbf{x}^*) \mathbf{K}(\mathbf{x}^*, \mathbf{x}^*)^{-1} \mathbf{y} \quad (21)$$

586 and covariance matrix

587

$$\hat{\mathbf{K}}_a = \mathbf{K}_a(\mathbf{x}, \mathbf{x}) - \mathbf{K}_a(\mathbf{x}, \mathbf{x}^*) \mathbf{K}(\mathbf{x}^*, \mathbf{x}^*)^{-1} \mathbf{K}_a(\mathbf{x}^*, \mathbf{x}). \quad (22)$$

588

589

590

591

592

593

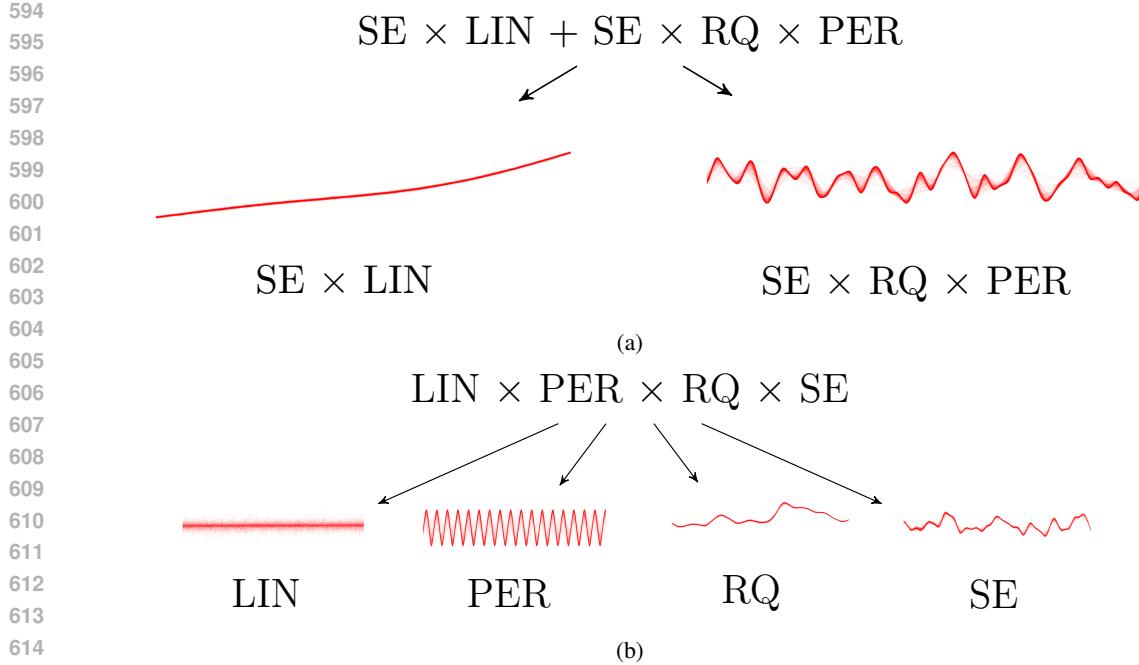


Figure 4: (a) most frequent sample drawn from the posterior on structure. We have found two global components. First, A smooth trend (LINxSE) with a non-linear increasing slope. Second, a periodic component with increasing variation and noise. (b) second most frequent sample drawn from the posterior on structure. We found one global component. It is comprised of local changes that are periodic and with changing variation.

4 Bayesian Optimization

Bayesian optimization casts the problem of finding the global maximum of an unknown function as a hierarchical decision problem (Ghahramani, 2015). Evaluating the actual function may be very expensive, either in computation time or in some other resource. For one example, when searching for the best configuration for the learning algorithm of a large convolutional neural network, a large amount of computational work is required to evaluate a candidate configuration, and the space of possible configurations is high-dimensional. Another common example, alluded to in Section 3.2.3, is data acquisition: for machine learning problems in which a large body of data is available, it is often desirable to choose the right queries to produce a data set on which learning will be most effective. In continuous settings, many Bayesian optimization methods employ GPs (e.g. Snoek et al., 2012).

We have implemented a version of Thompson sampling using GPs in Venture. Thompson sampling (Thompson, 1933) is a widely-used Bayesian framework for solving exploration-exploitation problems. Our implementation has two notable features: (i) the ability to search over a broader space of contexts than the parametric families that are typically used, and (ii) the parsimony of the resulting probabilistic program.

4.1 Thompson sampling framework

We now lay out the setup of Thompson sampling for Markov decision processes (MDPs). An agent is to take a sequence of actions a_1, a_2, \dots from a (possibly infinite) set of possible actions \mathcal{A} . After each action, a reward $r \in \mathbb{R}$ is received, according to an unknown conditional distribution $P_{\text{true}}(r|a)$. The agent's goal is to maximize the total reward received for all actions. In Thompson sampling, the Bayesian agent accomplishes this by placing a prior distribution $P(\theta)$ on the possible "contexts" $\theta \in \Theta$. Here a context is a believed model of the conditional distributions $\{P(r|a)\}_{a \in \mathcal{A}}$, or at least, a believed statistic of these conditional distributions which is sufficient for deciding an action a . One example of such a sufficient statistic is the conditional mean $V(a|\theta) = \mathbb{E}[r|a, \theta]$, which can be

648 thought of as a value function. Thompson sampling thus has the following steps, repeated as long
 649 as desired:

- 651 1. Sample a context $\theta \sim P(\theta)$.
- 652 2. Choose an action $a \in \mathcal{A}$ which (approximately) maximizes $V(a|\theta) = \mathbb{E}[r|a, \theta]$.
- 653 3. Let r_{true} be the reward received for action a . Update the believed distribution on θ , i.e.,
 654 $P(\theta) \leftarrow P_{\text{new}}(\theta)$ where $P_{\text{new}}(\theta) = P(\theta | a \mapsto r_{\text{true}})$.

655 Note that when $\mathbb{E}[r|a, \theta]$ (under the sampled value of θ for some points a) is far from the true value
 656 $\mathbb{E}_{P_{\text{true}}}[r|a]$, the chosen action a may be far from optimal, but the information gained by probing
 657 action a will improve the belief θ . This amounts to “exploration.” When $\mathbb{E}[r|a, \theta]$ is close to the
 658 true value except at points a for which $\mathbb{E}[r|a, \theta]$ is low, exploration will be less likely to occur, but
 659 the chosen actions a will tend to receive high rewards. This amounts to “exploitation.” Roughly
 660 speaking, exploration will happen until the context θ is reasonably sure that the unexplored actions
 661 are probably not optimal, at which time the sampler will exploit by choosing actions in regions it
 662 knows to have high value.

663 Typically, when Thompson sampling is implemented, the search over contexts $\theta \in \Theta$ is limited
 664 by the choice of representation. In traditional programming environments, θ often consists of a few
 665 numerical parameters for a family of distributions of a fixed functional form. With work, a mixture of
 666 a few functional forms is possible; but without probabilistic programming machinery, implementing
 667 a rich context space Θ would be an unworkably large technical burden. In a probabilistic programming
 668 language, however, the representation of heterogeneously structured or infinite-dimensional context
 669 spaces is quite natural. Any computable model of the conditional distributions $\{P(r|a)\}_{a \in \mathcal{A}}$ can be
 670 represented as a stochastic procedure $(\lambda(a) \dots)$. Thus, for computational Thompson sampling, the
 671 most general context space $\widehat{\Theta}$ is the space of program texts. Any other context space Θ has a natural
 672 embedding as a subset of $\widehat{\Theta}$.

674 4.2 Thompson sampling in Venture

675 Because Venture supports sampling and inference on (stochastic-)procedure-valued random variables
 676 (and the generative models which produce those procedures), Venture can capture arbitrary
 677 context spaces as described above. To demonstrate, we have implemented Thompson sampling
 678 in Venture in which the contexts θ are Gaussian processes over the action space $\mathcal{A} = \mathbb{R}$. That
 679 is, $\theta = (\mu, K)$, where the mean μ is a computable function $\mathcal{A} \rightarrow \mathbb{R}$ and the covariance K is
 680 a computable (symmetric, positive-semidefinite) function $\mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}$. This represents a Gaussian
 681 process $\{R_a\}_{a \in \mathcal{A}}$, where R_a represents the reward for action a . Computationally, we represent
 682 a context not as a pair of infinite lookup tables for μ and K , but as a finite data structure
 683 $\theta = (K_{\text{prior}}, \sigma, \ell, \mathbf{a}_{\text{past}}, \mathbf{r}_{\text{past}})$, where

- 684 • $K_{\text{prior}} = K_{\text{prior}, \sigma, \ell}$ is a procedure, with parameters σ, ℓ , to be used as the prior covariance
 685 function: $K_{\text{prior}}(a, a') = \sigma^2 \exp\left(-\frac{(a-a')^2}{2\ell^2}\right)$
- 686 • σ and ℓ are (hyper)parameters for K_{prior}
- 687 • $\mathbf{a}_{\text{past}} = (a_i)_{i=1}^n$ are the previously probed actions
- 688 • $\mathbf{r}_{\text{past}} = (r_i)_{i=1}^n$ are the corresponding rewards

689 To simplify the treatment, we take prior mean $\mu_{\text{prior}} \equiv 0$. The mean and covariance for θ are then
 690 gotten by the usual conditioning formula:

$$\begin{aligned}
 691 \mu(\mathbf{a}) &= \mu(\mathbf{a} | \mathbf{a}_{\text{past}}, \mathbf{r}_{\text{past}}) \\
 692 &= K_{\text{prior}}(\mathbf{a}, \mathbf{a}_{\text{past}}) K_{\text{prior}}(\mathbf{a}_{\text{past}}, \mathbf{a}_{\text{past}})^{-1} \mathbf{r}_{\text{past}} \\
 693 K(\mathbf{a}, \mathbf{a}) &= K(\mathbf{a}, \mathbf{a} | \mathbf{a}_{\text{past}}, \mathbf{r}_{\text{past}}) \\
 694 &= K_{\text{prior}}(\mathbf{a}, \mathbf{a}) - K_{\text{prior}}(\mathbf{a}, \mathbf{a}_{\text{past}}) K_{\text{prior}}(\mathbf{a}_{\text{past}}, \mathbf{a}_{\text{past}})^{-1} K_{\text{prior}}(\mathbf{a}_{\text{past}}, \mathbf{a}).
 \end{aligned}$$

695 Note that even in this simple example, the context space Θ is not a finite-dimensional parametric
 696 family, since the vectors \mathbf{a}_{past} and \mathbf{r}_{past} grow as more samples are taken. Θ is, however, quite easily
 697 representable as a computational procedure together with parameters and past samples, as we do in
 698 the representation $\theta = (K_{\text{prior}}, \sigma, \ell, \mathbf{a}_{\text{past}}, \mathbf{r}_{\text{past}})$.

4.3 Implementation with gpmem

As a demonstration, we use Thompson sampling to optimize an unknown function $V(x)$ (the value function) using `gpmem`. (**TODO** we should not assume V is deterministic, it would be easy enough to make it random or have it give noisy samples.) We assume V is made available to Venture as a black-box. The code for optimizing V is given in Listing ???. For step 3 of Thompson sampling, the Bayesian update, we not only condition on the new data (the chosen action a and the received reward r), but also perform inference on the hyperparameters σ, ℓ using a Metropolis–Hastings sampler. These two inference steps take 1 line of code: 0 lines to condition on the new data (as this is done automatically by `gpmem`), and 1 line to call Venture’s built-in MH operator. The results are shown in Figure ???. We can see from the figure that, roughly speaking, each successive probe point a is chosen either because the current model V_{emu} thinks it will have a high reward, or because the value of $V_{\text{emu}}(a)$ has high uncertainty. In the latter case, probing at a decreases this uncertainty and, due to the smoothing kernel, also decreases the uncertainty at points near a . We thus see that our Thompson sampler simultaneously learns the value function and optimizes it.

5 Conclusion

We have shown Venture GPs. We have introduced novel stochastic processes for a probabilistic programming language. We showed how flexible non-parametric models can be treated in Venture in only a few lines of code. We evaluated our contribution on a range of hard problems for state-of-the-art Bayesian non-parametrics. Venture GPs showed competitive performance in all of them.

Appendix

A Covariance Functions

SE and WN are defined in the text above, for completeness we will introduce the covariance:

$$k_{LIN}(x, x') = \theta(xx') \quad (23)$$

$$k_{PER}(x, x') = \theta \exp\left(\frac{2 \sin^2(\pi(x - x')/p)}{\ell^2}\right) \quad (24)$$

$$k_{RQ}(x, x') = \theta \left(1 + \frac{(x - x')^2}{2\alpha\ell^2} \right)^{-\alpha} \quad (25)$$

B Covariance Simplification

$$\begin{array}{ll}
 \text{SE} \times \text{SE} & \rightarrow \text{SE} \\
 \{\text{SE}, \text{PER}, \text{C}, \text{WN}\} \times \text{WN} & \rightarrow \text{WN} \\
 \text{LIN} + \text{LIN} & \rightarrow \text{LIN} \\
 \{\text{SE}, \text{PER}, \text{C}, \text{WN}, \text{LIN}\} \times \text{C} & \rightarrow \{\text{SE}, \text{PER}, \text{C}, \text{WN}, \text{LIN}\}
 \end{array}$$

Rule 1 is derived as follows:

$$\begin{aligned}
\sigma_c^2 \exp\left(-\frac{(x-x')^2}{2\ell_c^2}\right) &= \sigma_a^2 \exp\left(-\frac{(x-x')^2}{2\ell_a^2}\right) \times \sigma_b^2 \exp\left(-\frac{(x-x')^2}{2\ell_b^2}\right) \\
&= \sigma_c^2 \exp\left(-\frac{(x-x')^2}{2\ell_a^2}\right) \times \exp\left(-\frac{(x-x')^2}{2\ell_b^2}\right) \\
&= \sigma_c^2 \exp\left(-\frac{(x-x')^2}{2\ell_a^2} - \frac{(x-x')^2}{2\ell_b^2}\right) \\
&= \sigma_c^2 \exp\left(-\frac{(x-x')^2}{2\ell_c^2}\right)
\end{aligned} \tag{26}$$

Rule 3 is derived as follows:

$$\theta_c(x \times x') = \theta_a(x \times x') + \theta_b(x \times x') \quad (27)$$

756 For stationary kernels that only depend on the lag vector between x and x' it holds that multiplying
757 such a kernel with a WN kernel we get another WN kernel. Take for example the SE kernel:
758

$$759 \sigma_a^2 \exp\left(-\frac{(x - x')^2}{2\ell_c^2}\right) \times \sigma_b \delta_{x,x'} = \sigma_a \sigma_b \delta_{x,x'} \quad (28)$$

760

761 Multiplying any kernel with a constant obviously changes only the scale parameter of a kernel.
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809

810 **References**
811

- 812 Andrieu, C., De Freitas, N., Doucet, A., and Jordan, M. I. (2003). An introduction to MCMC for
813 machine learning. *Machine learning*, 50(1-2):5–43.
- 814 Benavoli, A. and Mangili, F. (2015). Gaussian processes for bayesian hypothesis tests on regression
815 functions. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence
816 and Statistics*, pages 74–82.
- 817 Box, G. E. P., Jenkins, G. M., and Reinsel, G. C. (1997). *Time series analysis: forecasting and
818 control*.
- 819 Duvenaud, D., Lloyd, J. R., Grosse, R., Tenenbaum, J., and Ghahramani, Z. (2013). Structure
820 discovery in nonparametric regression through compositional kernel search. In *Proceedings of
821 the 30th International Conference on Machine Learning (ICML-13)*, pages 1166–1174.
- 822 Ghahramani, Z. (2015). Probabilistic machine learning and artificial intelligence. *Nature*,
823 521(7553):452–459.
- 824 Goodman, N. D .and Mansinghka, V. K., Roy, D., Bonawitz, K., and Tenenbaum, J. (2008). Church:
825 A language for generative models. In *Proceedings of the 24th Conference on Uncertainty in
826 Artificial Intelligence, UAI 2008*, pages 220–229.
- 827 Lloyd, J. R., Duvenaud, D., Grosse, R., Tenenbaum, J., and Ghahramani, Z. (2014). Automatic
828 construction and natural-language description of nonparametric regression models. In *Twenty-
829 Eighth AAAI Conference on Artificial Intelligence*.
- 830 Mansinghka, V. K., Selsam, D., and Perov, Y. (2014). Venture: a higher-order probabilistic pro-
831 gramming platform with programmable inference. *arXiv preprint arXiv:1404.0099*.
- 832 McAllester, D., Milch, B., and Goodman, N. D. (2008). Random-world semantics and syntactic
833 independence for expressive languages. Technical report.
- 834 Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation
835 of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–
836 1092.
- 837 Neal, R. M. (1997). Monte carlo implementation of gaussian process models for bayesian regression
838 and classification. *arXiv preprint physics/9701026*.
- 839 Poole, D. (1993). Probabilistic horn abduction and bayesian networks. *Artificial Intelligence*,
840 64(1):81–129.
- 841 Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning (Adap-
842 tive Computation and Machine Learning)*. The MIT Press.
- 843 Sato, T. (1995). A statistical learning method for logic programs with distribution semantics. In *In
844 Proceedings of the 12th International Conference on Logic Programming*. Citeseer.
- 845 Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine
846 learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959.
- 847 Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view
848 of the evidence of two samples. *Biometrika*, pages 285–294.

851
852
853
854
855
856
857
858
859
860
861
862
863