
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 provides 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

- 082 • standard memoization
- 083 • memoization as described in (Goodman et al., 2008)

085 2.2 Venture

087 Venture is a compositional language for custom inference strategies that comes with a Scheme- and
 088 Java-Script-like front-end syntax. Its implementation is based on on three concepts. (i) stochas-
 089 tic procedure interfaces that specify and encapsulate random variables, analogously to conditional
 090 probability tables in a Bayesian network; (ii) probabilistic execution traces that represent execution
 091 histories and capture conditional dependencies; and (iii) scaffolds that partition execution histories
 092 and factor global inference problems into sub-problems. These building blocks provide a powerful
 093 way to represent probability distributions; some of which cannot be expressed with density func-
 094 tions. For the purpose of this work the most important Venture directives that operate on these
 095 building blocks to understand are ASSUME, OBSERVE, SAMPLE and INFER. ASSUME induces
 096 a hypothesis space for (probabilistic) models including random variables by binding the result of an
 097 expression to a symbol. SAMPLE simulates a model expression and returns a value. OBSERVE
 098 adds constraints to model expressions. INFER instructions incorporate observations and cause Ven-
 099 ture to find a hypothesis that is probable given the data.

100 INFER is most commonly done by deploying the Metropolis-Hastings algorithm (MH) (Metropolis
 101 et al., 1953). Many algorithms used in the MCMC world can be interpreted as special cases of
 102 MH (Andrieu et al., 2003). We can outline the MH algorithm as follows. For T steps we sample x^*
 103 from a proposal distribution q :

$$x^* \sim q(x^* | x^{(t)}) \quad (1)$$

104 which we accept ($x^{t+1} \leftarrow x^*$) with ratio:

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

105 106 107 Venture implements an MH transition operator for probabilistic execution traces.

108 **2.3 Gaussian Processes**
109

110 In the following, we will introduce GP related theory and notations. We will exclusively work on
111 two variable regression problems. Let the data be real-valued scalars $\{x_i, y_i\}_{i=1}^n$ (complete data will
112 be denoted by column vectors \mathbf{x}, \mathbf{y}). GPs present a non-parametric way to express prior knowledge
113 on the space of possible functions f that we assume to have generated the data. f is assumed latent
114 and the GP prior is given by a multivariate Gaussian $f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(x_i, x'_i))$, where $m(\mathbf{x})$ is
115 a function of the mean of all functions that map to y_i at x_i and $k(x_i, x'_i)$ is a kernel or covariance
116 function that summarizes the covariance of all functions that map to y_i at x_i . We can absorb the
117 mean function into the covariance function so without loss of generality we can set the mean to
118 zero. The marginal likelihood can be expressed as:

$$119 \quad p(\mathbf{y}|\mathbf{x}) = \int p(\mathbf{y}|\mathbf{f}, \mathbf{x}) p(\mathbf{f}|\mathbf{x}) d\mathbf{f} \quad (3)$$

120 where the prior is Gaussian $\mathbf{f}|\mathbf{x} \sim \mathcal{N}(0, k(\mathbf{x}, \mathbf{x}'))$. We can sample a vector of unseen data from the
121 predictive posterior with

$$122 \quad \mathbf{y}^* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad (4)$$

123 for a zero mean prior GP with a posterior mean of:

$$124 \quad \boldsymbol{\mu} = \mathbf{K}(\mathbf{x}, \mathbf{x}^*) \mathbf{K}(\mathbf{x}^*, \mathbf{x}^*)^{-1} \mathbf{y} \quad (5)$$

125 and covariance

$$126 \quad \boldsymbol{\Sigma} = \mathbf{K}(\mathbf{x}, \mathbf{x}) + \mathbf{K}(\mathbf{x}, \mathbf{x}^*) \mathbf{K}(\mathbf{x}^*, \mathbf{x}^*)^{-1} \mathbf{K}(\mathbf{x}^*, \mathbf{x}). \quad (6)$$

127 \mathbf{K} is a covariance function. The log-likelihood is defined as:

$$128 \quad \log P(\mathbf{y} | \mathbf{X}) = -\frac{1}{2}\mathbf{y}^\top (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{y} - \frac{1}{2} \log |\mathbf{K} + \sigma^2 \mathbf{I}| - \frac{n}{2} \log 2\pi \quad (7)$$

129 with n being the number of data-points and sigma the independent observation noise. Both log-
130 likelihood and predictive posterior can be computed efficiently in a Venture SP with an algorithm
131 that resorts to Cholesky factorization(Rasmussen and Williams, 2006, chap. 2) resulting in a com-
132 putational complexity of $\mathcal{O}(n^3)$ in the number of data-points.

133 The covariance function covers general high-level properties of the observed data such as linear-
134 ity, periodicity and smoothness. The most widely used type of covariance function is the squared
135 exponential covariance function:

$$136 \quad k(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right) \quad (8)$$

137 where σ and ℓ are hyper-parameters. σ is a scaling factor and ℓ is the typical length-scale. Smaller
138 variations can be achieved by adapting these hyper-parameters.

139 Larger variations are achieved by changing the type of the covariance function structure. Note that
140 covariance function structures are compositional. We can add covariance functions if we want to
141 model globally valid structures

$$142 \quad k_3(x, x') = k_1(x, x') + k_2(x, x') \quad (9)$$

143 and we can multiply covariance functions if the data is best explained by local structure

$$144 \quad k_4(x, x') = k_1(x, x') \times k_2(x, x'); \quad (10)$$

145 both, k_3 and k_4 are valid covariance function structures.

146 **3 Venture GPs**
147

148 Given a stochastic process that implements the GP algebra above we can imple-
149 ment a GP sampler (4) to perform GP inference in a few lines of code. We
150 can express simple GP smoothing with fixed hyper-parameters or a prior on hyper-
151 parameters and perform MH on it while allowing users to custom design covari-

```

1  ance functions. Throughout the paper, we will use the Scheme-like front-end syntax.
162
163
164 [ASSUME l (gamma 1 3)] ∈ {hyper-parameters}
165 [ASSUME sf (gamma 1 3)] ∈ {hyper-parameters}
166
167  $k(x, x') := \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$ 
168
169 [ASSUME f VentureFunction(k, σ, ℓ) ]
170 [ASSUME SE make-se (apply-function f l sf) ]
171 [ASSUME (make-gp 0 SE) ]
172
173 [SAMPLE GP (array 1 2 3)] % Prior
174 [OBSERVE GP D]
175 [SAMPLE GP (array 1 2 3)]
176 [INFER (MH {hyper-parameters} one 100) ]
177 [SAMPLE GP (array 1 2 3)] % Posterior

```

Listing 1: Bayesian GP Smoothing

The first two lines depict the hyper-parameters. We tag both of them to belong to the set {hyper-parameters}. Every member of this set belongs to the same inference scope. This scope controls the application of the inference procedure used. In this paper, we use MH throughout. Each scope is further subdivided into blocks that allow to do block-proposals. In the following we omit the block notation for readability, since we always choose the block of a certain scope at random.

The ASSUME directives describe the assumptions we make for the GP model, we assume the hyper-parameters l and sf (corresponding to ℓ, σ) to be 1 and 2. The squared exponential covariance function can be defined outside the Venture code with foreign conventional programming languages, e.g. Python. In that way, the user can define custom covariance functions without being restricted to the most common ones. We then integrate the foreign function into Venture as VentureFunction. In the next line this function is associated with the hyper-parameters. Finally, we assume a Gaussian Process SP with a zero mean and the previously assumed squared exponential covariance function.

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 1. 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 A Bayesian interpretation

3.1.1 GP 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 defined as a foreign procedure made available as a black-box to Venture:

```

207 def f_restr(x):
208     if x in D:
209         return D[x]
210     else:
211         raise Exception('Illegal input')

```

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

```

212 [ASSUME (list f_compute f_emu) (gpmem f_restr) ]
213 for i=1 to n:

```

```

216 [PREDICT (f_compute x[i])]
217 [INFER (MH {hyper-parameters} one 100)]
218 [SAMPLE (f_emu (array 1 2 3))]
219
220

```

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

228 More generally, `gpmem` is relevant not just when a data set is available, but also whenever we have
 229 at hand a function f_{restr} which is expensive or impractical to evaluate many times. `gpmem` allows us
 230 to model f_{restr} with a GP-based emulator f_{emu} , and also to use f_{emu} during the learning process to
 231 choose, in an online manner, an effective set of probe points $\{x_i\}$ on which to use our few evaluations
 232 of f_{restr} . This idea is illustrated in detail in Section 4. First, we will show how one can utilize `gpmem`
 233 for reproducing state-of-the-art models that are based on GP.

236 3.1.2 The efficacy of learning hyperparameters

237
 238 The probability of the hyper-parameters of a GP with assumptions as above and given covariance
 239 function structure \mathbf{K} can be described as:

$$243 P(\boldsymbol{\theta} | \mathbf{D}, \mathbf{K}) = \frac{P(\mathbf{D} | \boldsymbol{\theta}, \mathbf{K})P(\boldsymbol{\theta} | \mathbf{K})}{P(\mathbf{D} | \mathbf{K})}. \quad (11)$$

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

$$255 \ell^{(t)} \sim \Gamma(\alpha_1, \beta_1), \sigma^{(t)} \sim \Gamma(\alpha_2, \beta_2) \quad (12)$$

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

$$263 \alpha_1^{(t)} \sim \Gamma(\alpha_\alpha^1, \beta_\alpha^1), \alpha_2^{(t)} \sim \Gamma(\alpha_\alpha^2, \beta_\alpha^2), \dots \quad (13)$$

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

```

270 Assuming the covariance structure is an additive comprised of a smoothing and a white noise
271 kernel, one can represent this kind of model using gpmem with only a few lines of code:
272
1 [ASSUME alpha (mem (lambda (i) (gamma 1 3 )))] ∈ {hyper-parameters-Γ}
2 [ASSUME beta (mem (lambda (i) (gamma 1 3 )))] ∈ {hyper-parameters-Γ}
273
3
4 [ASSUME l (gamma (alpha 1) (beta 1))] ∈ {hyper-parameters}
5 [ASSUME sf (gamma (alpha 2) (beta 2))] ∈ {hyper-parameters}
6 [ASSUME sigma (uniform 0 5 )] ∈ {hyper-parameters}
274
7
8  $k_1(x, x') := \theta^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$ 
9  $k_2(x, x') := \sigma^2 \delta_{x,x'}$ 
275
10
11 [ASSUME k1 VentureFunction(k1, θ, ℓ) ]
12 [ASSUME k2 VentureFunction(k2, σ) ]
276
13
14 [ASSUME SE make-se (apply-function k1 l sf) ]
15 [ASSUME WN make-se (apply-function k1 sigma) ]
277
16
17 [ASSUME (list f_compute f_emu) (gpmem f_restr (function-plus SE WN) )]
18 [SAMPLE (f_emu (array 1 2 3))] % prior, Fig. 1b
278
19
20 for i=1 to n:
21   [PREDICT (f_compute x[i])] % observing with a look-up function
22 [SAMPLE (f_emu (array 1 2 3))] % after observation, Fig. 1c
279
23
24 [INFER (REPEAT 100
25   (DO (MH {hyper-parameters} one 2)
26     (MH {hyper-parameters-Γ} one 2) ))
27 [SAMPLE (f_emu (array 1 2 3))] % Posterior , Fig. 1d
280
295
296
```

Listing 2: Hierarchical GP Smoothing

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:

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

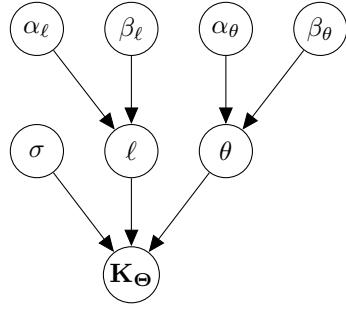
We synthetically generate outliers by setting $\sigma = 0.1$ in 95% of the cases and to $\sigma = 1$ in the remaining cases. gpmem can capture the true underlying function within only 100 MH steps on the hyper-parameters to get a good approximation for their posterior (see Fig. 1). Note that Neal devices 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 outliers².

3.2 Structure Learning

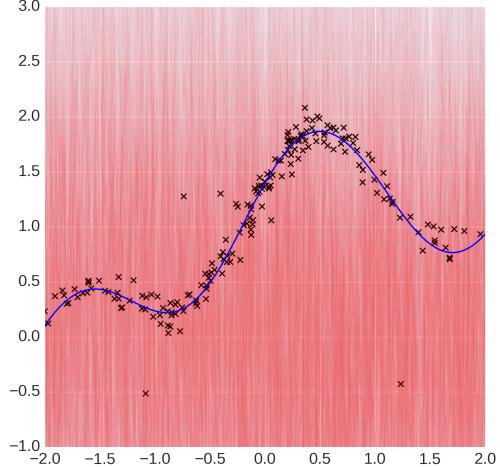
The space of possible kernel composition is infinite. Combining inference over this space with the problem of finding a good parameterization that could potentially explain the observed data best poses a hard problem. The natural language interpretation of the meaning of a kernel and its composition renders this a problem of symbolic computation. Duvenaud and colleagues note that 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 kernel results in high values only if k_1 AND k_2 have high values (see Fig. 3 for examples how to interpret global vs. local aspects and its symbolic analog respectively).

²For this pedagogical example we have increased the probability for outliers in the data generation slightly from 0.05 to 0.2

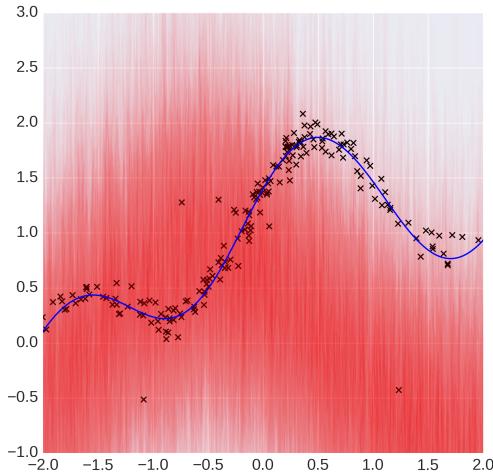
324
 325
 326
 327
 328
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 340
 341
 342
 343
 344
 345
 346



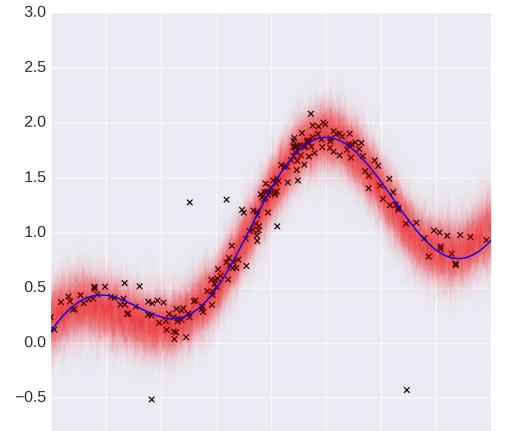
(a) Hierarchical Prior



(b) Prior Inference



(c) Observed



(d) Inferred

367
 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 a Venture GP 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.

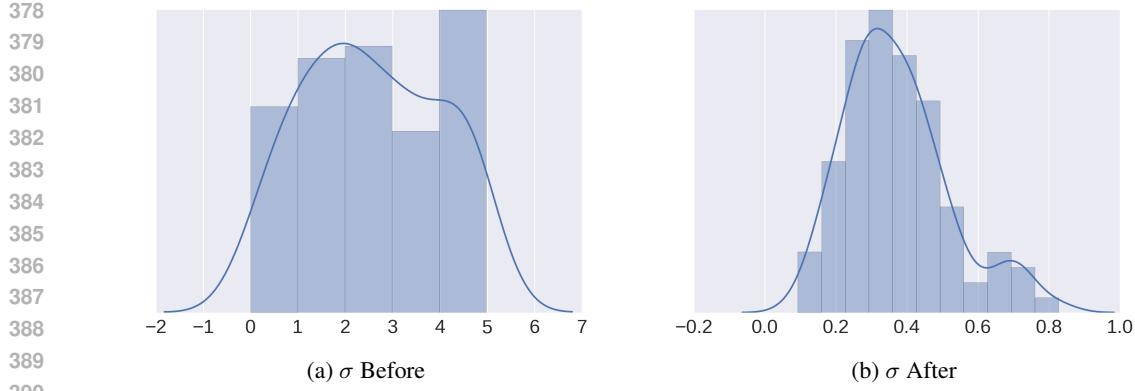


Figure 2: Hyper-parameter inference on the parameter of the noise kernel. We show 100 samples drawn from the distribution on σ . One can clearly recognise the shift from the uniform prior $\mathcal{U}(0, 5)$ to a double peak distribution around the two modes - normal and outlier.

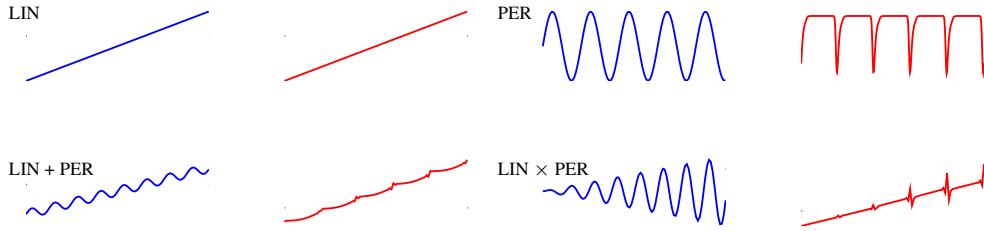


Figure 3: Composition of covariance functions (blue, left) and samples from the distribution of curves they can produce (red, right).

In the following, we will refer to covariance functions that are not composite as base covariance functions. Note that this form of composition can be easily expressed in Venture, for example if one wishes to add a linear and a periodic kernel:

```

1 [ASSUME l (gamma 1 3)]
2 [ASSUME sf (gamma 1 2)]
3 [ASSUME a (gamma 2 2)]
4
5  $k_{LIN}(x, x') := \sigma_1^2(x - \ell)(x' - \ell)$ 
6  $k_{PER}(x, x') := \sigma_2^2 \exp(-\frac{2 \sin^2(\pi(x-x')/p)}{\sigma_2^2})$ 
7
8 [ASSUME fLIN VentureFunction(kLIN, σ1) ]
9 [ASSUME fPER VentureFunction(kPER, σ2, ℓ, p) ]
10 [ASSUME LIN (make-LIN (apply-function fLIN a)) ]
11 [ASSUME PER (make-PER (apply-function fPER 1 sf)) ]
12 [ASSUME (make-gp 0 (function-times LIN PER)) ]

```

Listing 3: LIN × PER

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 the expectation for the prediction:

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

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

$$434 \quad 435 \quad 436 \quad y^* \approx \frac{1}{T} \sum_{t=1}^T f(x^* | \boldsymbol{\theta}^{(t)}, \mathbf{K}^{(t)}). \quad (16)$$

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

$$440 \quad \mathbf{K}^{(t)} \sim P(\mathbf{K} | \boldsymbol{\Omega}, s, n) \quad (17)$$

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

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

$$448 \quad P(\mathbf{K} | \boldsymbol{\Omega}, s, n) = P(\boldsymbol{\Omega} | s, n) \times P(s | n) \times P(n), \quad (18)$$

449 with

$$450 \quad 451 \quad P(\boldsymbol{\Omega} | s, n) = \binom{n}{r} p_{+ \times}^k (1 - p_{+ \times})^{n-k} \quad (19)$$

452 and

$$453 \quad P(s | n) = \frac{n!}{|s|!} \quad (20)$$

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

460 Many equivalent covariance structures can be sampled due to covariance function algebra
 461 and equivalent representations with different parameterization (Lloyd et al., 2014). Certain
 462 covariance functions can differ in terms of the hyper-parameterization but can be
 463 absorbed into a single covariance function with a different parameterization. To inspect
 464 the posterior of these equivalent structures we convert each kernel expression into
 465 a sum of products and subsequently simplify expressions using the following grammar:

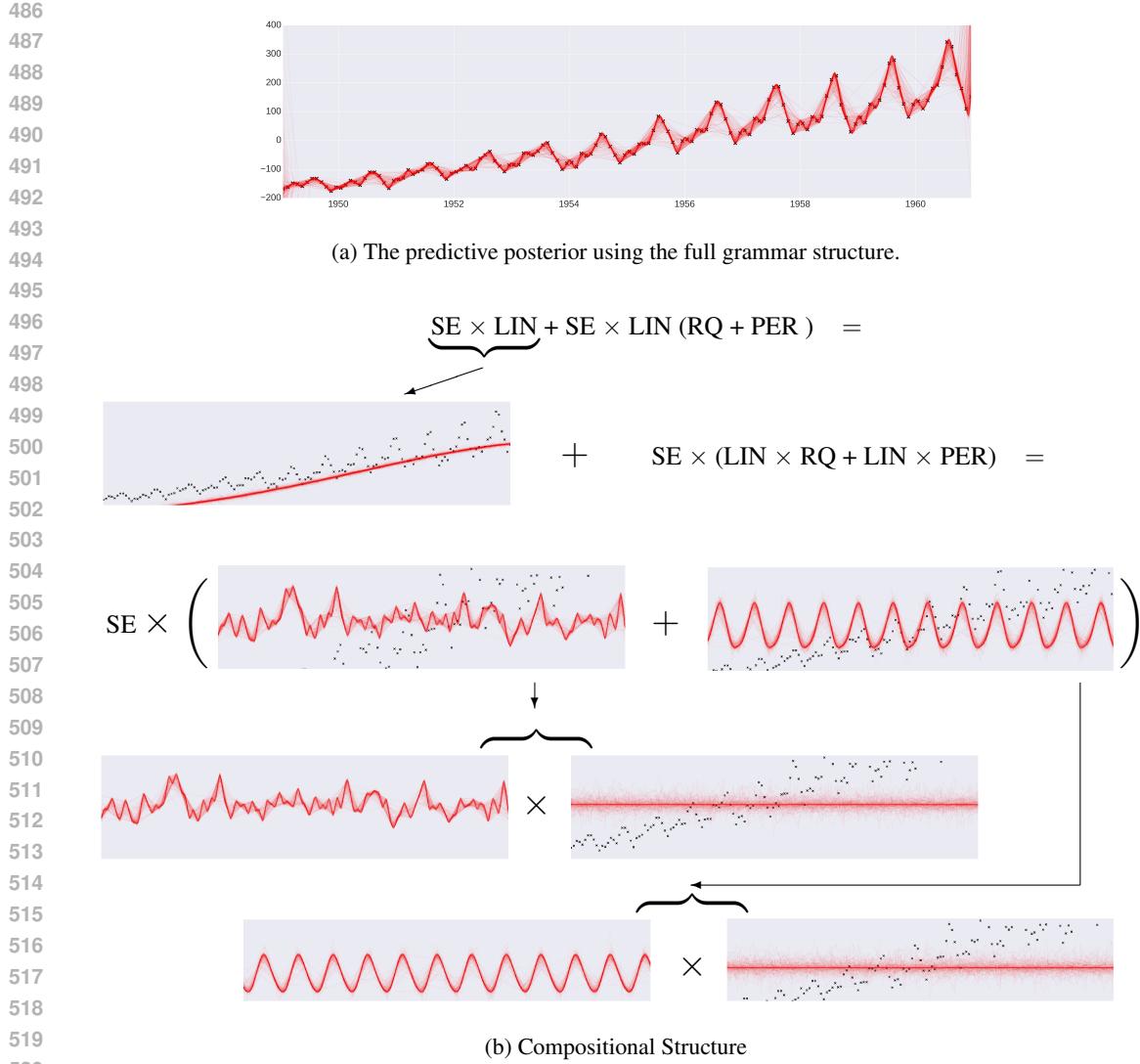
```
466 1 SE × SE → SE
 2 {SE, PER, C, WN} × WN → WN
 3 LIN + LIN → LIN
 4 {SE, PER, C, WN, LIN} × C → {SE, PER, C, WN, LIN}
```

470 Listing 4: Grammar to simplify expressions

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

```
476 1 [ASSUME S (array K1, K2, ..., Kn)] // (defined as above)
 2 [ASSUME p_n (uniform_structure n)]
 3 [ASSUME S (array K1, K2, ..., Kn)]
 4 [ASSUME K* (grammar S p_n)]
 5 [ASSUME GP (make-gp 0 K*)]
 6
 7 [OBSERVE GP D]
 8
 9 [INFER (REPEAT 2000 (DO
 10   (MH 10 p_n one 1)
 11   (MH 10 K* one 1)
 12   (MH 10 {hyper-parameters} one 10)) ]
```

485 Listing 5: Venture Code for Bayesian GP Structure Learning



521 Figure 4: a) We see the predictive posterior as a result 1000 nested MH steps on the airline data
 522 set. b) depicts a decomposition of this posterior for the structures sampled by Venture.
 523 RQ is the rational quadratic covariance function. The first line shows the global trend and denotes the rest of
 524 the structure that is shown above. In the second line, the see the periodic component on the right
 525 hand side. The left hand side denotes short term deviations both multiplied by a smoothing kernel.
 526 The third and fourth lines denote how we reach the second line: both periodic and rational quadratic
 527 covariance functions are multiplied by a linear covariance function with slope zero.

531 We defined the space of covariance structures in a way allowing us to reproduce results for covari-
 532 ance function structure learning as in the Automated Statistician. This lead to coherent results, for
 533 example for the airline data set. We will elaborate the result using a sample from the posterior (Fig.
 534 4). The sample is identical with the highest scoring result reported in previous work using a search-
 535 and-score method (Duvenaud et al., 2013) for the CO₂ data set () and the predictive capability is
 536 comparable. However, the components factor in a different way due to different parameterization of
 537 the individual base kernels.

538 We further investigated the quality of our stochastic processes by running a leave one out cross-
 539 validation to gain confidence on the posterior. This resulted in 545 independent runs of the Markov
 chain that produced a coherent posterior: our Bayesian interpretation of GP structure and GPs pro-

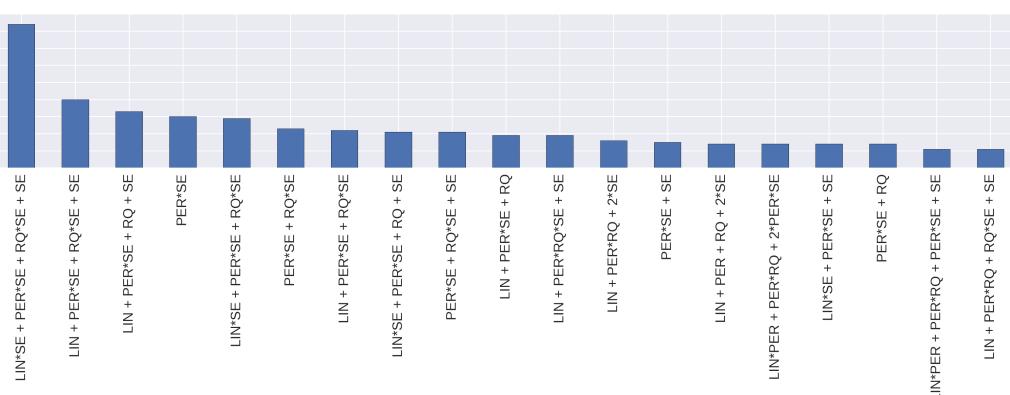


Figure 5: Posterior on structure of the CO2 data. We have cut the tail of the distribution for space reasons since the number of possible structures is large. We see the final sample of the each of the 545 chains with 2000 nested steps each. Note that Duvenaud et al. (2013) report $\text{LIN} \times \text{SE} + \text{PER} \times \text{SE} + \text{RQ} \times \text{SE}$.

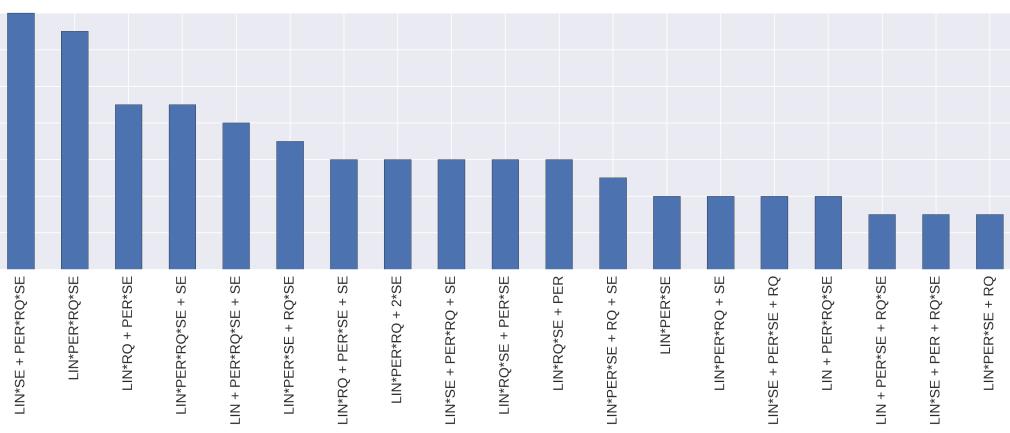


Figure 6: Posterior on structure of airline data set. We have cut the tail of the distribution for space reasons since the number of possible structures is large. We see the final sample of the each of the 144 chains with 2000 nested steps each. Note that Duvenaud et al. (2013) report $\text{LIN} \times \text{SE} + (\text{PER} + \text{RQ}) \times \text{SE} \times \text{LIN}$

duced a posterior of structures that is in line with previous results on this data set (Duvenaud et al., 2013; see Fig. 8).

We ran similar evaluation on the airline data set () resulting in a similar structure to what was previously reporte (Fig. 6, residuals and log-score along the Markov chain see Fig. 7).

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

4 Bayesian Optimization

Bayesian Optimization poses the problem of finding the global maximum of an unknown function as a hierarchical decision problem (Ghahramani, 2015). Evaluating the actual function can be very expensive. For example, finding the best configuration for the learning algorithm of a large convolutional neural network implies expensive function evaluations to compare a potentially infinite number of configurations. Another common example is the example of data acquisition. For problems with large amounts of data available it may be interesting to chose certain informative

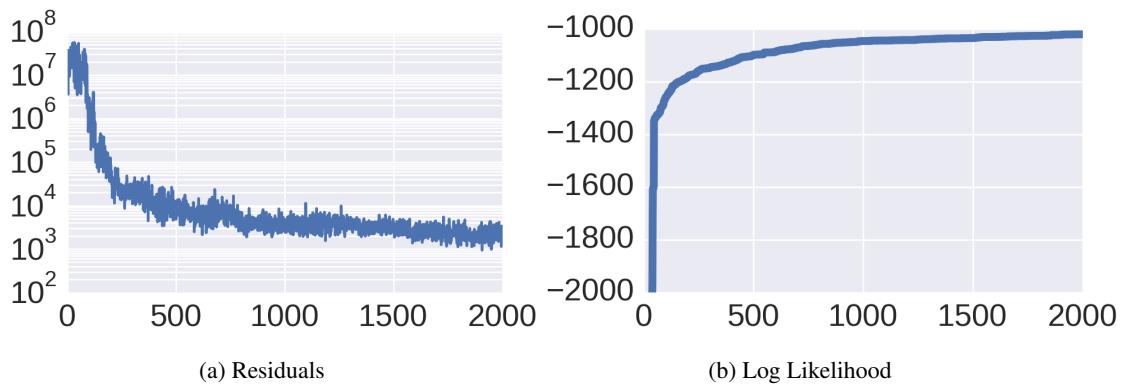


Figure 7: 2000 steps along the Markov Chain.

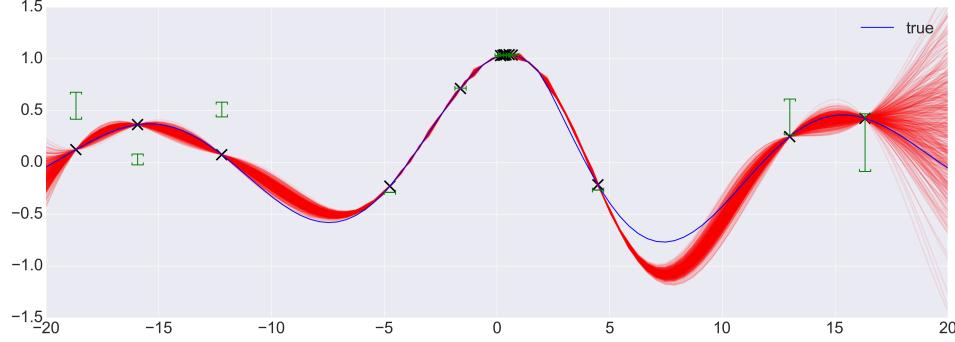


Figure 8: Bayesian Optimization. Each successive probe point x is the (stochastic) maximum of a GP-based emulator conditioned on the values of the previously probed points. In the figure, each probe point x is marked with an \times , and a vertical green bar is drawn showing the mean \pm one standard deviation of the “leave-one-out” distribution—the distribution that would arise from the same covariance function if all marked points *except* x had been probed. Note that there are many probe points near the true maximum, and the uncertainty is quite low. Also note that probed points far away from the true maximum tend to be points at which the uncertainty is high.

data-points to evaluate a model on. In continuous domains, many Bayesian Optimization methods deploy GPs (e.g. Snoek et al., 2012).

The hierarchical nature of Bayesian Optimization makes it an ideal application for GPs in Venture. The following Bayesian Optimization scheme is closely related to Thompson Sampling Thompson (1933). Thompson Sampling is a general framework to solve exploration-exploitation problems that applies to our notion of Bayesian Optimization.

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.

648 **References**
649

- 650 Andrieu, C., De Freitas, N., Doucet, A., and Jordan, M. I. (2003). An introduction to mcmc for
651 machine learning. *Machine learning*, 50(1-2):5–43.
652 Duvenaud, D., Lloyd, J. R., Grosse, R., Tenenbaum, J., and Ghahramani, Z. (2013). Structure
653 discovery in nonparametric regression through compositional kernel search. In *Proceedings of*
654 *the 30th International Conference on Machine Learning (ICML-13)*, pages 1166–1174.
655 Ghahramani, Z. (2015). Probabilistic machine learning and artificial intelligence. *Nature*,
656 521(7553):452–459.
657 Goodman, N. D .and Mansinghka, V. K., Roy, D., Bonawitz, K., and Tenenbaum, J. (2008). Church:
658 A language for generative models. In *Proceedings of the 24th Conference on Uncertainty in*
659 *Artificial Intelligence, UAI 2008*, pages 220–229.
660 Lloyd, J. R., Duvenaud, D., Grosse, R., Tenenbaum, J., and Ghahramani, Z. (2014). Automatic
661 construction and natural-language description of nonparametric regression models. In *Twenty-*
662 *Eighth AAAI Conference on Artificial Intelligence*.
663 Mansinghka, V. K., Selsam, D., and Perov, Y. (2014). Venture: a higher-order probabilistic pro-
664 gramming platform with programmable inference. *arXiv preprint arXiv:1404.0099*.
665 Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation
666 of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–
667 1092.
668 Neal, R. M. (1997). Monte carlo implementation of gaussian process models for bayesian regression
669 and classification. *arXiv preprint physics/9701026*.
670 Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning (Adap-*
671 *tive Computation and Machine Learning*). The MIT Press.
672 Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine
673 learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959.
674 Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view
675 of the evidence of two samples. *Biometrika*, pages 285–294.

677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701