
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 is based on 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. GPs have been part of a recent system for in-

054 ductive learning of symbolic expressions called the Automated Statistician Duvenaud et al. (2013);
 055 Lloyd et al. (2014). Learning such expressions is a hard problem that requires careful design of
 056 approximation techniques since standard inference method do not apply. In the following, we will
 057 present GPs as a novel feature for probabilistic programming languages that solves such problems.
 058 Our contribution is threefold: (i) we introduce a new stochastic process for GPs in a probabilistic
 059 programming language; (ii) we show how one can solve hard problems of state-of-the-art machine
 060 learning related to GP with only a few lines of Venture code; and (iii) we introduce an additional
 061 stochastic process that samples from a probabilistic context free grammar for GP covariance struc-
 062 ture generation.

063 We evaluate the contribution on hard problems posed by the GP community using real world and
 064 synthetic data by assessing quality in terms of posterior distributions of symbolic outcome and in
 065 terms of the residuals produced by the model. The paper is structured as follows, we will first
 066 provide some background on probabilistic programming in Venture and GPs. We will then elaborate
 067 on our new stochastic processes. Finally, we will show how we can apply those on problems of
 068 hyper-parameter inference, structure discovery for Gaussian Processes and Bayesian Optimization
 069 including experiments with real world and synthetic data.

071 2 Background

072 2.1 Venture

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

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

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

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

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

091 Venture implements an MH transition operator for probabilistic execution traces.

092 2.2 Gaussian Processes

093 In the following, we will introduce GP related theory and notations. We will exclusively work on
 094 two variable regression problems. Let the data be real-valued scalars $\{x_i, y_i\}_{i=1}^n$ (complete data will
 095 be denoted by column vectors \mathbf{x}, \mathbf{y}). GPs present a non-parametric way to express prior knowledge
 096 on the space of possible functions f that we assume to have generated the data. f is assumed latent
 097 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
 098 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
 099 function that summarizes the covariance of all functions that map to y_i at x_i . We can absorb the
 100 mean function into the covariance function so without loss of generality we can set the mean to

108 zero. The marginal likelihood can be expressed as:
109

$$110 \quad p(\mathbf{y}|\mathbf{x}) = \int p(\mathbf{y}|\mathbf{f}, \mathbf{x}) p(\mathbf{f}|\mathbf{x}) d\mathbf{f} \quad (3)$$

112 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
113 predictive posterior with

$$114 \quad \mathbf{y}^* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad (4)$$

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

$$116 \quad \boldsymbol{\mu} = \mathbf{K}(\mathbf{x}, \mathbf{x}^*) \mathbf{K}(\mathbf{x}^*, \mathbf{x}^*)^{-1} \mathbf{y} \quad (5)$$

118 and covariance

$$119 \quad \boldsymbol{\Sigma} = \mathbf{K}(\mathbf{x}, \mathbf{x}) + \mathbf{K}(\mathbf{x}, \mathbf{x}^*) \mathbf{K}(\mathbf{x}^*, \mathbf{x}^*)^{-1} \mathbf{K}(\mathbf{x}^*, \mathbf{x}). \quad (6)$$

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

$$121 \quad \log P(\mathbf{y} | \mathbf{X}) = -\frac{1}{2} \mathbf{y}^\top (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{y} - \frac{1}{2} \log |\mathbf{K} + \sigma^2 \mathbf{I}| - \frac{n}{2} \log 2\pi \quad (7)$$

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

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

$$131 \quad k(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right) \quad (8)$$

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

136 3 Venture GPs

138 Given a stochastic process that implements the GP algebra above we can imple-
139 ment a GP sampler (4) to perform GP inference in a few lines of code. We
140 can express simple GP smoothing with fixed hyper-parameters or a prior on hyper-
141 parameters and perform MH on it while allowing users to custom design covari-
142 ance functions. Throughout the paper, we will use the Scheme-like front-end syntax.

```
1 [ASSUME l (gamma 1 3)] ∈ {hyper-parameters}
2 [ASSUME sf (gamma 1 3)] ∈ {hyper-parameters}
3
4 k(x, x') := σ² exp(-((x - x')² / (2ℓ²)))
5
6 [ASSUME f VentureFunction(k, σ, ℓ) ]
7 [ASSUME SE make-se (apply-function f l sf) ]
8 [ASSUME (make-gp 0 SE) ]
9
10 [SAMPLE GP (array 1 2 3)] % Prior
11 [OBSERVE GP D]
12 [SAMPLE GP (array 1 2 3)]
13 [INFER (MH {hyper-parameters} one 100) ]
14 [SAMPLE GP (array 1 2 3)] % Posterior
15
```

156 Listing 1: Bayesian GP Smoothing
157

158 The first two lines depict the hyper-parameters. We tag both of them to belong to the set {hyper-
159 parameters}. Every member of this set belongs to the same inference scope. This scope controls the
160 application of the inference procedure used. In this paper, we use MH throughout. Each scope is
161 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.

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

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

172 We have already implemented this in listing 1. We draw the hyper-parameters from a Γ -prior for
 173 a Bayesian treatment of hyper-parameters. This is simple using the build in stochastic procedure
 174 that simulates drawing samples from a gamma distribution. The program gives rise to a Bayesian
 175 representation of GPs, which we will explore in the following.

178 3.1 A Bayesian interpretation

181 3.1.1 GP modelling as a special case of `gpmem`

183 From the standpoint of computation, a data set of the form $\{(x_i, y_i)\}$ can be thought of as a function
 184 $y = f_{\text{restr}}(x)$, where f_{restr} is restricted to only allow evaluation at a specific set of inputs x . Modelling
 185 the data set with a GP then amounts to trying to learn a smooth function f_{emu} (“emu” stands for
 186 “emulator”) which extends f to its full domain. Indeed, if f_{restr} is defined as a foreign procedure
 187 made available as a black-box to Venture:

```

188
189 def f_restr(x):
190     if x in D:
191         return D[x]
192     else:
193         raise Exception('Illegal input')
194
  
```

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

```

197
198 [ASSUME (list f_compute f_emu) (gpmem f_restr)]
199 for i=1 to n:
200   [PREDICT (f_compute x[i])]
201   [INFER (MH {hyper-parameters} one 100)]
202   [SAMPLE (f_emu (array 1 2 3))]
203
  
```

204 This rewriting has at least two benefits: (i) readability (in some cases), and (ii) amenability to active
 205 learning. As to (i), the statistical code of creating a Gaussian process is replaced with a memoization-
 206 like idiom, which will be more familiar to programmers. As to (ii), when using `gpmem`, it is quite
 207 easy to decide incrementally which data point to sample next: for example, the loop from $x[1]$ to
 208 $x[n]$ could be replaced by a loop in which the next index i is chosen by a supplied decision rule.
 209 In this way, we could use `gpmem` to perform online learning using only a subset of the available
 210 data.

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

216 **3.1.2 The efficacy of learning hyperparameters**
 217

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

$$220 \quad P(\boldsymbol{\theta} | \mathbf{D}, \mathbf{K}) = \frac{P(\mathbf{D} | \boldsymbol{\theta}, \mathbf{K})P(\boldsymbol{\theta} | \mathbf{K})}{P(\mathbf{D} | \mathbf{K})}. \quad (9)$$

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

$$226 \quad \ell^{(t)} \sim \Gamma(\alpha_1, \beta_1), \sigma^{(t)} \sim \Gamma(\alpha_2, \beta_2) \quad (10)$$

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

$$229 \quad \alpha_1^{(t)} \sim \Gamma(\alpha_\alpha^1, \beta_\alpha^1), \alpha_2^{(t)} \sim \Gamma(\alpha_\alpha^2, \beta_\alpha^2), \dots \quad (11)$$

230 We can represent this kind of model using gpmem with only a few lines of code
ToDo: Turn this into gpmem and change cov structure so that it accounts for
 231 **WN kernel, move the background on kernel composition from structure learning**
 232 **to background so that one can understand SE + WN in the case below:**
 233

```

1 [ASSUME alpha (mem (lambda (i) (gamma 1 3)))] ∈ {hyper-parameters-Γ}
2 [ASSUME beta (mem (lambda (i) (gamma 1 3)))] ∈ {hyper-parameters-Γ}
3
4
5 [ASSUME l (gamma (alpha 1) (beta 1))] ∈ {hyper-parameters}
6 [ASSUME sf (gamma (alpha 2) (beta 2))] ∈ {hyper-parameters}
7
8 k(x, x') := σ² exp(-\frac{(x-x')²}{2ℓ²})
9
10 [ASSUME f VentureFunction(k, σ, ℓ) ]
11 [ASSUME SE make-se (apply-function f l sf) ]
12 [ASSUME (make-gp 0 SE) ]
13
14
15 [SAMPLE GP (array 1 2 3)] % Prior
16 [OBSERVE GP D]
17 [SAMPLE GP (array 1 2 3)]
18 [INFER (REPEAT 100
19   (DO (MH {hyper-parameters} one 2)
20     (MH {hyper-parameters-Γ} one 2) ))]
21 [SAMPLE GP (array 1 2 3)] % Posterior
22
23
```

253 Listing 2: Bayesian GP Smoothing

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

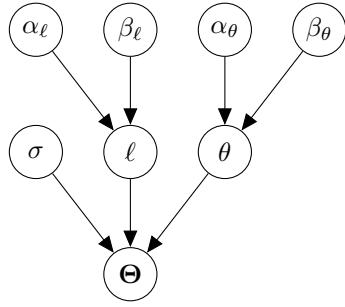
$$256 \quad f(x) = 0.3 + 0.4x + 0.5 \sin(2.7x) + \frac{1.1}{(1+x^2)} + \eta \quad \text{with } \eta \sim \mathcal{N}(0, \sigma) \quad (12)$$

257 We synthetically generate outliers by setting $\sigma = 0.1$ in 95% of the cases and to $\sigma = 1$ in the
 258 remaining cases. gpmem can capture the true underlying function within only 100 MH steps on the
 259 hyper-parameters to get a good approximation for their posterior (see Fig. 1). Note that Neal devices
 260 an additional noise model and performs large number of Hybrid-Monte Carlo and Gibbs steps. We
 261 illustrate the hyper-parameter by showing the shift of the distribution on the noise parameter σ (Fig.
 262 2). We see that gpmem learns the posterior distribution well, the posterior even exhibits a bimodal
 263 histogram when sampling σ 100 times reflecting the two modes of data generation, that is normal
 264 noise and outliers².

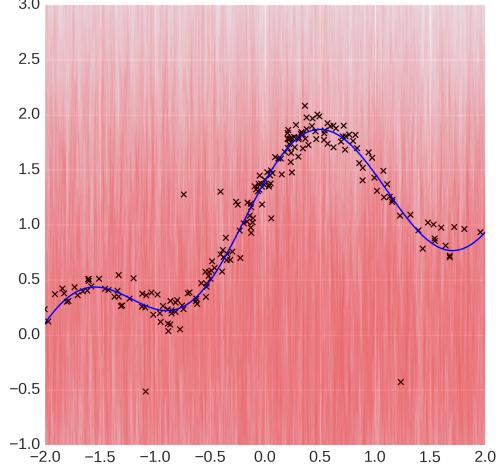
265 ¹In (Neal, 1997) the sum of an SE plus a constant kernel is used. We stick to the WN kernel for illustrative
 266 purposes.

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

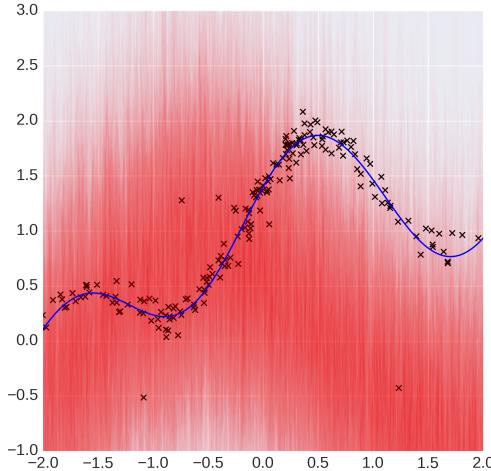
270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 290
 291
 292



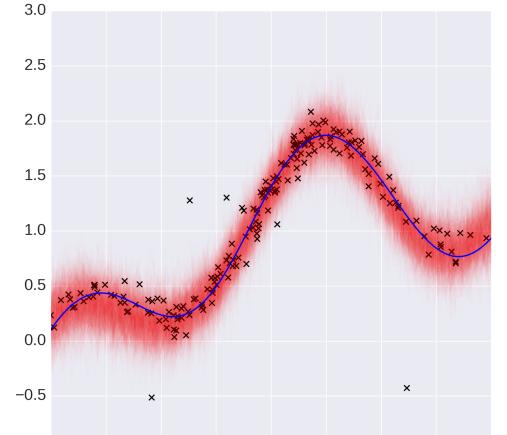
(a) Hierarchical Prior



(b) Prior Inference



(c) Observed



(d) Inferred

311
 312
 313
 314
 315
 316
 317
 318
 319
 320
 321
 322
 323

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

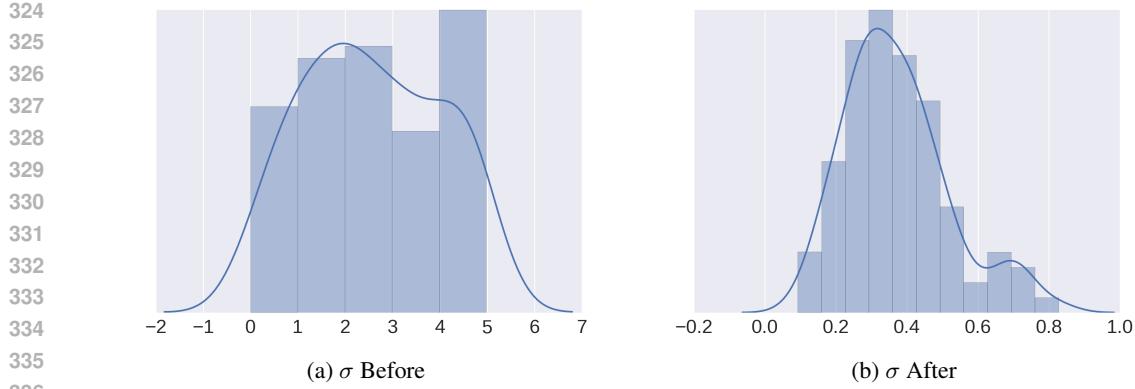


Figure 2: Hyper-parameter inference on the parameter of the noise kernel. We show 100 samples drawn from the distribution on σ . One can clearly recognise the shift from the uniform prior $\mathcal{U}(0, 5)$ to a double peak distribution around the two modes - normal and outlier.

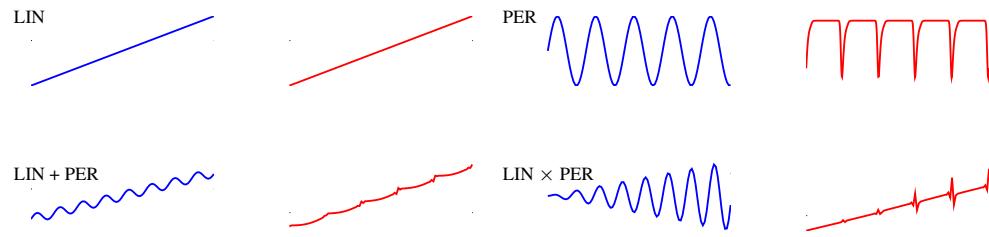


Figure 3: Composition of covariance functions (blue, left) and samples from the distribution of curves they can produce (red, right).

3.2 Structure Learning

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

$$k_3(x, x') = k_1(x, x') + k_2(x, x') \quad (13)$$

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

$$k_4(x, x') = k_1(x, x') \times k_2(x, x'); \quad (14)$$

both, k_3 and k_4 are valid covariance function structures. This leads to an infinite space of possible structures that could potentially explain the observed data best (e.g. Fig. 3). 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  $k_{LIN}(x, x') := \sigma_{12}^2 (x - \ell) \frac{(x' - \ell)}{\sqrt{2\pi^2/\sigma_{12}^2}}$ 
6  $k_{PER}(x, x') := \sigma_2^2 \exp(-\frac{(x - x')^2}{\ell^2})$ 
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 \times 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 Statisti-

cian 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:

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

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

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

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:

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

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

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

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

with

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

and

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

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

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 expressions using the following grammar:

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}

Listing 4: Grammar to simplify expressions

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

```

432
433 [ASSUME S (array K1,K2,...,Kn) ] // (defined as above)
434 [ASSUME pn (uniform_structure n) ]
435 [ASSUME S (array K1,K2,...,Kn) ]
436 [ASSUME K* (grammar S pn) ]
437 [ASSUME GP (make-gp 0 K*) ]
438 [OBSERVE GP D]
439
440 [INFER (REPEAT 2000 (DO
441     (MH 10 pn one 1)
442     (MH 10 K* one 1)
443     (MH 10 {hyper-parameters} one 10)) ]

```

Listing 5: Venture Code for Bayesian GP Structure Learning

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

We further investigated the quality of our stochastic processes by running a leave one out cross-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 produced 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 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

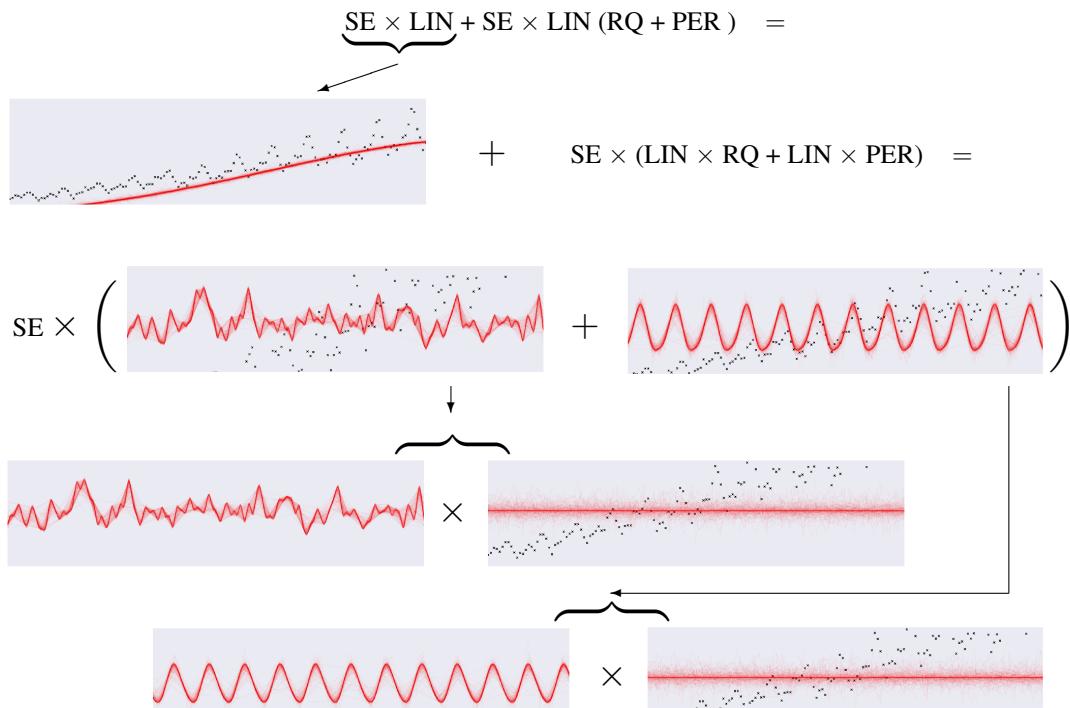
We have shown Venture GPs. We have introduced novel stochastic processes for a probabilistic programming language. We showed how flexible non-parametric models can be treated in Venture in only a few lines of code. We evaluated our contribution on a range of hard problems for state-of-the-art Bayesian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

486
487
488
489
490
491
492
493
494
495
496
497
498
499



(a) The predictive posterior using the full grammar structure.

500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526



(b) Compositional Structure

527
528
529
530
531
532
533
534
535
536
537
538
539

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

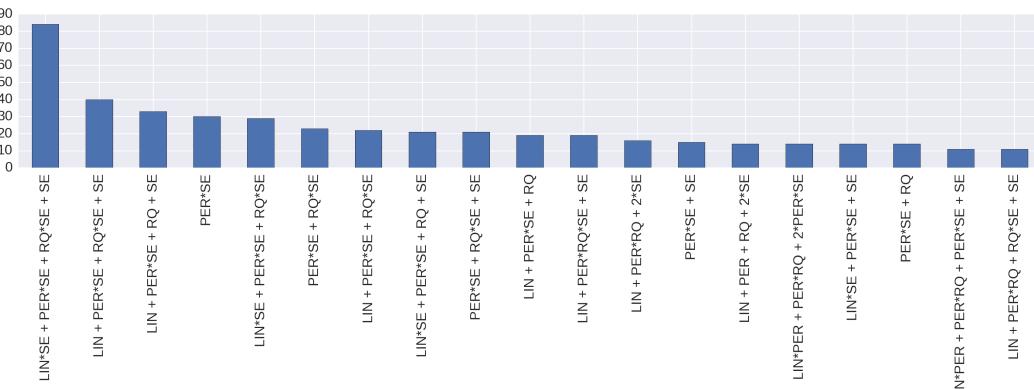


Figure 5: Posterior on structure of the CO₂ data. We have cut the tail of the distribution for space reasons since the number of possible structures is large. We see the final sample of the each of the 545 chains with 2000 nested steps each. Note that Duvenaud et al. (2013) report LIN × SE + PER × SE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

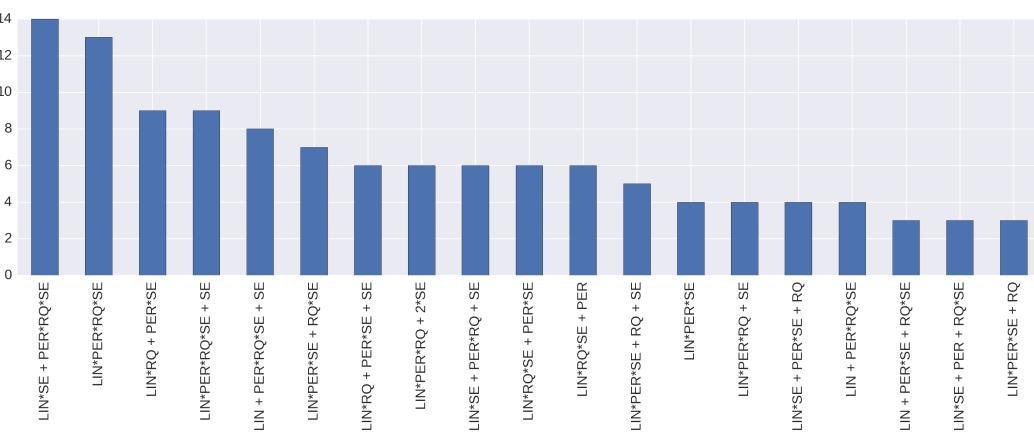


Figure 6: Posterior on structure of airline data set. We have cut the tail of the distribution for space reasons since the number of possible structures is large. We see the final sample of the each of the 144 chains with 2000 nested steps each. Note that Duvenaud et al. (2013) report $\text{LIN} \times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

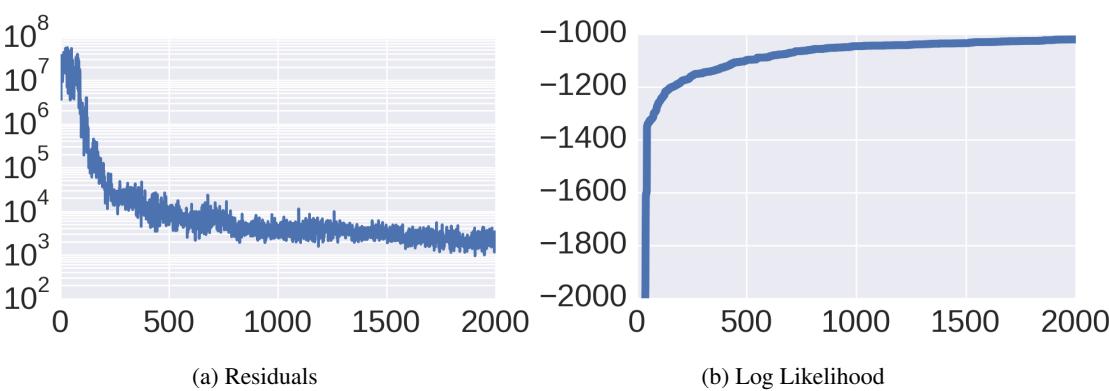


Figure 7: 2000 steps along the Markov Chain.

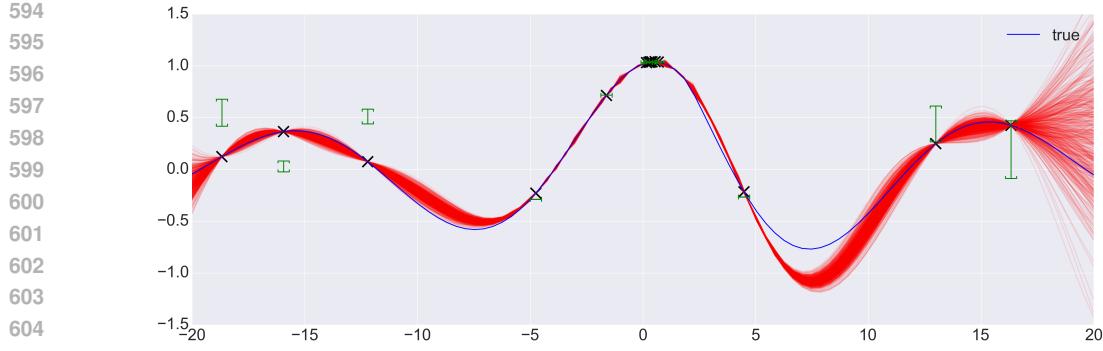


Figure 8: Bayesian Optimization. Each successive probe point x is the (stochastic) maximum of a GP-based emulator conditioned on the values of the previously probed points. In the figure, each probe point x is marked with an \times , and a vertical green bar is drawn showing the mean \pm one standard 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.

References

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