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

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

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

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

2 Background

2.1 Memoization

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

2.2 Venture

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

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

$$x^* \sim q(x^* \mid x^{(t)}) \tag{1}$$

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

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

Venture implements an MH transition operator for probabilistic execution traces.

2.3 Gaussian Processes

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

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

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

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

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

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

and covariance

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

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

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

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

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

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

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

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

$$k_3(x, x') = k_1(x, x') + k_2(x, x') \tag{9}$$

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

$$k_4(x, x') = k_1(x, x') \times k_2(x, x'); \tag{10}$$

both, k_3 and k_4 are valid covariance function structures.

3 Venture GPs

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

```
[ASSUME l  (gamma 1 3)] ∈ {hyper-parameters}
[ASSUME sf (gamma 1 3)] ∈ {hyper-parameters}
```

$$k(x, x') := \sigma^2 \exp(-\frac{(x-x')^2}{2\ell^2})$$

```
[ASSUME f VentureFunction(k,σ,ℓ) ]
[ASSUME SE make-se (apply-function f l sf) ]
[ASSUME (make-gp 0 SE) ]

[SAMPLE GP (array 1 2 3)] % Prior
[OBSERVE GP D]
[SAMPLE GP (array 1 2 3)]
[INFER  (MH {hyper-parameters} one 100) ]
[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:

```
def f_restr(x):
  if x in D:
    return D[x]
  else:
    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]`,...,`x[n]`):

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

```
  [PREDICT (f_compute x[i])]
[INFER (MH {hyper-parameters} one 100)]
[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.

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.First, we will show how one can utilize `gpmem` for reproducing state-of-the-art models that are based on GP.

3.1.2 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:

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

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)[1]. The work suggests a hierarchical system of hyper-parameterization (Fig. 1a). Here, we draw hyper-parameters from a Γ distributions:

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

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

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

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 WN kernel, move the background on kernel composition from structure learn-**

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

ing to background so that one can understand SE + WN in the case below:

```
[ASSUME alpha (mem (lambda (i)  (gamma  1 3 )))] ∈ {hyper-parameters-Γ}
[ASSUME beta  (mem (lambda (i)  (gamma  1 3 )))] ∈ {hyper-parameters-Γ}


[ASSUME l  (gamma (alpha 1) (beta 1))] ∈ {hyper-parameters}
[ASSUME sf (gamma (alpha 2) (beta 2))] ∈ {hyper-parameters}

k(x,x') := σ^2 exp(−(x−x')^2/(2ℓ^2))
k(x,x') := σ^2 exp(−(x−x')^2/(2ℓ^2))
[ASSUME f VentureFunction(k,σ,ℓ) ]
[ASSUME SE make-se (apply-function f l sf) ]
[ASSUME (make-gp 0 SE) ]

[SAMPLE GP (array 1 2 3)] % Prior
[OBSERVE GP D]
[SAMPLE GP (array 1 2 3)]
[INFER (REPEAT 100
  (DO (MH {hyper-parameters} one 2)
      (MH {hyper-parameters-Γ} one 2) ))]
[SAMPLE GP (array 1 2 3)] % Posterior
```

Listing 2: Bayesian 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:

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

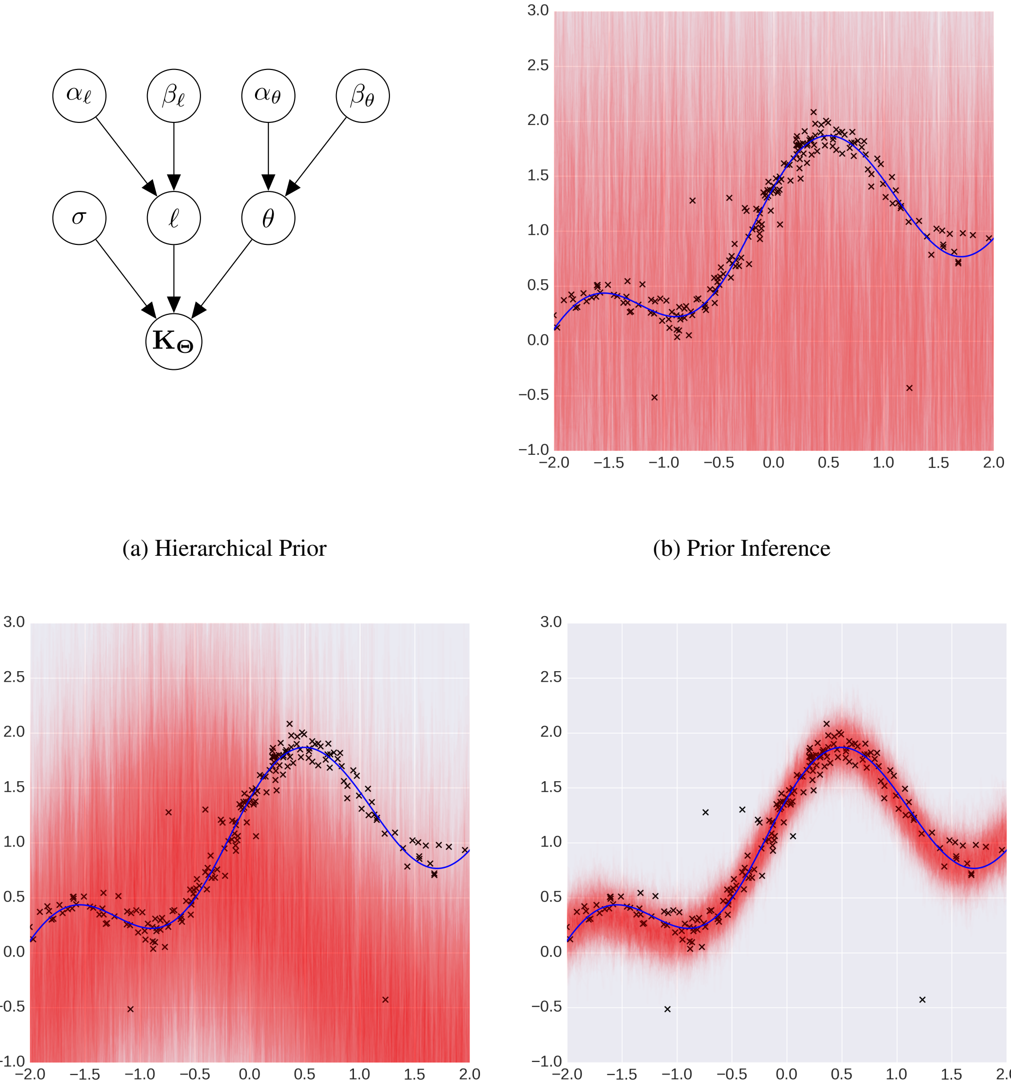
We synthetically generate outliers by setting $\sigma = 0.1$ in 95% of the cases and to $\sigma = 1$ in the remaining cases. `gpmem` can capture the true underlying function within only 100 MH steps on the hyper-parameters to get a good approximation for their posterior (see Fig. 1). Note that Neal devices 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 outliers[2].

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 it's 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).

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:

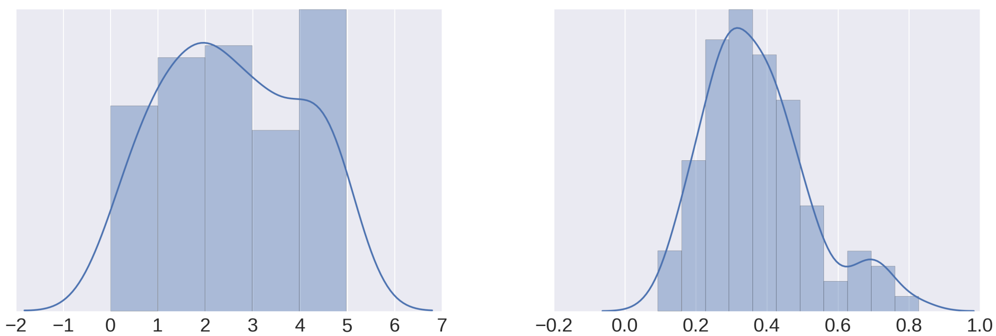
[2]For this pedagogical example we have increased the probability for outliers in the data generation slightly from 0.05 to 0.2



(a) Hierarchical Prior (b) Prior Inference

(c) Observed (d) Inferred

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.



(a) σ Before

(b) σ After

Figure 2: Hyper-parameter inference on the parameter of the noise kernel. We show a 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.

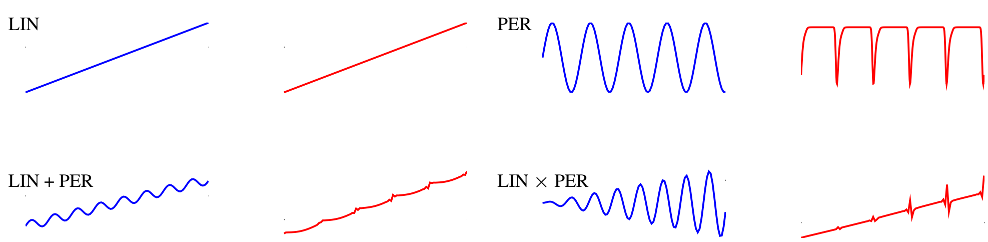


Figure 3: Composition of covariance functions (blue, left) and samples from the distribution of curves they can produce (red, right).

```
[ASSUME l (gamma 1 3)]
[ASSUME sf (gamma 1 2)]
[ASSUME a (gamma 2 2)]

k_{LIN}(x,x') := \sigma_1^2 (x - \ell)(x' - \ell)
k_{PER}(x,x') := \sigma_2^2 \exp(-\frac{2\sin^2(\pi(x-x')/p)}{\ell^2})

[ASSUME f_{LIN} VentureFunction(k_{LIN},\sigma_1) ]
[ASSUME f_{PER} VentureFunction(k_{PER},\sigma_2,\ell,p) ]
[ASSUME LIN (make-LIN (apply-function f_{LIN} a)) ]
[ASSUME PER (make-PER (apply-function f_{PER} l sf)) ]
[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 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:

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

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

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

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:

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

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:

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

with

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

and

$$P(s \mid n) = \frac{n!}{\mid s \mid !} \tag{20}$$

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:

```
SE × SE                   → SE
{SE,PER,C,WN} × WN        → WN
LIN + LIN                 → LIN
{SE,PER,C,WN,LIN} × C     →  {SE,PER,C,WN,LIN}
```

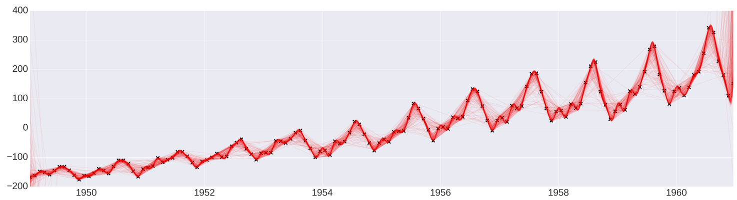
Listing 4: Grammar to simplify expressions

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.

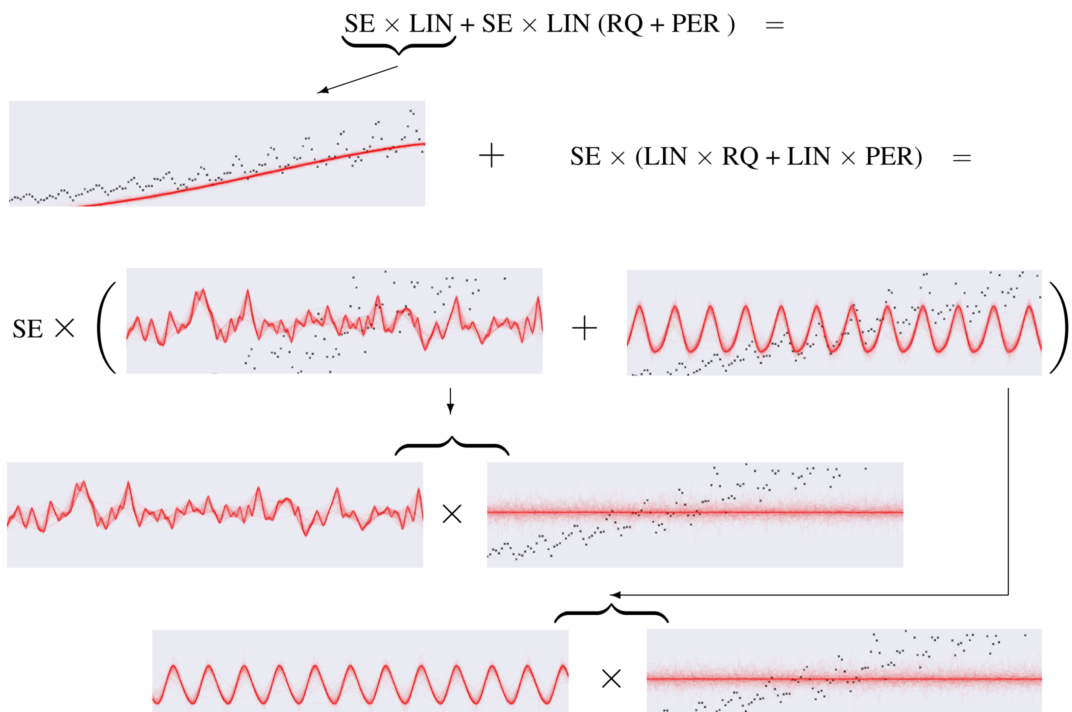
```
[ASSUME S (array K_1,K_2,···,K_n)] // (defined as above)
[ASSUME p_n (uniform_structure n)]
[ASSUME S (array K_1,K_2,···,K_n)]
[ASSUME K* (grammar S p_n)]
[ASSUME GP (make-gp 0 K* ]

[OBSERVE GP D]

[INFER  (REPEAT 2000 (DO
      (MH 10 p_n one 1)
      (MH 10 K* one 1)
      (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 CO_2 data set () and the predictive capability is



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



(b) Compositional Structure

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.

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).

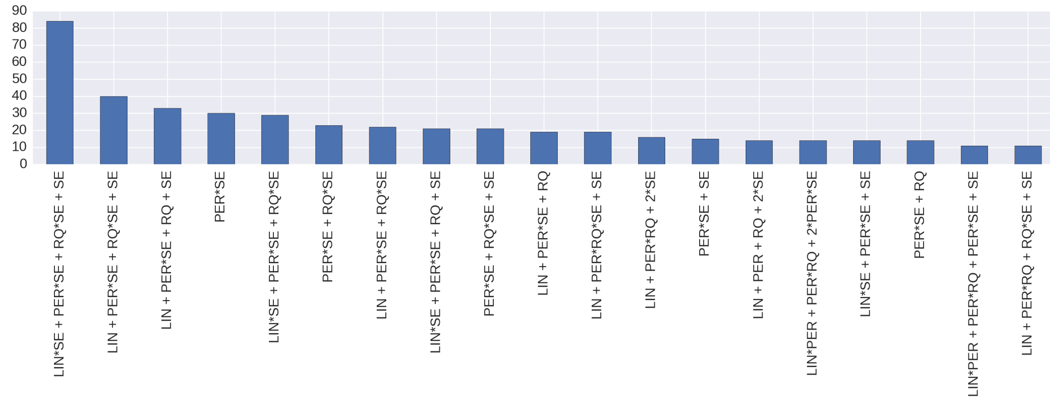


Figure 5: Posterior on structure of the CO2 data. We have cut the tail of the distribution for space reasons since the number of possible structures is large. We see the final sample of the each of the 545 chains with 2000 nested steps each. Note that Duvenaud et al. (2013) report LIN $\times$ SE $+$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

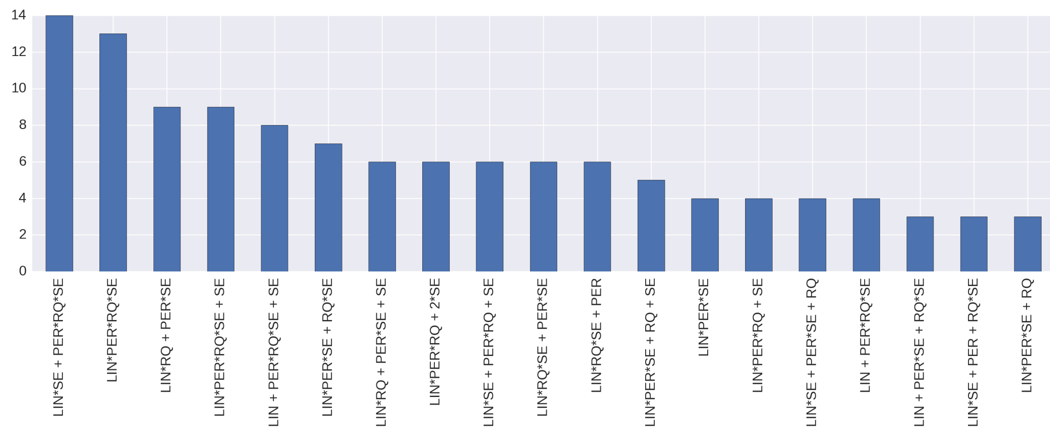


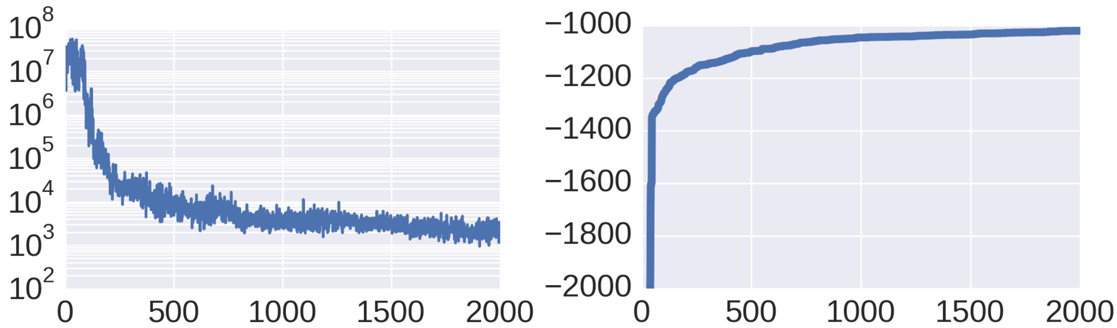
Figure 6: Posterior on structure of airline data set. We have cut the tail of the distribution for space reasons since the number of possible structures is large. We see the final sample of the each of the 144 chains with 2000 nested steps each. Note that Duvenaud et al. (2013) report LIN $\times$ SE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

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



(a) Residuals (b) Log Likelihood

Figure 7: 2000 steps along the Markov Chain.

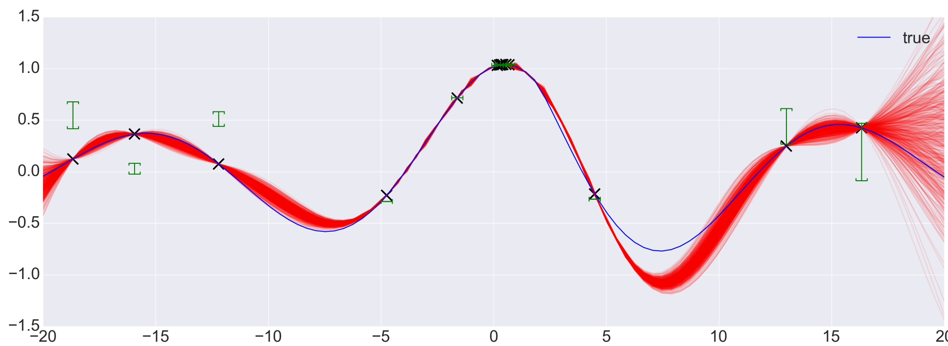


Figure 8: Bayesian Optimization. Each successive probe point x is the (stochastic) maximum of a GP-based emulator conditioned on the values of the previously probed points. In the figure, each probe point x is marked with an $\times$, and a vertical green bar is drawn showing the mean $\pm$ one standard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

(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.

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.

Goodman, N. D .and Mansinghka, V. K., Roy, D., Bonawitz, K., and Tenenbaum, J. (2008). Church: A language for generative models. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence, UAI 2008*, pages 220–229.

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.