
Gaussian Processes with Probabilistic Programming

000
001
002
003
004
005
006
007
008
009
010
011
012
013
014
015
016
017
018
019
020
021
022
023
024
025
026
027
028
029
030
031
032
033
034
035
036
037
038
039
040
041
042
043
044
045
046
047
048
049
050
051
052
053
Anonymous Author(s)

Affiliation
Address
email

Abstract

We introduce Venture GPs, a way to formulate Gaussian Processes with probabilistic programming. Gaussian Processes are flexible non-parametric models that can be applied to a broad class of problems. We represent Gaussian Processes in Venture, a Turing complete probabilistic programming language. Venture provides a compositional language with a generalized inference engine which liberates a user quickly design and test a very broad class of models. The engine builds on a stochastic procedure interface. This stochastic procedure interface specifies and encapsulates primitive random variables analogously to conditional probability tables in Bayesian Networks. The programming language is extended with a set of stochastic processes that allow a user to formulate Gaussian Process models and to perform numerically stable inference over them while hiding the linear algebra needed for this inside the language. We show how we can extend the non-parametric model to incorporate hierarchical causal priors on model structure and hyper-parameters with only a few lines of code. We also show state-of-the-art applications of Gaussian Processes in this framework, namely structure discovery of high-level properties of Gaussian Processes, Bayesian Optimization and hyper-parameter inference. We evaluate the performance of the programs with synthetic and real world data.

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 inductive learning of symbolic expressions called the Automated Statistician Duvenaud et al. (2013); Lloyd et al. (2014). Learning such 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 GPs as a novel feature for probabilistic programming languages that solves such problems. Our contribution is threefold: (i) we introduce a new stochastic process for GPs in a probabilistic programming language; (ii) we show how one can solve hard problems of state-of-the-art machine learning related to GP with only a few lines of Venture code; and (iii) we introduce an additional stochastic process that samples from a probabilistic context free grammar for GP covariance structure generation.

We evaluate the contribution on hard 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 the model. The paper is structured as follows, we will first provide some background on probabilistic programming in Venture and GPs. We will then elaborate on our new stochastic processes. Finally, we will show how we can apply those on problems of hyper-parameter inference, structure discovery for Gaussian Processes and Bayesian Optimization including experiments with real world and synthetic data.

2 Background

2.1 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) (?). 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 :

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

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

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

Venture implements an MH transition operator for probabilistic execution traces.

2.2 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:

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

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 predictive posterior with

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

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

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

and covariance

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

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

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

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:

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

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.

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.

```

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
```

Listing 1: Bayesian GP Smoothing

The first two lines depict the hyper-parameters. We tag both of them to belong to the set $\{\text{hyper-parameters}\}$. 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.

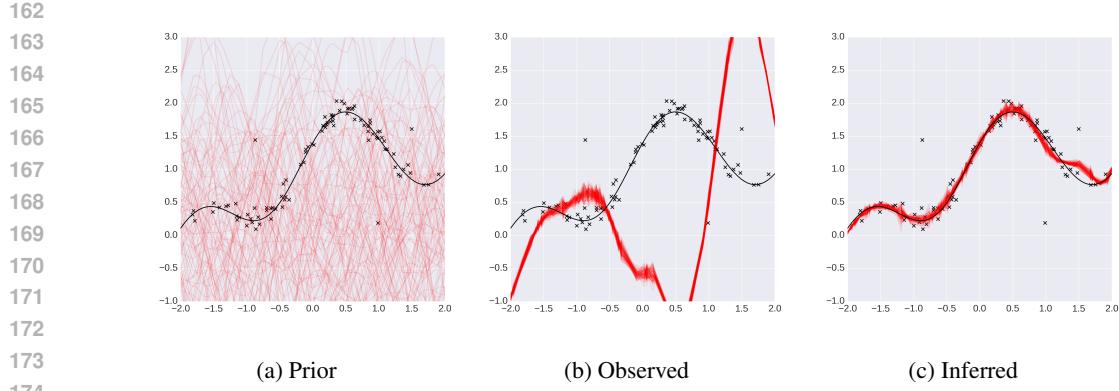


Figure 1: Running a Venture GP on Neal’s example for MCMC showing the prior, after having observed the data and after performing inference on the hyper-parameters. Note how the GP is choosing outliers to smooth instead of essential data before inference takes place.

The ASSUME directives describe the assumptions we make for the GP model, we assume the hyper-parameters ℓ and σ (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) where we draw hyper-parameters from Γ distributions:

$$\ell^{(t)} \sim \Gamma(\alpha_1, \beta_1), \sigma^{(t)} \sim \Gamma(\alpha_2, \beta_2) \quad (9)$$

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 The efficacy of learning hyperparameters

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

$$P(\boldsymbol{\theta} | \mathbf{D}, \mathbf{K}) = \frac{P(\mathbf{D} | \boldsymbol{\theta}, \mathbf{K})P(\boldsymbol{\theta} | \mathbf{K})}{P(\mathbf{D} | \mathbf{K})}. \quad (10)$$

Neal suggested the treatment of outliers as a use-case for a Bayesian treatment of Gaussian processes (1997). He evaluates his MCMC setting using the following synthetic data problem. Let f be the underlying function that generates the data:

$$f(x) = 0.3 + 0.4x + 0.5 \sin(2.7x) + \frac{1.1}{(1+x^2)} + \eta \quad \text{with } \eta \sim \mathcal{N}(0, \sigma) \quad (11)$$

We synthetically generate outliers by setting $\sigma = 0.1$ in 95% of the cases and to $\sigma = 1$ in the remaining cases. Venture GPs 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).

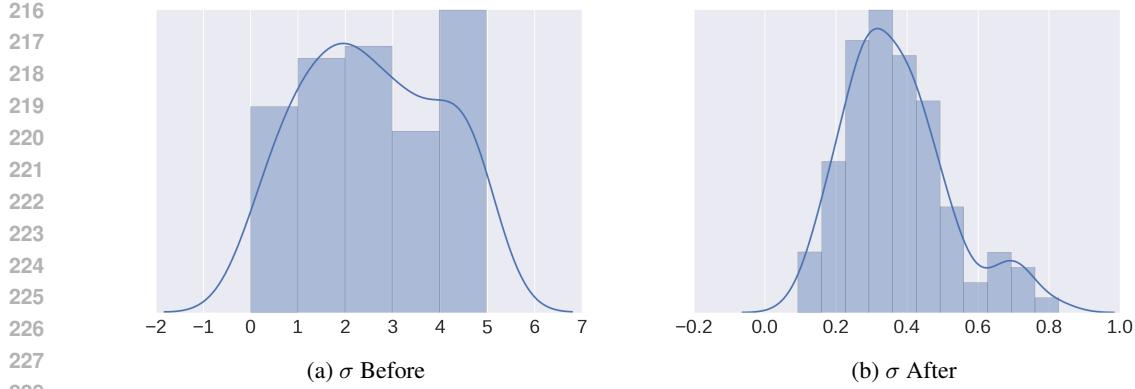


Figure 2: Hyper-parameter inference on the parameter of the noise kernel. We show 100 samples drawn from the distribution on σ . One can clearly recognise the shift from the uniform prior $\mathcal{U}(0, 5)$ to a double peak distribution around the two modes - normal and outlier.

3.1.2 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 $x[1], \dots, x[n]$):

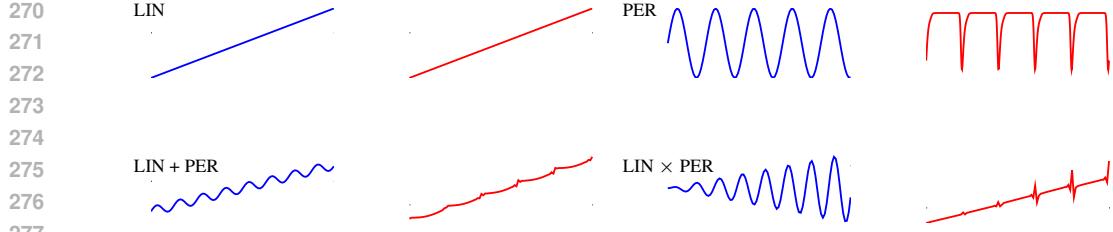
```
[ASSUME (list f_compute f_emu) (gpmem f_restr)]
for i=1 to n:
    [PREDICT (f_compute x[i])]
    [INFER (MH {hyper-parameters} one 100)]
    [SAMPLE (f_emu (array 1 2 3))]
```

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.

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



270
271
272
273
274
275
276
277
278
279
Figure 3: Composition of covariance functions (blue, left) and samples from the distribution of
curves they can produce (red, right).

280

281

model globally valid structures

283

$$k_3(x, x') = k_1(x, x') + k_2(x, x') \quad (12)$$

284

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

286

287

$$k_4(x, x') = k_1(x, x') \times k_2(x, x'); \quad (13)$$

288

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:

293

```

1 [ASSUME l (gamma 1 3)]
2 [ASSUME sf (gamma 1 2)]
3 [ASSUME a (gamma 2 2)]
4
5 kLIN(x, x') := σ122(x - ℓ)(x' - ℓ)
6 kPER(x, x') := σ222 exp(-2sin2(π(x - x')/p))
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)) ]
```

302

303

Listing 2: LIN × PER

304

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 (Duvénac 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:

311

312

$$\mathbb{E}[y^* | x^*, \mathbf{D}, \mathbf{K}] = \iint f(x^*, \boldsymbol{\theta}, \mathbf{K}) P(\boldsymbol{\theta} | \mathbf{D}, \mathbf{K}) P(\mathbf{K} | \Omega, s, n) d\boldsymbol{\theta} d\mathbf{K}. \quad (14)$$

313

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

316

317

$$y^* \approx \frac{1}{T} \sum_{t=1}^T f(x^* | \boldsymbol{\theta}^{(t)}, \mathbf{K}^{(t)}). \quad (15)$$

318

319

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:

321

322

$$\mathbf{K}^{(t)} \sim P(\mathbf{K} | \Omega, s, n) \quad (16)$$

323

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.

324 The marginal probability of a kernel structure which allows us to sample is characterized by the
 325 probability of a uniformly chosen subset of the set of n possible covariance functions times the
 326 probability of sampling a global or a local structure which is given by a binomial distribution:
 327

$$P(\mathbf{K} | \boldsymbol{\Omega}, s, n) = P(\boldsymbol{\Omega} | s, n) \times P(s | n) \times P(n), \quad (17)$$

328 with
 329

$$P(\boldsymbol{\Omega} | s, n) = \binom{n}{r} p_{+ \times}^k (1 - p_{+ \times})^{n-k} \quad (18)$$

330 and
 331

$$P(s | n) = \frac{n!}{|s|!} \quad (19)$$

332 where $P(n)$ is a prior on the number of base kernels used which can sample from a discrete uniform
 333 distribution. This will strongly prefer simple covariance structures with few base kernels since
 334 individual base kernels are more likely to be sampled in this case due to (19). Alternatively, we
 335 can approximate a uniform prior over structures by weighting $P(n)$ towards higher numbers. It is
 336 possible to also assign a prior for the probability to sample global or local structures, however, we
 337 have assigned complete uncertainty to this with the probability of a flip $p = 0.5$.
 338

339 Many equivalent covariance structures can be sampled due to covariance function algebra
 340 and equivalent representations with different parameterization (Lloyd et al., 2014). Certain
 341 covariance functions can differ in terms of the hyper-parameterization but can be
 342 absorbed into a single covariance function with a different parameterization. To inspect
 343 the posterior of these equivalent structures we convert each kernel expression into
 344 a sum of products and subsequently simplify expressions using the following grammar:
 345

```
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}
```

350 Listing 3: Grammar to simplify expressions
 351

352 For reproducing results from the Automated Statistician Project in a Bayesian fashion we first define
 353 a prior on the hypothesis space. Note that, as in the implementation of the Automated Statistician,
 354 we upper-bound the complexity of the space of covariance functions we want to explore. We also
 355 put vague priors on hyper-parameters.
 356

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

357 Listing 4: Venture Code for Bayesian GP Structure Learning
 358

359 We defined the space of covariance structures in a way allowing us to reproduce results for covariance
 360 function structure learning as in the Automated Statistician. This lead to coherent results, for
 361 example for the airline data set. We will elaborate the result using a sample from the posterior (Fig.
 362 4). The sample is identical with the highest scoring result reported in previous work using a search-
 363 and-score method (Duvenaud et al., 2013) for the CO₂ data set () and the predictive capability is
 364 comparable. However, the components factor in a different way due to different parameterization of
 365 the individual base kernels.
 366

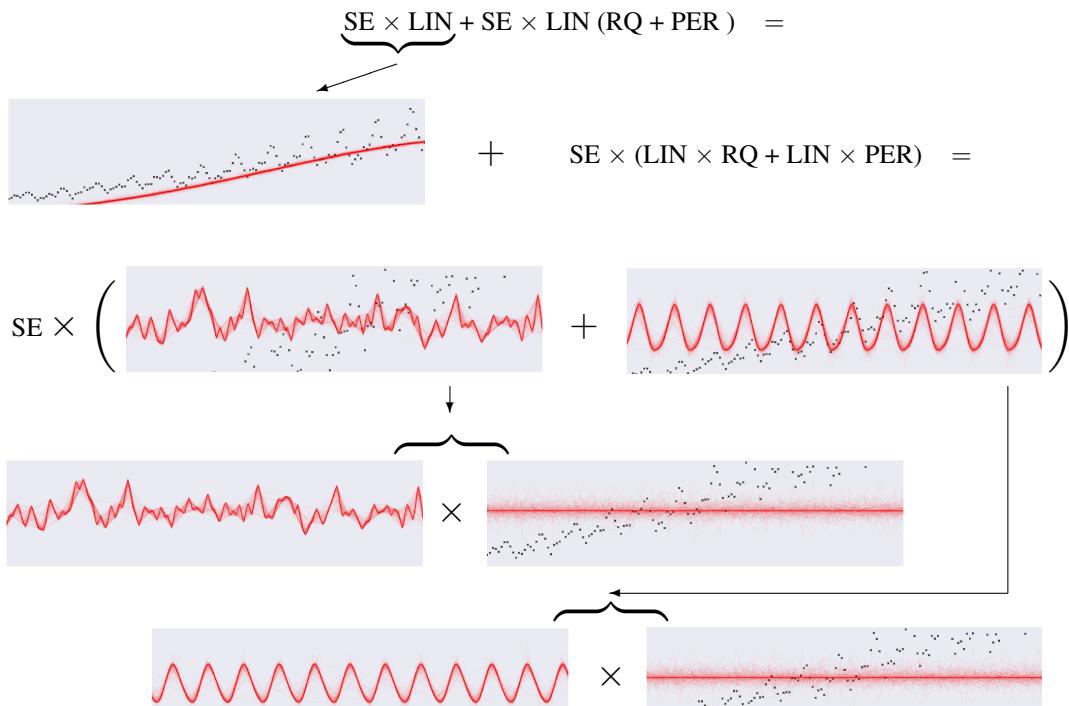
367 We further investigated the quality of our stochastic processes by running a leave one out cross-
 368 validation to gain confidence on the posterior. This resulted in 545 independent runs of the Markov
 369 chain that produced a coherent posterior: our Bayesian interpretation of GP structure and GPs pro-
 370

378
379
380
381
382
383
384
385
386
387
388
389
390
391



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

392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418



(b) Compositional Structure

419
420
421
422
423
424
425
426
427
428
429
430
431

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.

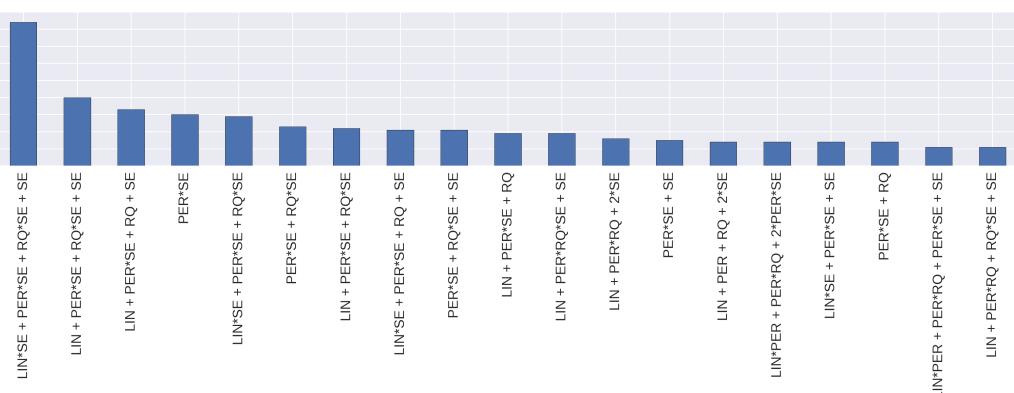


Figure 5: Posterior on structure of the CO2 data. We have cut the tail of the distribution for space reasons since the number of possible structures is large. We see the final sample of the each of the 545 chains with 2000 nested steps each. Note that Duvenaud et al. (2013) report $\text{LIN} \times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

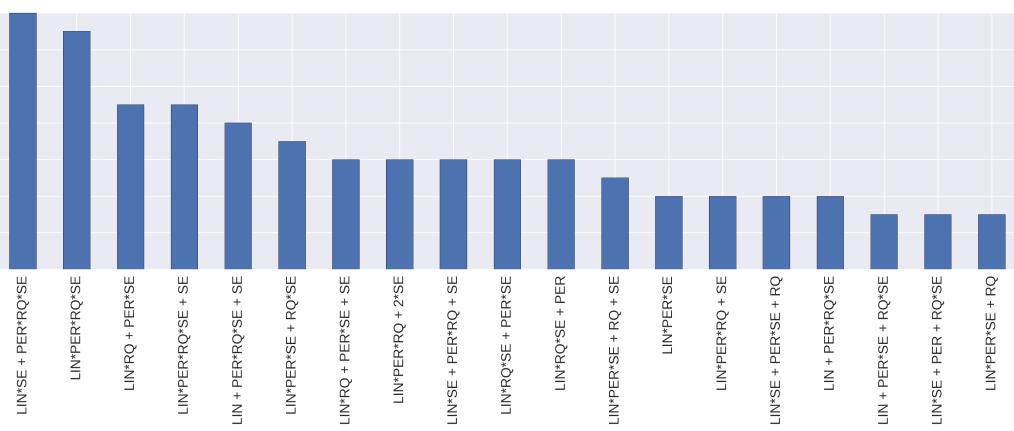


Figure 6: Posterior on structure of airline data set. We have cut the tail of the distribution for space reasons since the number of possible structures is large. We see the final sample of the each of the 144 chains with 2000 nested steps each. Note that Duvenaud et al. (2013) report $\text{LIN} \times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

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

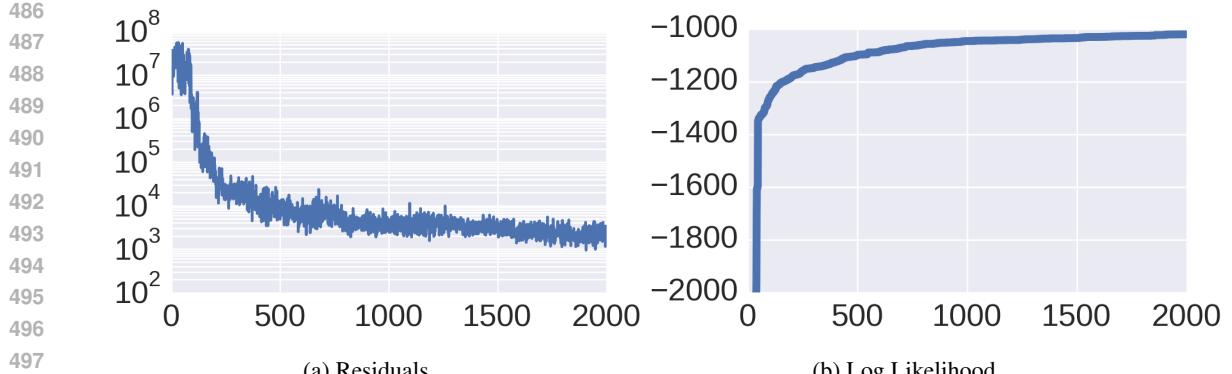


Figure 7: 2000 steps along the Markov Chain.

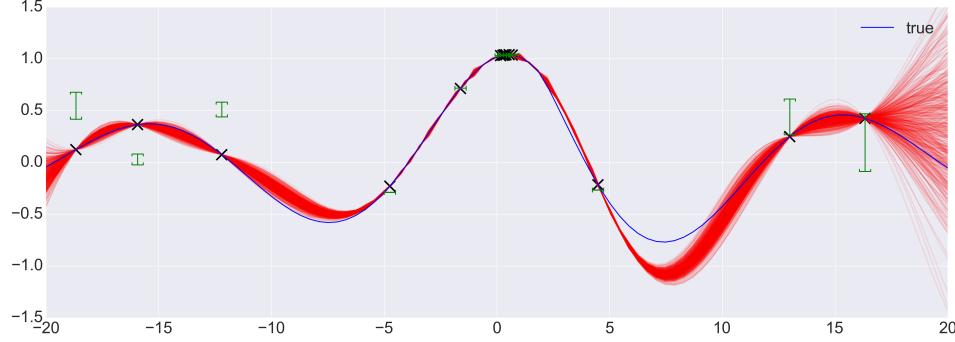


Figure 8: Bayesian Optimization. Each successive probe point x is the (stochastic) maximum of a GP-based emulator conditioned on the values of the previously probed points. In the figure, each probe point x is marked with an \times , and a vertical green bar is drawn showing the mean \pm one standard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

540 **References**
541

- 542 Andrieu, C., De Freitas, N., Doucet, A., and Jordan, M. I. (2003). An introduction to mcmc for
543 machine learning. *Machine learning*, 50(1-2):5–43.
- 544 Duvenaud, D., Lloyd, J. R., Grosse, R., Tenenbaum, J., and Ghahramani, Z. (2013). Structure
545 discovery in nonparametric regression through compositional kernel search. In *Proceedings of*
546 *the 30th International Conference on Machine Learning (ICML-13)*, pages 1166–1174.
- 547 Ghahramani, Z. (2015). Probabilistic machine learning and artificial intelligence. *Nature*,
548 521(7553):452–459.
- 549 Lloyd, J. R., Duvenaud, D., Grosse, R., Tenenbaum, J., and Ghahramani, Z. (2014). Automatic
550 construction and natural-language description of nonparametric regression models. In *Twenty-*
551 *Eighth AAAI Conference on Artificial Intelligence*.
- 552 Mansinghka, V., Selsam, D., and Perov, Y. (2014). Venture: a higher-order probabilistic program-
553 ming platform with programmable inference. *arXiv preprint arXiv:1404.0099*.
- 554 Neal, R. M. (1997). Monte carlo implementation of gaussian process models for bayesian regression
555 and classification. *arXiv preprint physics/9701026*.
- 556 Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning (Adap-*
557 *tive Computation and Machine Learning*). The MIT Press.
- 558 Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine
559 learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959.
- 560 Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view
561 of the evidence of two samples. *Biometrika*, pages 285–294.
- 562
- 563
- 564
- 565
- 566
- 567
- 568
- 569
- 570
- 571
- 572
- 573
- 574
- 575
- 576
- 577
- 578
- 579
- 580
- 581
- 582
- 583
- 584
- 585
- 586
- 587
- 588
- 589
- 590
- 591
- 592
- 593