Gaussian Processes with Probabilistic Programming

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 that builds on a stochastic procedure interface. This stochastic procedure interface specifies and encapsulates primitive random variables analogously 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). 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 and 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 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. In the following, we will present Gaussian Processes as a novel feature for probabilistic programming languages. Our contribution is threefold:

- we introduce a new stochastic process for GPs in a probabilistic programming language;

- we show how one can solve hard problems of state-of-the-art machine learning with only a few lines of Venture code; and
- we introduce an additional stochastic process that samples from a probabilistic context free grammar for GP covariance structure generation.

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. It's implementation is based on on three concepts:

1. stochastic procedure interfaces that specify and encapsulate random variables, analogously to conditional probability tables in a Bayesian network;
2. probabilistic execution traces that represent execution histories and capture conditional dependencies; and
3. scaffolds that partition execution histories and factor global inference problems into subproblems.

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 to 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.

2.2 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:

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

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.

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 and perform MH on it code while allowing users to custom design covariance functions. Throughout the paper, we will use the Scheme-like front-end syntax.

Listing 1: GP Smoothing

```
[ASSUME l 1] ∈ {hyper−parameters}
[ASSUME sf 2] ∈ {hyper−parameters}

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 (MH {hyper−parameters} one 100) ]

[SAMPLE GP (array 1 2 3)] % Posterior
```

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) where we draw hyper-parameters from Γ distributions:

$$\ell^{(t)} \sim \Gamma(\alpha_1, \beta_1) \tag{9}$$

$$\sigma^{(t)} \sim \Gamma(\alpha_2, \beta_2) \tag{10}$$

Extending the program described in listing 1 to draw the hyper-parameters for a Bayesian treatment of hyper-parameters is simple using the build in stochastic procedure that simulates drawing samples from a gamma distribution:

Listing 2: Bayesian GP Smoothing

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

Larger variations are achieved by changing the type of the covariance function structure. A different type could be a linear covariance function:

$$k(x, x') = \sigma^2 (x - \ell)(x' - \ell). \tag{11}$$

Note that covariance function structures are compositional. We can add covariance functions if we want to model globally valid structures

$$k_3(x, x') = k_1(x, x') + k_2(x, x') \tag{12}$$

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

$$k_4(x, x') = k_1(x, x') \times k_2(x, x'); \tag{13}$$

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. 1). 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:

Listing 3: LIN × PER

```
[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,σ1) ]
[ASSUME f_PER VentureFunction(k_PER,σ2,ℓ,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)) ]
```

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.

3.1 A Bayesian interpretation

In the following, we will explore a Bayesian representation of GP. The probability of the hyper-parameters of a GP with assumptions as above and given covariance function structure $\mathbf{K}$ can be described as:

$$P(\boldsymbol{\theta} \mid \mathbf{D}, \mathbf{K}) = \frac{P(\mathbf{D} \mid \boldsymbol{\theta}, \mathbf{K}) P(\boldsymbol{\theta} \mid \mathbf{K})}{P(\mathbf{D} \mid \mathbf{K})}. \tag{14}$$

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:

$$f(x) = 0.3 + 0.4x + 0.5\sin(2.7x) + \frac{1.1}{(1+x^2)} + \eta \quad with \;\; \eta \sim \mathcal{N}(0,\sigma) \tag{15}$$

We synthetically generate outliers by setting $\sigma = 0.1$ in 95% of the case and to $\sigma = 1$ in the remaining cases. Venture GPs can capture the true underlying function within only 100 MH steps (see Fig. 2). Note that Neal devices an additional noise model and performs large numbe of Hybrid-Monte Carlo and Gibbs steps.

3.2 Structure Learning

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:

$$\mathbb{E}[y^* \mid x^*, D, \mathbf{K}_\Omega^s] = \iint f(x^*, \boldsymbol{\theta}, \mathbf{K})\, P(\boldsymbol{\theta} \mid \mathbf{D}, \mathbf{K})\, P(\mathbf{K} | \boldsymbol{\Omega}, s, n)\, \mathbf{d}\boldsymbol{\theta}\mathbf{d}\mathbf{K}. \tag{16}$$

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

$$y^* \approx \frac{1}{T}\sum_{t=1}^{T} f(x^* | \boldsymbol{\theta}^{(t)}, \mathbf{K}^{(t)}). \tag{17}$$

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:

$$\mathbf{K}^{(t)} \sim P(\mathbf{K} \mid \boldsymbol{\Omega}, s, n) \tag{18}$$

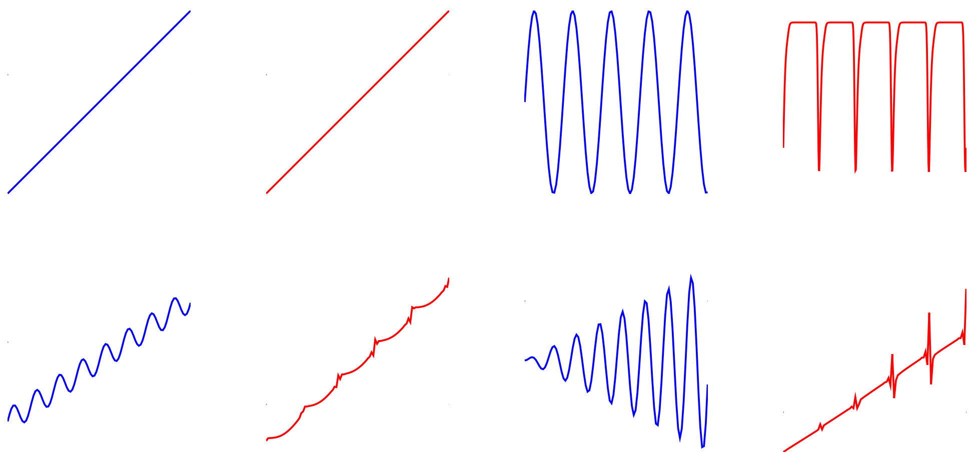
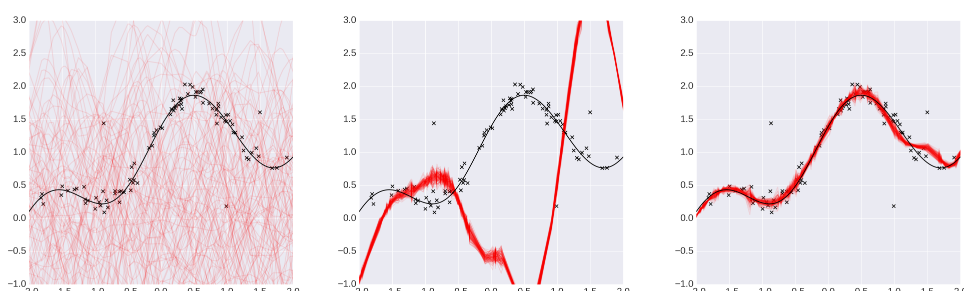


Figure 1: Composition of covariance functions (blue, left) and samples from the distribution of curves they can produce (red, right).



(a) Prior (b) Observed (c) Inferred

Figure 2: 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.

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:

$$P(\mathbf{K} \mid \mathbf{\Omega}, s, n) = P(\mathbf{\Omega} \mid s, n) \times P(s \mid n) \times P(n), \tag{19}$$

with

$$P(\mathbf{\Omega} \mid s, n) = \binom{n}{r} p_{+\times}^{k} (1 - p_{+\times})^{n-k} \tag{20}$$

and

$$P(s \mid n) = \frac{n!}{\mid s \mid !} \tag{21}$$

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 (21). 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:

Listing 4: Grammar to simplify expressions

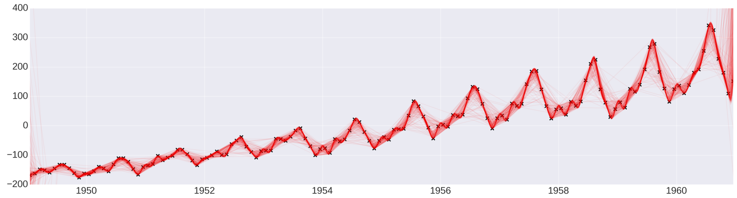
```
SE × SE                        → SE
{SE, PER, C, WN} × WN          → WN
LIN + LIN                      → LIN
{SE, PER, C, WN, LIN} × C      → {SE, PER, C, WN, LIN}
```

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.

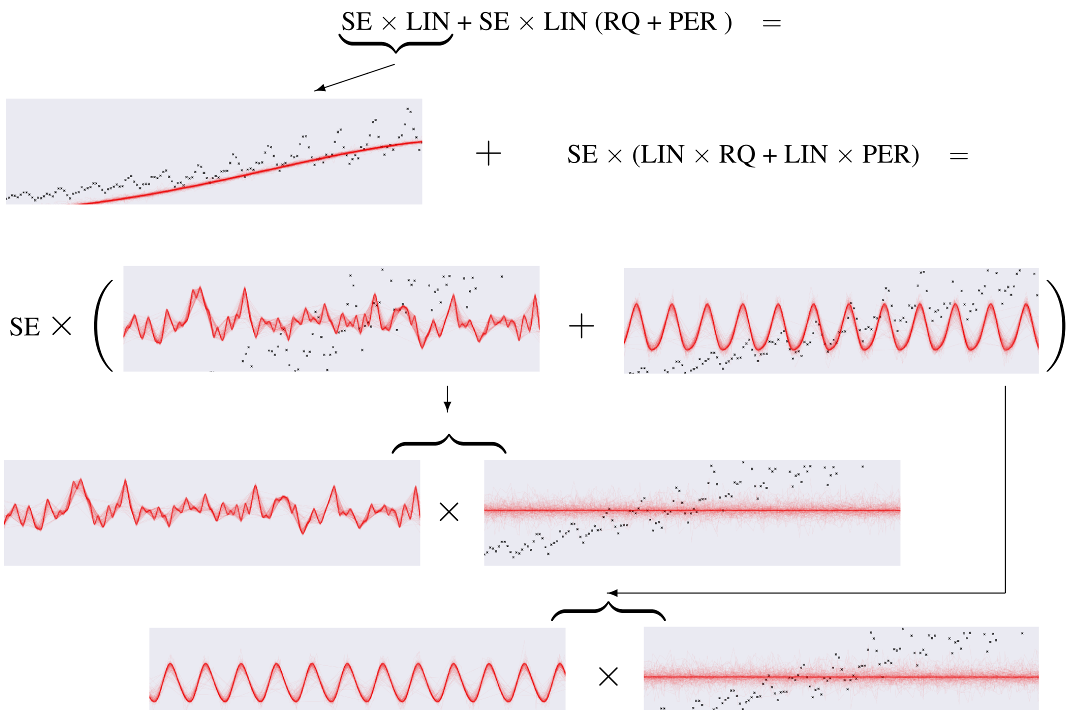
Listing 5: Venture Code for Bayesian GP Structure Learning

```
[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)) ]
```

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.



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



(b) Compositional Structure

Figure 3: a) We see the predictive posterior as an 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. Note that although the overall result is in line with what Duvenaud et al. (2013) report a slightly different composition is implied by the sampled parameters for the structure. 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 line with slope zero.

3). The sample is identical with the highest scoring result reported in previous work using a search-and-score method (Duvenaud et al., 2013) 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. 4).

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 interested to chose certain informative datapoints 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. We we sample a probe from the posterior

$$\hat{\boldsymbol{\theta}} \sim P(\hat{\boldsymbol{\theta}}|\vec{\mathbf{x}}) \tag{22}$$

and then optimize the expected reward by choosing an action a

$$a_i = \arg\max_a \mathbb{E}_{P(\cdots|\theta)}[r(world_\theta(a))] \tag{23}$$

$$x_i \sim world_\theta(a_i) \tag{24}$$

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.

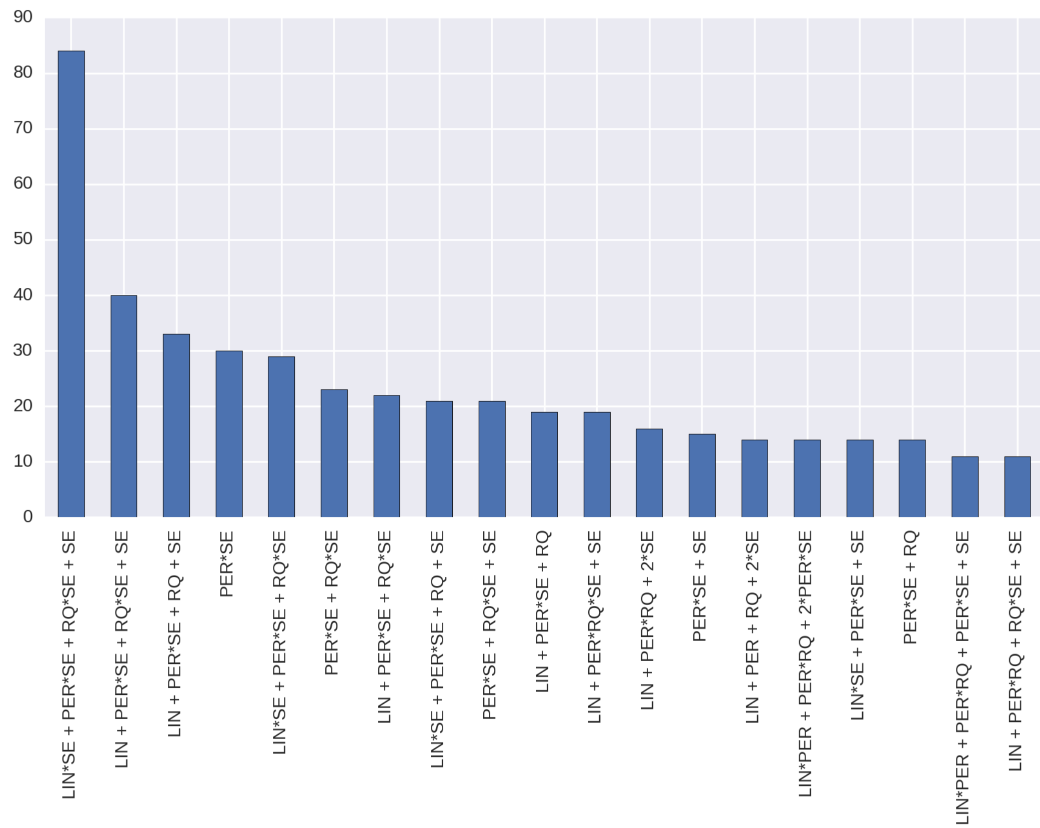


Figure 4: Posterior on structure of the CO2 data. We have cut the tail of the distribution for space reasons since the number of possible structures is large. We see the final sample of the each of the 545 chains with 2000 nested steps each. Note that Duvenaud et al. (2013) report LIN $\times$ SE $+$ PER $\times$ SE $+$ RQ $\times$ SE.