Probabilistic Programming with Gaussian Process Memoization

Ulrich Schaechtle
Department of Computer Science
Royal Holloway, Univ. of London

Ben Zinberg
Computer Science & AI Lab
Massachusetts Institute of Technology

Vikash K. Mansinghka
Computer Science & AI Lab
Massachusetts Institute of Technology

Kostas Stathis
Department of Computer Science
Royal Holloway, Univ. of London

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-cachingâAI 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.

GP Memoization: `gpmem`

Gaussian Processes

$f(\mathbf{x})$ is the multivariate Gaussian $f(\mathbf{x}) \sim \mathcal{N}(0, k(\mathbf{x}, \mathbf{x}))$, where $k(\mathbf{x}, \mathbf{x}') = \mathrm{Cov}_f\left(f(\mathbf{x}), f(\mathbf{x}')\right)$ is the covariance function, a.k.a. kernel. The marginal likelihood can be expressed as:

$$p\left(f(\mathbf{x}) = \mathbf{y} \mid \mathbf{x}\right) = \int p\left(f(\mathbf{x}) = \mathbf{y} \mid f, \mathbf{x}\right)\, p(f|\mathbf{x})\, df$$

A Variation on Memoization

GP memoization produces two components:

$$f \to (f_{\text{compute}}, f_{\text{emu}})$$

- is a generalization of traditional memoization;
- changes semantics of a probabilistic program;
- f_{emu} is a statistical emulator with GP prior;
- each time f_{compute} is called, an observation is incorporated into f_{emu} and predictions improve.

Bayesian GP

We show how we can use `gpmem` to reproduce Neal's Hierarchical Bayesian GP regression [2] for data with outliers.

Regression on data set D

$$\Updownarrow$$

Statistical emulation of $f_{\text{restr}}(x) = \begin{cases} D[x], & \text{if } x \text{ is a data point} \\ \text{Error}, & \text{otherwise} \end{cases}$

Listing 1: Hierarchical GP Smoothing

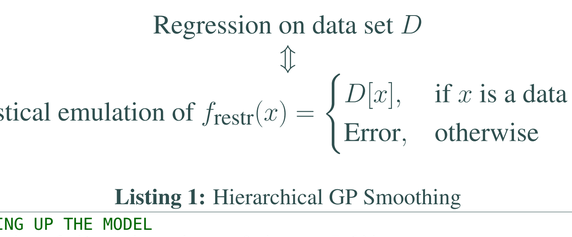
```
;;; SETTING UP THE MODEL
ASSUME alpha-sf (tag 'hyperhyper 0 (gamma 7 1))
ASSUME {{... several more hyper-hyperparameters ...}}
;; Parameters of the covariance function
ASSUME sf (tag 'hyper 0 (log (gamma alpha-sf beta-sf)))
ASSUME l (tag 'hyper 1 (log (gamma alpha-l beta-l)))
ASSUME sigma (tag 'hyper 2 (uniform-continuous 0 2))
;; The covariance function
ASSUME se (make-squaredexp sf l)
ASSUME wn (make-whitenoise sigma)
ASSUME composite-covariance (add-funcs se wn)

;;; PERFORMING INFERENCE
;; Create a prober and emulator using gpmem
ASSUME f-restr (get-neal-prober)
ASSUME compute-and-emu (gpmem f-restr composite-covariance)

;; Probe all data points
PREDICT (mapv (first compute-and-emu) (get-neal-data-xs))

;; Infer hypers and hyperhypers
INFER (repeat 100 (do
    (mh 'hyperhyper one 2)
    (mh 'hyper one 1)))
```

[**TODO:** "Here we use a squared-exponential plus white-noise kernel…" (the parameters ℓ, θ, σ, or whatever notation you choose to use, should be declared here)]

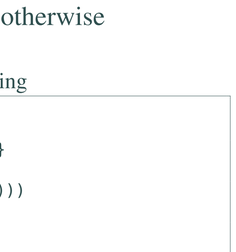
Observed data is generated with the following model:

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

where $\sigma = 0.1$ with probability 0.95, $\sigma = 1$ otherwise.



Above, we see (left to right) predictions before data has been observed, after data has been observed, and after we inferred the hyperparameters. On the right, we see a contour plot and marginals of the the ℓ and θ hyperparameters of the above program after inference (right).



GP Structure Learning

- reasoning over GP kernel structure;
- never done in a fully Bayesian fashion;
- competitive predictive performance; and
- adequate symbolic expressions

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

$$LIN = \theta(xx')$$

$$PER = \theta \exp\left(\frac{2\sin^2(\pi(x-x')/p}{\ell^2}\right)$$

$$RQ = \theta\left(1 + \frac{(x-x')^2}{2\alpha\ell^2}\right)^{-\alpha}$$

Kernel Composition

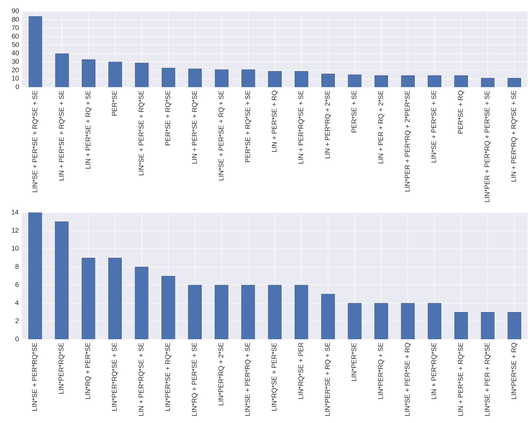
$$\mathbf{K}_{new} = \mathbf{K}_1 + \mathbf{K}_2 \quad \text{or} \quad \mathbf{K}_{new} = \mathbf{K}_1 \times \mathbf{K}_2$$

Listing 2: Venture Code for Bayesian GP Structure Learning

```
;;; GRAMMAR FOR KERNEL STRUCTURE
[ASSUME base-kernels (list K1,K2,···,Kn)] ;; defined as above
[ASSUME pn (uniform-structure n)]       ;; prior on the number of kernels
[ASSUME SK (subset base-kernels pn)]   ;; sampling a subset of size n
[ASSUME composition (lambda (l)          ;; kernel composition
                     (if (lte ( size l) 1)
                         (first l)
                          (if (flip)
                              (add-funcs (first l) (cov-compo (rest l)))
                              (mult-funcs (first l) (cov-compo (rest l)))
                          )
                     )
                )

[ASSUME K (composition  SK)]

[ASSUME (list f-compute f-emu) (gpmem f-restr K )]

for i=1 to n:
  [PREDICT (f-compute x[i])] ;; observing with a look-up function

;;; PERFORMING INFERENCE
[INFER  (repeat 2000 (DO
                          (mh pn one 1)
                          (mh SK one 1)
                          (mh K* one 1)
                          (mh {hyper-parameters} one 10)) ]
```

Results

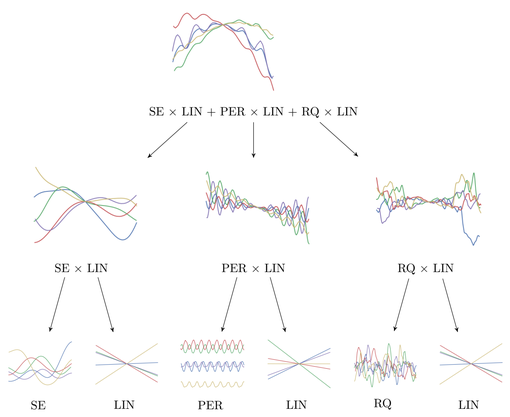
500 and 100 samples drawn from the posterior on structure for two problems used in the Automated Statistician Project [1], the airline dataset (above) and the CO2 dataset (below):



Listing 3: Venture Code for Bayesian GP Structure Learning

```
Place Holder
```

Decomposing a candidate Structure

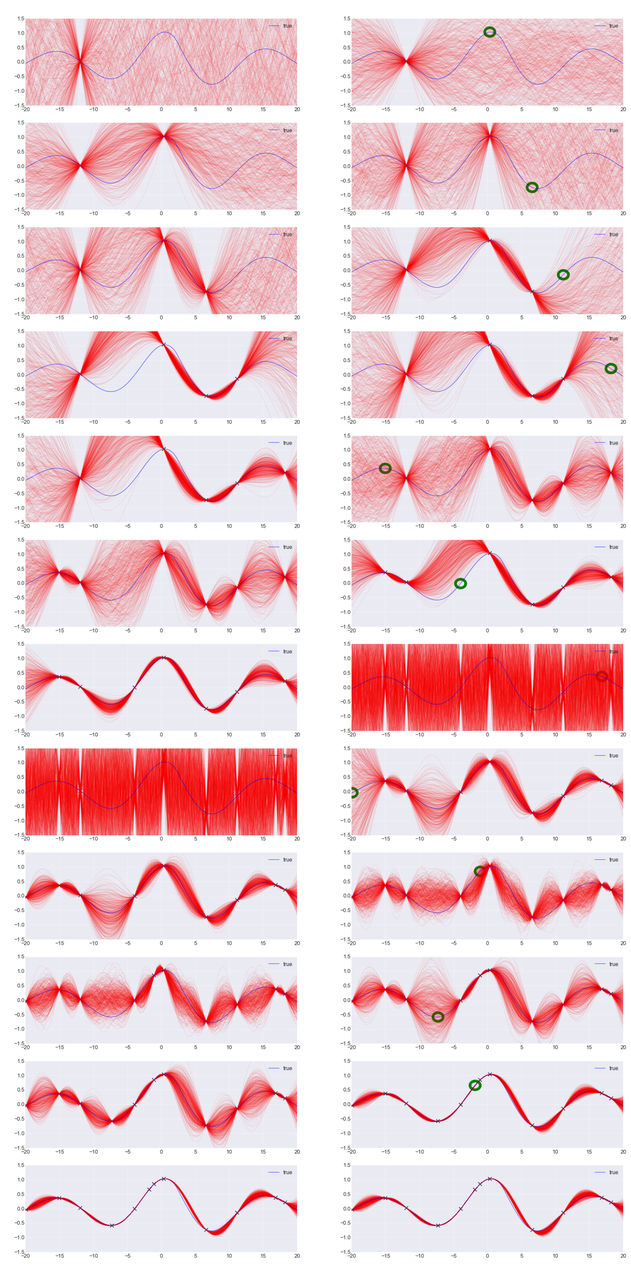


Bayesian Optimization

Listing 4: Venture Code for Bayesian Optimization

```
ASSUME sf1 (tag 'hyper 0 (log (uniform-continuous 0 10)))
ASSUME l1 (tag 'hyper 1 (log (uniform-continuous 0 10)))
ASSUME se (make-se sf1 l1)
ASSUME blackbox-f (get-bayesopt-blackbox)
ASSUME compute-and-emu (gpmem blackbox-f se)

DEFINE get-uniform-candidate
        (lambda (prev-xs) (uniform-continuous -20 20))
DEFINE mc-argmax
        (lambda (emulator prev-xs)
             ((lambda (candidate-xs)
                (lookup candidate-xs
                        (argmax-of-array (mapv emulator candidate-xs))))
              (mapv (lambda (i) (get-uniform-candidate prev-xs))
                    (linspace 0 19 20))))
DEFINE emulator-point-sample
        (lambda (x)
          (run (sample
            (lookup ((second compute-and-emu) (array ,x))
                    0))))

INFER
    (repeat 15 (do pass

      ;; Call f-compute on the next probe point
      (PREDICT ((first compute-and-emu)
                ,(mc-argmax emulator-point-sample '-)))

      ;; Hyperparameter inference
      (mh 'hyper one 50)
```

Thompson Sampling for Homogeneous Sequential Markov Decision Processes



Above: Dynamics of Thompson sampling

- the blue curve is the true function V;
- the red region is a blending of 100 samples of the curve generated (jointly) by a GP-based emulator V_{emu};
- the left and right columns show the state of V_{emu} before and after hyperparameter inference
- in the right column, the next chosen probe point is circled in green;
- each successive probe point a is the (stochastic) maximum of V_{emu}, sampled pointwise and conditioned on the values of the previously probed points; and
- probes tend to happen at points either where the value of V_{emu} is high, or where V_{emu} has high uncertainty.

References

[1] D. Duvenaud, J. R. Lloyd, R. Grosse, J. Tenenbaum, and Z. Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1166–1174, 2013.

[2] R. M. Neal. Monte carlo implementation of gaussian process models for bayesian regression and classification. *arXiv preprint physics/9701026*, 1997.