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

Alexey Radul
Computer Science & AI Lab
Massachusetts Institute of Technology

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

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

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 85-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

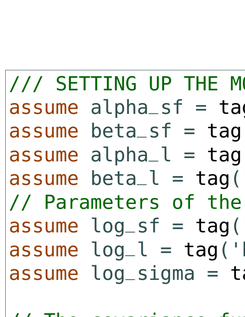
We show how we can use `gpmem` to reproduce Neal's Hierarchical Bayesian GP regression [2] for data with outliers.

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

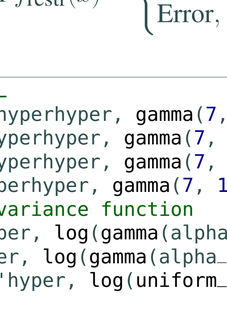
```
/// SETTING UP THE MODEL
assume alpha_sf = tag('hyperhyper, gamma(7, 1))
assume beta_sf = tag('hyperhyper, gamma(7, 1))
assume alpha_l = tag('hyperhyper, gamma(7, 1))
assume beta_l = tag('hyperhyper, gamma(7, 1))
// Parameters of the covariance function
assume log_sf = tag('hyper, log(gamma(alpha_sf, beta_sf)))
assume log_l = tag('hyper, log(gamma(alpha_l, beta_l)))
assume log_sigma = tag('hyper, log(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_blackbox()
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)))
```

Results

Ran the above program on data with outliers: noise $\sigma = 1.0$ with probability 0.05, noise $\sigma = 0.1$ otherwise.



Data unobserved



Data observed

Hyperparameters inferred

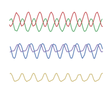
GP Structure Learning

Base Kernels

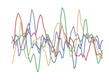
$$SE = \sigma^2 \exp(-\tfrac{(x-x')^2}{2\ell^2})$$

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

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

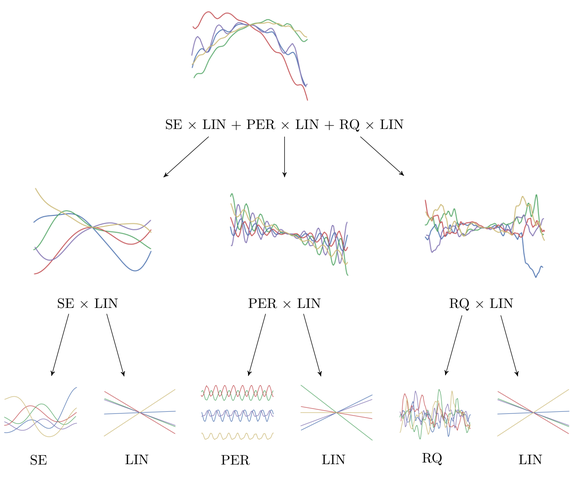


$$RQ = \theta\left(1 + \frac{(x-x')^2}{2\alpha\ell^2}\right)^{-\alpha}$$



Kernel Composition

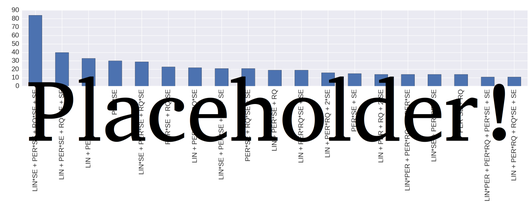
$$\mathbf{K}_{\text{new}} = \mathbf{K}_1 + \mathbf{K}_2 \quad \text{or} \quad \mathbf{K}_{\text{new}} = \mathbf{K}_1 \times \mathbf{K}_2$$




```
// GRAMMAR FOR KERNEL STRUCTURE
assume base_kernels = list(se, wn, lin, per, rq) // defined as above

// prior on the number of kernels
assume p_number_k = tag('number_kernels, uniform_structure(n))
assume s = tag('choice_subset, subset(base_kernels, p_number_k))

assume cov_compo = proc(l) {
  // kernel composition
  if (size(l) <= 1)
  then { first(l) }
  else { if (flip()) then { add_funcs(first(l), cov_compo(rest(l))) }
                     else { mult_funcs(first(l), cov_compo(rest(l))) }
       }
  })

assume K = tag('composit, cov_compo(s))

assume (f_compute f_emu) = gpmem(f_restr, K)
predict mapv(f_compute(get_data_xs)) // probe all data points

// PERFORMING INFERENCE
infer  repeat(2000, do(
                        mh('number_kernel, one, 1),
                        mh('choice_subset, one, 1),
                        mh('composit, one, 1),
                        mh('hyper, one, 10)))
```

Results

500 samples drawn from the posterior on structure for the CO2 dataset used in the Automated Statistician Project [1]:



Posterior samples for curves for the airline dataset where SE × LIN + PER × LIN + RQ × LIN is sampled from the structure grammar:

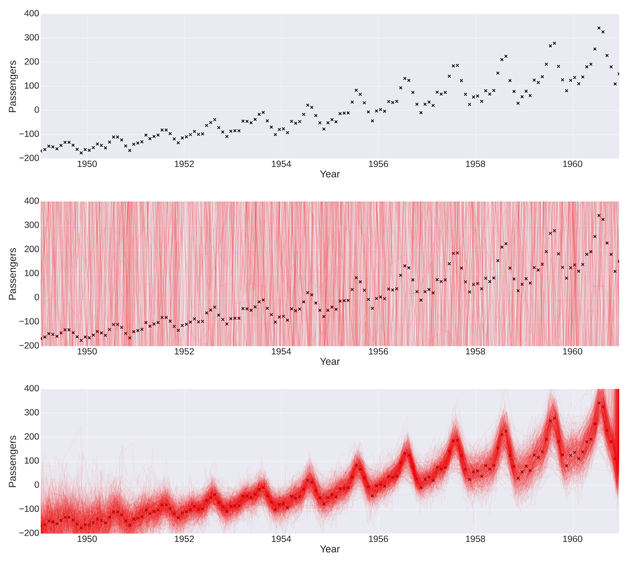


Figure 1: Top: raw data, monthly airline passengers. Middle, prior on functions. Bottom, posterior.

Bayesian Optimization

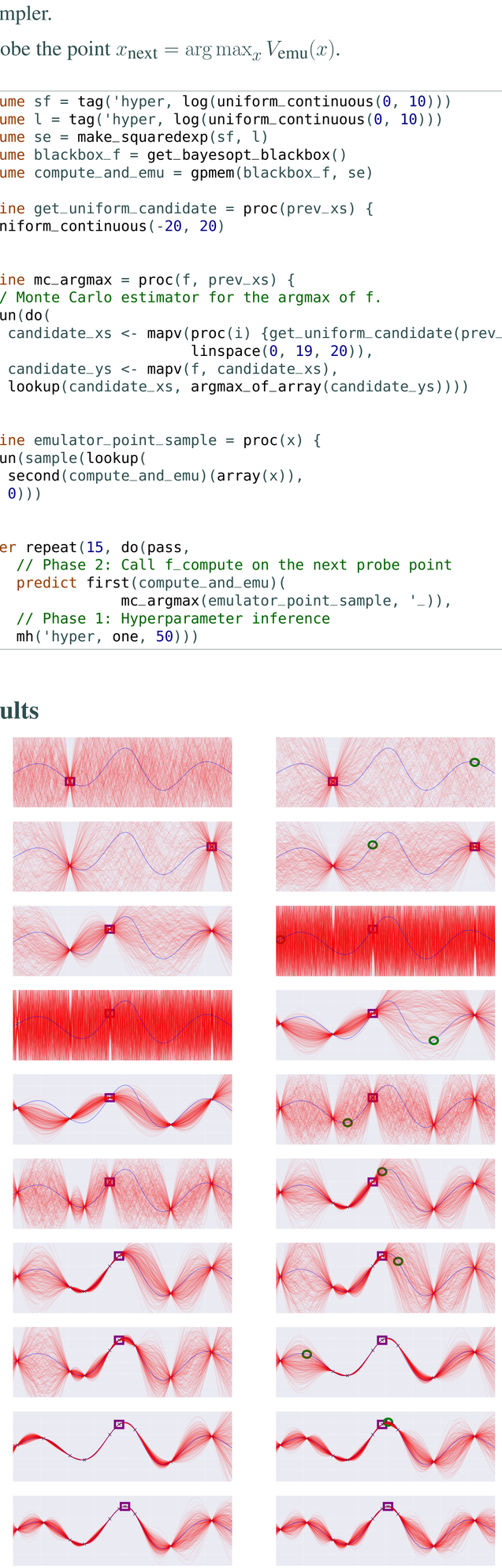
Thompson Sampling for Homogeneous Sequential Markov Decision Processes

Two phases, each takes one line of code:

1. Sample a context $V_{\text{emu}} \sim P(V_{\text{emu}})$ using a Metropolis–Hastings sampler.
2. Probe the point $x_{\text{next}} = \arg\max_x V_{\text{emu}}(x)$.

```
assume sf = tag('hyper, log(uniform_continuous(0, 10)))
assume l = tag('hyper, log(uniform_continuous(0, 10)))
assume se = make_squaredexp(sf, l)
assume blackbox_f = get_bayesopt_blackbox()
assume compute_and_emu = gpmem(blackbox_f, se)

define get_uniform_candidate = proc(prev_xs) {
  uniform_continuous(-20, 20)
}

define mc_argmax = proc(f, prev_xs) {
  // Monte Carlo estimator for the argmax of f.
  run(do(
    candidate_xs <- mapv(proc(i) {get_uniform_candidate(prev_xs)},
                         linspace(0, 19, 20)),
    candidate_ys <- mapv(f, candidate_xs),
    lookup(candidate_xs, argmax_of_array(candidate_ys))))
}

define emulator_point_sample = proc(x) {
  run(sample(lookup(
    second(compute_and_emu)(array(x)),
    0)))
}

infer repeat(15, do(pass,
     // Phase 2: Call f_compute on the next probe point
     predict first(compute_and_emu)(
                 mc_argmax(emulator_point_sample, '_)),
     // Phase 1: Hyperparameter inference
     mh('hyper, one, 50)))
```

Results



- Blue: Unknown true value function V
- Red: Current state of our statistical emulator V_{emu}, conditioned on previously probed values of V
- Purple: Best probe point seen so far, x_{best}
- Green: Next chosen probe point, x_{next}
- Left: before hyperparameter inference; right: after hyperparameter inference

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.