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

Gaussian process memoizer is 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. The work 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 and (iii) a bandit formulation of Bayesian optimization.

GP Memoization: `gpmem`

Gaussian Processes

For any finite set of inputs $\mathbf{x}$, the marginal prior on $f(\mathbf{x})$ is the multivariate Gaussian $f(\mathbf{x}) \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}))$, 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:

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

where here $p(f|\mathbf{x}) = p(f) \sim \mathcal{GP}(m, k)$ since we assume no dependence of f on $\mathbf{x}$. We can sample a vector of unseen data $\mathbf{y}^* = f(\mathbf{x}^*)$ from the predictive posterior with

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

with $\boldsymbol{\mu} = k(\mathbf{x}, \mathbf{x}^*)\, k(\mathbf{x}^*, \mathbf{x}^*)^{-1}\, \mathbf{y}$ and $\boldsymbol{\Sigma} = k(\mathbf{x}, \mathbf{x}) - k(\mathbf{x}, \mathbf{x}^*) k(\mathbf{x}^*, \mathbf{x}^*)^{-1} k(\mathbf{x}^*, \mathbf{x})$'

Memoization

Memoization is the practice of storing previously computed values of a function so that future calls with the same inputs can be evaluated by lookup rather than re-computation. Gaussian Process memoization:

- is a statistical alternative to standard memoization;
- changes semantics of a probabilistic program; and
- produces a statistical emulator whose predictions improve with memoization.

Bayesian GP

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

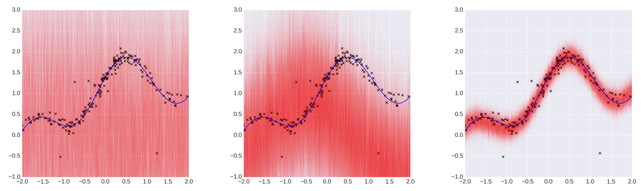
Listing 1: Hierarchical GP Smoothing

```
[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}
[ASSUME sigma (uniform 0 5 )] ∈ {hyper-parameters} % Fig. ??
% above: structured prior, Fig. ??
k_SE(x, x') := θ^2 exp(-(x-x')^2 / (2ℓ^2))
k_WN(x, x') := σ^2 δ_{x,x'}
[ASSUME k_SE VentureFunction(k_SE, θ, ℓ) ]
[ASSUME k_WN VentureFunction(k_WN, σ) ]
[ASSUME SE make-se (apply-function k_SE l sf) ]
[ASSUME WN make-se (apply-function k_WN sigma) ]
[ASSUME (list f_compute f_emu) (gpmem f_restr (function-plus SE WN) )]
[SAMPLE (f_emu (array 1 2 3))] % prior, Fig. ??
for i=1 to n:
  [PREDICT (f_compute x[i])] % observing with a look-up function
[SAMPLE (f_emu (array 1 2 3))] % after observation, Fig. ??
[INFER (REPEAT 100
        (DO (MH {hyper-parameters} one 2)
            (MH {hyper-parameters-Γ} one 2) ))]
[SAMPLE (f_emu (array 1 2 3))] % posterior , Fig. ??
```

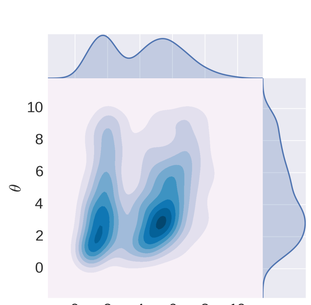
Observed data is generated with the following model:

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

where σ is 0.1 with probability 0.95, 1 otherwise.



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

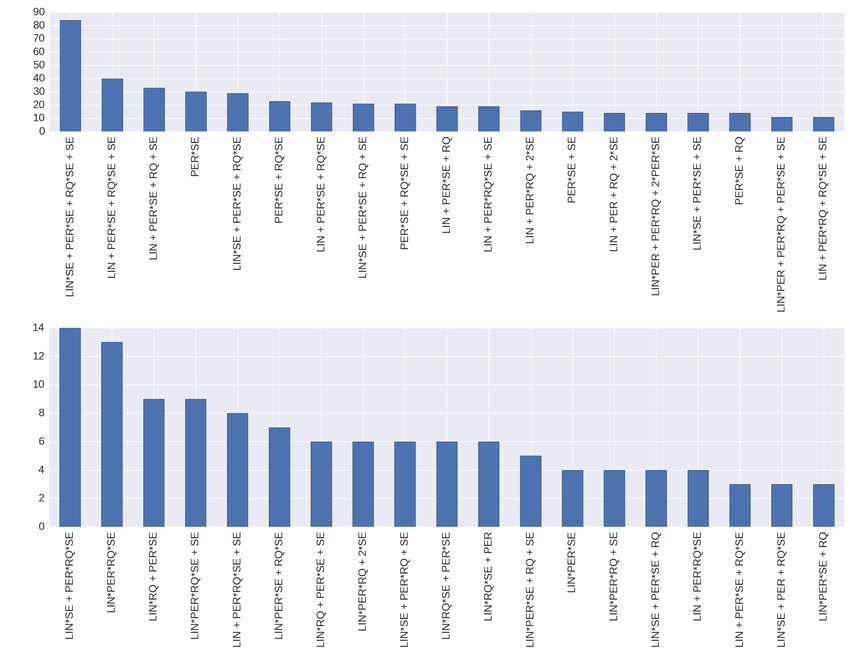


Learning of Symbolic Structure

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

Results

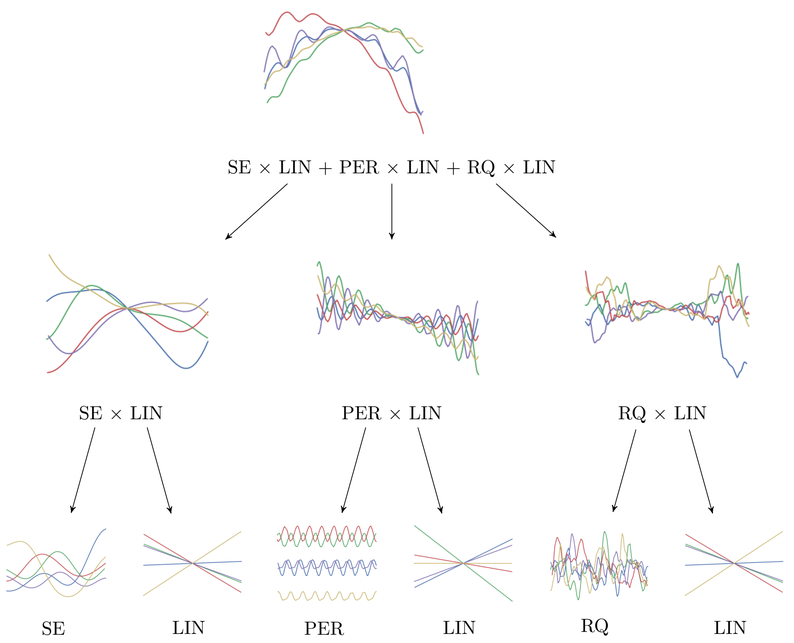
500 and 100 samples drawn from the posterior on structure for two problems used in the Automated Statistician Project [1]:



Listing 2: Venture Code for Bayesian GP Structure Learning

```
[ASSUME base_kernels (list K_1 ,K_2 ,⋯,K_n )] % defined as above
[ASSUME p_n (uniform_structure n)] % prior on the number of kernels
[ASSUME S_K (subset base_kernels p_n )] % sampling a subset of size n
[ASSUME composition (lambda (l) % kernel composition
                     (if (lte ( size l) 1)
                         (first l)
                          (if (flip)
                              (func_plus (first l) (cov_compo (rest l)))
                              (func_times (first l) (cov_compo (rest l)))
                          )
                     )
                )

[ASSUME K (composition S_K)]

[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

[INFER (REPEAT 2000 (DO
                            (MH p_n one 1)
                            (MH S_K one 1)
                            (MH K* one 1)
                            (MH {hyper-parameters} one 10)) ]
```

Decomposing a candidate Structure



Bayesian Optimization

Thompson Sampling

In Thompson sampling, the goal is to maximize the total reward received for all actions by placing a prior distribution $P(\theta)$ on the possible "contexts" $\theta \in \Theta$ with reward $r \in \mathbb{R}$.

Repeat:

1. Sample a context $\theta \sim P(\theta)$.
2. Choose an action $a \in \mathcal{A}$ which (approximately) maximizes $V(a|\theta) = \mathbb{E}\left[r|a, \theta\right]$.
3. Let r_{true} be the reward received for action a. Update the believed distribution on θ, i.e., $P(\theta) \leftarrow P_{\text{new}}(\theta)$ where $P_{\text{new}}(\theta) = P\left(\theta \mid a \mapsto r_{\text{true}}\right)$.

Bayesian Optimization with `gpmem`

- contexts $\theta = (\mu, K)$ are Gaussian processes over the action space $\mathcal{A} = \mathbb{R}$.
- $K_{\text{prior}} = K_{\text{prior},\sigma,\ell}$ is a procedure, with parameters σ, ℓ, to be used as the prior covariance function: $K_{\text{prior}}(a, a') = \sigma^2 \exp\left(-\frac{(a-a')^2}{2\ell^2}\right)$
- σ and ℓ are (hyper)parameters for K_{prior}
- $\mathbf{a}_{\text{past}} = (a_i)_{i=1}^n$ are the previously probed actions
- $\mathbf{r}_{\text{past}} = (r_i)_{i=1}^n$ are the corresponding rewards

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.

Conclusions

`gpmem` is a generalization of a number of GP related models and can be applied to:

- Hierarchical Bayesian GP
- Kernel Structure Learning
- Bayesian Optimization

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.