Supplement - Deep Structured Mixtures of Gaussian Processes

Martin Trapp
TU Graz & OFAI
SPSC Lab

Robert Peharz
TU Eindhoven
Information Systems WSK&I

Franz Pernkopf
TU Graz
SPSC Lab

Carl Edward Rasmussen
University of Cambridge
CBL Lab

0.1 Shared Cholesky Decomposition

We naturally have overlapping local GPs in DSMGPs and, therefore, experts at the leaves share parts of their kernel matrix. This property can be utilised to share solutions of the Cholesky decompositions, which speeds up computations. Therefore, let us consider the case in which two leaves, denoted as L_i and L_j, are such that $\mathcal{X}_{\mathsf{L}_j}$ is contained in $\mathcal{X}_{\mathsf{L}_i}$. Further, let us consider the scenarios for which the number of observation in $\mathcal{X}_{\mathsf{L}_j}$ is less than in $\mathcal{X}_{\mathsf{L}_i}$, i.e. $\#\mathbf{X}_{(j)} < \#\mathbf{X}_{(i)}$ where $\mathbf{X}_{(i)}$ is shorthand for $\mathbf{X}_{(\mathsf{L}_i)}$.

In the first scenario the kernel matrix $k_{\mathbf{X}_{(j)},\mathbf{X}_{(j)}}$ of L_j is a submatrix of the kernel matrix of L_i and $[k_{\mathbf{X}_{(j)},\mathbf{X}_{(j)}}]_{1,1} = [k_{\mathbf{X}_{(i)},\mathbf{X}_{(i)}}]_{1,1}$. Therefore, the lower-triangular matrix of the Cholesky decomposition for the kernel matrix of L_j is a submatrix of the decomposition for the kernel matrix of L_i. Let L_{L_i} and L_{L_j} denote the lower-triangular matrix of the Cholesky decomposition for the respective kernel matrices. Then,

$$L_{\mathsf{L}_i} = \begin{bmatrix} L_{\mathsf{L}_j} & v^T \\ v & \tilde{L} \end{bmatrix}, \tag{1}$$

where the vector $v \in \mathbb{R}^P$ and $\tilde{L} \in \mathbb{R}^{P \times P}$ with P being the additional dimensions contained in L_{L_i}. Thus, we can copy the respective sub-matrix to obtain L_{L_j}.

In the second scenario $[k_{\mathbf{X}_{(j)},\mathbf{X}_{(j)}}]_{1,1} \neq [k_{\mathbf{X}_{(i)},\mathbf{X}_{(i)}}]_{1,1}$ but both kernel matrices share the last column/row. This scenario can be solved efficiently using rank-1 updates. Therefore, let L_{L_i} be defined as

$$L_{\mathsf{L}_i} = \begin{bmatrix} l_{1,1} & \mathbf{0} \\ \boldsymbol{l}_{2:N,1} & L_{2:N,2:N} \end{bmatrix}, \tag{2}$$

and let us assume that the kernel matrix of L_j contains all observations the kernel matrix of L_i contains, except the first one, i.e. the first index. We now aim to obtain L_{L_j} without solving the Cholesky decomposition explicitly. For this purpose, let A be

$$A = \begin{bmatrix} 0 & \mathbf{0} \\ \mathbf{0} & \tilde{L}_{2:N,2:N} \end{bmatrix}, \tag{3}$$

and let $\tilde{L}_{2:N,2:N} = L_{\mathsf{L}_j}$ be the sub-matrix of interest. Using a rank-1 update with $\boldsymbol{l}_{2:N,1}$, i.e.

$$\tilde{L}_{2:N,2:N} = L_{2:N,2:N} + \boldsymbol{l}_{2:N,1}\boldsymbol{l}_{2:N,1}^T, \tag{4}$$

we can efficiently obtain L_{L_j} by solving Equation (4) and dropping the first column and row of A. Note that in case of multiple missing observations, we can apply rank-1 updates on A consecutively. To perform rank-1 updates numerically stable we use the approach in [Seeger, 2008]. Note that other scenarios are either a combination of the two discussed scenarios, or can be solved by continuing the Cholesky decomposition after applying rank-1 updates or have to be solved directly to obtain sufficiently stable results[1].

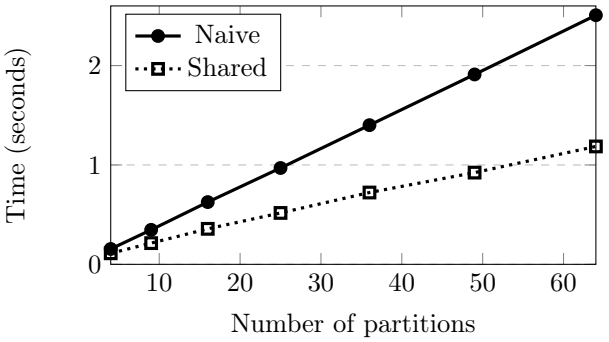

Figure 1: Time required to solve the Cholesky decomposition of a DSMGP on a synthetic dataset using a naive approach or using our shared approach.

We empirically evaluated the performance gains through sharing solutions of the Cholesky decompositions, shown in Figure 1. The plot compares the runtime, measured on an i7-6900k CPU @ 3.2 GHz, for a synthetic dataset consisting of $1,000$ observations against an increasing number of partitions. We see that sharing Cholesky decompositions reduces the runtime by a factor of two, allowing us to explore twice as many partitions of the input space.

[1] We empirically evaluated the numerical errors for different scenarios and found that rank-1 downgrades do not result in numerically stable solutions.

0.2 Datasets

If available we used the existing training set / testing set splits and otherwise randomly split the dataset into 70% for training and 30% for testing.

We pre-processed each dataset to have zero mean and unit variance – in the inputs and outputs – and used a zero mean function for each approach. Note that all of the datasets (without pre-processing) can be found on GitHub under `https://github.com/trappmartin/DeepStructuredMixtures/releases/download/v0.1/datasets.tar.gz`.

Table 1: Statistics of benchmark datasets. For each dataset we list the number of training samples N (train), the number of test samples N (test), the number of input dimensions (D) and the number of output dimensions (P).

Dataset	N (train)	N (test)	D	P
Airfoil	1,052	451	5	1
Parkin.	4,112	1,763	16	2
Kin40k	10,000	30,000	8	1
House	15,949	6,835	16	1
Protein	32,011	13,719	9	1
Year	360,742	154,603	90	1
Flight	500,000	200,000	8	2

0.3 Scores

To assess the performance we computed the root mean squared error (RMSE), the mean absolute error (MAE) and the negative log predictive density (NLPD), i.e.

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2}\,, \tag{5}$$

$$\text{MAE} = \frac{1}{N}\sum_{n=1}^{N}|\hat{y}_n - y_n|\,, \tag{6}$$

$$\text{NLPD} = -\log p(y_n \,|\, \mathcal{D}, \mathbf{x}_n, \theta)\,, \tag{7}$$

where $\hat{y}_n$ is the prediction for test datum n and $\mathcal{D}$ is the training set.

0.4 Algorithms

We applied the structure construction algorithm described in Section 5.1 in the paper to automatically build hierarchical structures. In the following text, we will explain the algorithms for structure construction, posterior inference in pseudo-code. For an efficient implementation of the algorithms, we refer to the Julia package https://github.org/trappmartin/DeepStructuredMixtures.

0.4.1 Structure Construction

The Algorithm 1 recursively creates a tree structured DSMGP containing sum nodes with K_{S} many children and product nodes with K_{P} many children. The argument $\min N$ controls the minimum number of observations per GP expert. Note that in the Julia implementation provided on GitHub, we additionally control for the number of recursions, that is the number of consecutive sum and product nodes. Note that `N isa S` denotes a check if `N` is a `S` or not and leverage the Julia syntax of using an exclamation mark to denote an in-place operation, e.g. `push!(`Q`, N)` adds `N` into Q.

0.4.2 Exact Posterior Inference

The following sub-section illustrates the implementation of exact posterior inference in DSMGPs. The procedure shown in 2 recursively performs exact posterior updates and is called using the root node of the DSMGP. Note that for reasons of numerical stability, an actual implementation of the algorithm will need to perform the operations in log-space. Again, we refer to the accompanied Julia implementation for an efficient example implementation.

References

[Rasmussen and Williams, 2006] Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press.

[Seeger, 2008] Seeger, M. (2008). Low rank updates for the cholesky decomposition. Technical report, University of California at Berkeley.

Algorithm 1 structure construction algorithm

procedure LEARNDSMGP($K_{\mathsf{S}}, K_{\mathsf{P}}, \min N$)
  $\mathcal{S} \leftarrow \mathsf{S}$
  $Q \leftarrow$ empty queue
  push!$(Q, \mathcal{S})$
  while $Q \neq \emptyset$ **do**
    $\mathsf{N} \leftarrow$ pop!(Q)
    if N isa S **then**
      for $k = 1, \ldots, K_{\mathsf{S}}$ **do**
        $\mathbf{ch}(\mathsf{N})[k] \leftarrow \mathsf{P}$
        $w_{\mathsf{N},\mathbf{ch}(\mathsf{N})[k]} = \frac{1}{K_{\mathsf{S}}}$
        push!$(Q, \mathbf{ch}(\mathsf{N})[k])$
    else if N isa P **then**
      $\mathcal{D}^{(\mathsf{N})} \leftarrow \{\mathbf{X}^{(\mathsf{N})}, \mathbf{y}^{(\mathsf{N})}\}$
      $d \sim [\text{variance}(\mathbf{X}_i^{(\mathsf{N})}) \, \forall i]$
      $d_{\min} \leftarrow \min \mathbf{X}_d^{(\mathsf{N})}$
      $d_{\text{med}} \leftarrow \text{median}(\mathbf{X}_d^{(\mathsf{N})})$
      $v \leftarrow \max \mathbf{X}_d^{(\mathsf{N})} - d_{\min}$
      for $k = 1, \ldots, K_{\mathsf{P}} - 1$ **do**
        $s_k \sim 0.5[v\mathcal{B}eta(2,2) + d_{\min}] + 0.5 d_{\text{med}}$
      sort!(s)
      $s_{\min} \leftarrow d_{\min}$
      for $k = 1, \ldots, K_{\mathsf{P}} - 1$ **do**
        $s_{\max} \leftarrow s[k]$
        $\mathcal{D}^{(\mathsf{C})} \leftarrow \mathcal{D}^{(\mathsf{N})}[s_{\min} : s_{\max}]$
        if $\#\mathcal{D}^{(\mathsf{C})} > \min N$ **then**
          $\mathbf{ch}(\mathsf{N})[k] \leftarrow \mathsf{S}$
          $\mathcal{D}^{\mathbf{ch}(\mathsf{N})[k]} \leftarrow \mathcal{D}^{(\mathsf{C})}$
          push!$(Q, \mathbf{ch}(\mathsf{N})[k])$
          $s_{\min} \leftarrow s[k]$
        else
          $\mathbf{ch}(\mathsf{N})[k] \leftarrow \mathsf{L}$
          $\mathcal{D}^{\mathbf{ch}(\mathsf{N})[k]} \leftarrow \mathcal{D}^{(\mathsf{C})}$
      $s_{\max} \leftarrow \max \mathbf{X}^{(\mathsf{N})}$
      $\mathcal{D}^{(\mathsf{C})} \leftarrow \mathcal{D}^{(\mathsf{N})}[s_{\min} : s_{\max}]$
      if $\#\mathcal{D}^{(\mathsf{C})} > \min N$ **then**
        $\mathbf{ch}(\mathsf{N})[k] \leftarrow \mathsf{S}$
        $\mathcal{D}^{\mathbf{ch}(\mathsf{N})[k]} \leftarrow \mathcal{D}^{(\mathsf{C})}$
        push!$(Q, \mathbf{ch}(\mathsf{N})[k])$
      else
        $\mathbf{ch}(\mathsf{N})[k] \leftarrow \mathsf{L}$
        $\mathcal{D}^{\mathbf{ch}(\mathsf{N})[k]} \leftarrow \mathcal{D}^{(\mathsf{C})}$

Algorithm 2 Exact posterior inference

procedure EXACTINFERENCE(N)
  $z \leftarrow 0$
  if N isa S **then**
    for $\mathsf{C} \in \mathbf{ch}(\mathsf{N})$ **do**
      $w_{\mathsf{N},\mathsf{C}} \leftarrow w_{\mathsf{N},\mathsf{C}} *$ EXACTINFERENCE(C)
      $z \leftarrow z + w_{\mathsf{N},\mathsf{C}}$
    for $\mathsf{C} \in \mathbf{ch}(\mathsf{N})$ **do**
      $w_{\mathsf{N},\mathsf{C}} \leftarrow w_{\mathsf{N},\mathsf{C}} / z$
  else if N isa P **then**
    for $\mathsf{C} \in \mathbf{ch}(\mathsf{N})$ **do**
      $z \leftarrow z +$ EXACTINFERENCE(C)
  else
    $z \leftarrow p_{\mathsf{N}}(\boldsymbol{y} \mid \boldsymbol{X})$ [Rasmussen and Williams, 2006]
  return z