Scalable Gaussian Processes

Zhenwen Dai

Amazon

September 4, 2018 @GPSS2018
Gaussian process

Input and Output Data:

\[ y = (y_1, \ldots, y_N), \quad X = (x_1, \ldots, x_N)^\top \]

\[ p(y|f) = \mathcal{N}(y|f, \sigma^2 I), \quad p(f|X) = \mathcal{N}(f|0, K(X, X)) \]
The scaling behavior w.r.t. $N$

The computational cost of Gaussian process is $O(N^3)$. 

![Graph showing the scaling behavior of Gaussian processes with respect to data size $N$. The x-axis represents the data size $N$, and the y-axis represents time in seconds. The graph shows a clear upward trend as $N$ increases, indicating a cubic relationship.]
Behind a Gaussian process fit

- Point Estimate / Maximum A Posteriori (MAP) of hyper-parameters.

\[ \theta^* = \arg \max_{\theta} \log p(y|X, \theta) = \arg \max_{\theta} \log \mathcal{N}(y|0, K + \sigma^2 I) \]

- Prediction on a test point given the observed data and the optimized hyper-parameters.

\[ p(f_*|X_*, y, X, \theta) = \mathcal{N}(f_*|K_*(K + \sigma^2 I)^{-1}y, K_{**} - K_*(K + \sigma^2 I)^{-1}K_*^\top) \]
How to implement the log-likelihood (1)

- Compute the covariance matrix $K$:

$$K = \begin{pmatrix}
    k(x_1, x_1) & \cdots & k(x_1, x_N) \\
    \vdots & \ddots & \vdots \\
    k(x_N, x_1) & \cdots & k(x_N, x_N)
\end{pmatrix}$$

where $k(x_i, x_j) = \gamma \exp \left( -\frac{1}{2l^2}(x_i - x_j)\top(x_i - x_j) \right)$

- The complexity is $O(N^2Q)$. 
How to implement the log-likelihood (2)

- Plug in the log-pdf of multi-variate normal distribution:

\[
\log p(y|X) = \log \mathcal{N} (y|0, K + \sigma^2 I)
\]
\[
= - \frac{1}{2} \log |2\pi (K + \sigma^2 I)| - \frac{1}{2} y^\top (K + \sigma^2 I)^{-1} y
\]
\[
= - \frac{1}{2} (||L^{-1} y||^2 + N \log 2\pi) - \sum_i \log L_{ii}
\]

- Take a Cholesky decomposition: \( L = \text{cho}(K + \sigma^2 I) \).

- The computational complexity is \( O(N^3 + N^2 + N) \). Therefore, the overall complexity including the computation of \( K \) is \( O(N^3) \).
A quick profiling \((N=1000, Q=10)\)

Time unit is microsecond.

<table>
<thead>
<tr>
<th>Line #</th>
<th>Time</th>
<th>% Time</th>
<th>Line Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6.0</td>
<td>0.0</td>
<td>def log_likelihood(kern, X, Y, sigma2):</td>
</tr>
<tr>
<td>3</td>
<td>55595.0</td>
<td>58.7</td>
<td>N = X.shape[0]</td>
</tr>
<tr>
<td>4</td>
<td>4369.0</td>
<td>4.6</td>
<td>K = kern.K(X)</td>
</tr>
<tr>
<td>5</td>
<td>30012.0</td>
<td>31.7</td>
<td>Ky = K + np.eye(N)*sigma2</td>
</tr>
<tr>
<td>6</td>
<td>4361.0</td>
<td>4.6</td>
<td>L = np.linalg.cholesky(Ky)</td>
</tr>
<tr>
<td>7</td>
<td>49.0</td>
<td>0.1</td>
<td>LinvY = dtrtrs(L, Y, lower=1)[0]</td>
</tr>
<tr>
<td>8</td>
<td>82.0</td>
<td>0.1</td>
<td>logL = N<em>np.log(2</em>np.pi)/-2.</td>
</tr>
<tr>
<td>9</td>
<td>208.0</td>
<td>0.2</td>
<td>logL += np.square(LinvY).sum()/-2.</td>
</tr>
<tr>
<td>10</td>
<td>2.0</td>
<td>0.0</td>
<td>logL += -np.log(np.diag(L)).sum()</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td>return logL</td>
</tr>
</tbody>
</table>
Too slow or too many data points?

A lot of data does not necessarily mean a complex model.
Pseudo Data

Summarize real data into a small set of pseudo data.
Sparse Gaussian Process

Sparse GPs refers to a family of approximations:

- Nyström approximation [Williams and Seeger, 2001]
- Fully independent training conditional (FITC) [Snelson and Ghahramani, 2006]
- Variational sparse Gaussian process [Titsias, 2009]
Approximation by subset

Let’s randomly pick a subset from the training data: $Z \in \mathbb{R}^{M \times Q}$.

Approximate the covariance matrix $K$ by $\tilde{K}$.

\[ \tilde{K} = K_z K_{zz}^{-1} K_z^\top, \text{ where } K_z = K(X, Z) \text{ and } K_{zz} = K(Z, Z). \]

Note that $\tilde{K} \in \mathbb{R}^{N \times N}$, $K_z \in \mathbb{R}^{N \times M}$ and $K_{zz} \in \mathbb{R}^{M \times M}$.

The log-likelihood is approximated by

\[ \log p(y|X, \theta) \approx \log \mathcal{N}(y|0, K_z K_{zz}^{-1} K_z^\top + \sigma^2 I). \]
Efficient computation using Woodbury formula

- The naive formulation does not bring any computational benefits.

\[
\tilde{\mathcal{L}} = -\frac{1}{2} \log |2\pi(\tilde{K} + \sigma^2 I)| - \frac{1}{2} y^\top (\tilde{K} + \sigma^2 I)^{-1} y
\]

- Apply the Woodbury formula:

\[
(K_z K_{zz}^{-1} K_z^\top + \sigma^2 I)^{-1} = \sigma^{-2} I - \sigma^{-4} K_z (K_{zz} + \sigma^{-2} K_z^\top K_z)^{-1} K_z^\top
\]

- Note that \((K_{zz} + \sigma^{-2} K_z^\top K_z) \in \mathbb{R}^{M \times M}\).

- The computational complexity reduces to \(O(NM^2)\).
Nyström approximation

- The above approach is called Nyström approximation by Williams and Seeger [2001].
- The approximation is directly done on the covariance matrix without the concept of pseudo data.
- The approximation becomes exact if the whole data set is taken, i.e., $KK^{-1}K^\top = K$.
- The subset selection is done randomly.
Snelson and Ghahramani [2006] proposes the idea of having pseudo data. This approach is later referred to as Fully independent training conditional (FITC).

Augment the training data \((X, y)\) with pseudo data \(u\) at location \(Z\).

\[
p \left( \begin{bmatrix} y \\ u \end{bmatrix} \mid \begin{bmatrix} X \\ Z \end{bmatrix} \right) = \mathcal{N} \left( \begin{bmatrix} y \\ u \end{bmatrix} \mid 0, \begin{bmatrix} K_{ff} + \sigma^2 I & K_{fu} \\ K_{fu}^\top & K_{uu} \end{bmatrix} \right)
\]

where \(K_{ff} = K(X, X)\), \(K_{fu} = K(X, Z)\) and \(K_{uu} = K(Z, Z)\).
Gaussian process with Pseudo Data (2)

- Thanks to the marginalization property of Gaussian distribution,
  \[ p(y|X) = \int_u p(y, u|X, Z). \]

- Further re-arrange the notation:
  \[ p(y, u|X, Z) = p(y|u, X, Z)p(u|Z) \]

where
\[ p(u|Z) = \mathcal{N}(u|0, K_{uu}), \]
\[ p(y|u, X, Z) = \mathcal{N}(y|K_{fu}K_{uu}^{-1}u, K_{ff} - K_{fu}K_{uu}^{-1}K_{fu}^\top + \sigma^2 I). \]
So far, \( p(y|X) \) has not been changed, but there is no speed-up, \( K_{ff} \in \mathbb{R}^{N \times N} \) in \( K_{ff} - K_{fu} K_{uu}^{-1} K_{fu}^\top + \sigma^2 I \).

The FITC approximation assumes

\[
\tilde{p}(y|u, X, Z) = \mathcal{N} \left( y | K_{fu} K_{uu}^{-1} u, \Lambda + \sigma^2 I \right),
\]

where \( \Lambda = (K_{ff} - K_{fu} K_{uu}^{-1} K_{fu}^\top) \circ I \).
FITC approximation (2)

- Marginalize $u$ from the model definition:

$$
\tilde{p}(y|X, Z) = \mathcal{N}(y|0, K_{fu}K_{uu}^{-1}K_{fu}^\top + \Lambda + \sigma^2 I)
$$

- Woodbury formula can be applied in the same way as in Nyström approximation:

$$
(K_zK_{zz}^{-1}K_z^\top + \Lambda + \sigma^2 I)^{-1} = A - AK_z(K_{zz} + K_z^\top AK_z)^{-1}K_z^\top A,
$$

where $A = (\Lambda + \sigma^2 I)^{-1}$. 
FITC approximation (3)

- FITC allows the pseudo data not being a subset of training data.
- The inducing inputs $Z$ can be optimized via gradient optimization.
- Like Nyström approximation, when taking all the training data as inducing inputs, the FITC approximation is equivalent to the original GP:

$$\tilde{p}(y|X, Z = X) = \mathcal{N}(y|0, K_{ff} + \sigma^2 I)$$

- FITC can be combined easily with expectation propagation (EP). Bui et al. [2017] provides an overview and a nice connection with variational sparse GP.
Model Approximation vs. Approximate Inference

When the exact model/inference is intractable, typically there are two types of approaches:

- Approximate the original model with a simpler one such that inference becomes tractable, like Nyström approximation, FITC.
- Keep the original model but derive an approximate inference method which is often not able to return the true answer, like variational inference.
A problem with model approximation is that

- when an approximated model requires some tuning, e.g., for hyper-parameters, it is unclear how to improve it based on training data.
- In the case of FITC, we know the model is correct if $Z = X$, however, optimizing $Z$ will not necessarily lead to a better location.
- In fact, optimizing $Z$ can lead to overfitting. [Quiñonero-Candela and Rasmussen, 2005]
Variational Sparse Gaussian Process (1)

- Titsias [2009] introduces a variational approach for sparse GP.
- It follows the same concept of pseudo data:

\[
p(y|X) = \int_{f,u} p(y|f)p(f|u,X,Z)p(u|Z)
\]

where

\[
p(u|Z) = \mathcal{N}(u|0, K_{uu}),
\]

\[
p(y|u,X,Z) = \mathcal{N}(y|K_{fu}K_{uu}^{-1}u, K_{ff} - K_{fu}K_{uu}^{-1}K_{fu}^\top + \sigma^2 I).
\]
Instead of approximate the model, Titsias [2009] derives a variational lower bound.

Normally, a variational lower bound of a marginal likelihood, also known as evidence lower bound (ELBO), looks like

\[
\log p(y|X) = \log \int_{\mathbf{f}, \mathbf{u}} p(y|\mathbf{f}) p(\mathbf{f}|\mathbf{u}, X, \mathbf{Z}) p(\mathbf{u}|\mathbf{Z}) \\
\geq \int_{\mathbf{f}, \mathbf{u}} q(\mathbf{f}, \mathbf{u}) \log \frac{p(y|\mathbf{f}) p(\mathbf{f}|\mathbf{u}, X, \mathbf{Z}) p(\mathbf{u}|\mathbf{Z})}{q(\mathbf{f}, \mathbf{u})}.
\]

Zhenwen Dai (Amazon)
Scalable Gaussian Processes
September 4, 2018 @GPSS2018
Titsias [2009] defines an unusual variational posterior:

\[ q(f, u) = p(f|u, X, Z)q(u), \quad \text{where} \quad q(u) = \mathcal{N}(u|\mu, \Sigma). \]

Plug it into the lower bound:

\[
\mathcal{L} = \int_{f,u} p(f|u, X, Z)q(u) \log \frac{p(y|f)p(f|u, X, Z)p(u|Z)}{p(f|u, X, Z)q(u)}
\]

\[
= \langle \log p(y|f) \rangle_{p(f|u, X, Z)q(u)} - \text{KL} (q(u) \parallel p(u|Z))
\]

\[
= \langle \log \mathcal{N}(y|K_{fu}K_{uu}^{-1}u, \sigma^2 I) \rangle_{q(u)} - \text{KL} (q(u) \parallel p(u|Z))
\]
There is no inversion of any big covariance matrices in the first term:

\[-\frac{N}{2} \log 2\pi \sigma^2 - \frac{1}{2\sigma^2} \langle (K_{fu}K_{uu}^{-1}u - y)^\top (K_{fu}K_{uu}^{-1}u - y) \rangle_{q(u)}\]

The overall complexity of the lower bound is \(O(NM^2)\).
Tighten the Bound

- Find the optimal parameters of $q(u)$:

$$\mu^*, \Sigma^* = \arg \max_{\mu, \Sigma} \mathcal{L}(\mu, \Sigma).$$

- Make the bound as tight as possible by plugging in $\mu^*$ and $\Sigma^*$:

$$\mathcal{L} = \log \mathcal{N}(y|0, K_{fu}K_{uu}^{-1}K_{fu}^\top + \sigma^2 I) - \frac{1}{2\sigma^2} \text{tr} \left(K_{ff} - K_{fu}K_{uu}^{-1}K_{fu}^\top \right).$$

- The overall complexity of the lower bound remains $O(NM^2)$. 
Variational sparse GP

- Note that $\mathcal{L}$ is not a valid log-pdf, $\int_y \exp(\mathcal{L}(y)) \leq 1$, due to the trace term.
- As inducing points are variational parameters, optimizing the inducing inputs $\mathbf{Z}$ always leads to a better bound.
- The model does not “overfit” with too many inducing points.
An alternative view of sparse GP

Is variational sparse GP a *hack* only working for GP?
The two key ingredients of variational sparse GP:

- Variational compression [Hensman and Lawrence, 2014]
- Variational posterior with “pseudo data” (“variational Gaussian process” by Tran et al. [2016])
A generic variational lower bound

Consider a generic probabilistic model $p(y|h)p(h|x)$ with $h$ being a latent variable. A variational lower bound is typically like

$$\log p(y|x) \geq \int_h q(h) \log \frac{p(y|h)p(h|x)}{q(h)}$$
Use prior as posterior

We are free to choose the form of the variational posterior. It is always possible to choose $q(h) = p(h|x)$. This results into a lower bound:

$$
\mathcal{L} = \int_h p(h|x) \log p(y|h).
$$

The same idea can be easily applied to a deeper model:

$$
\log p(y|x) = \log \int_{h_1, h_2, h_3} p(y|h_1)p(h_1|h_2)p(h_2|h_3)p(h_3|x) \geq \int_{h_1, h_2, h_3} p(h_1|h_2)p(h_2|h_3)p(h_3|x) \log p(y|h_1)
$$

This is weird. Does it work?
Sigmoid Belief Networks

It works surprisingly well. Dai and Lawrence [2015] applied this trick to sigmoid belief networks (SBN):

\[
p(y|h_1) \prod_{l=1}^{L-1} p(h_l|h_{l+1})p(h_L),
\]

where

\[
p(y|h_1) = \prod_i \sigma(W_{1,i}h_1 + b_{1,i})^{y_i} \sigma(-W_{1,i}h_1 - b_{1,i})^{1-y_i},
\]

\[
p(h_l|h_{l+1}) = \prod_i \sigma(W_{l+1,i}h_{l+1} + b_{l+1,i})^{h_{l,i}} \sigma(-W_{l+1,i}h_{l+1} - b_{l+1,i})^{1-h_{l,i}}
\]

\[
p(h_L) = \prod_i \pi_i^{h_Li}(1 - \pi_i)^{1-h_{Li}}
\]
Variational Inference of SBN is very hard [Mnih and Gregor, 2014]:

\[
\log p(y) \geq \sum_{h_1, \ldots, h_L} q(h_1, \ldots, h_L) \log \frac{p(y|h_1) \prod_{l=1}^{L-1} p(h_l|h_{l+1}) p(h_L)}{q(h_1, \ldots, h_L)}
\]

\[
\mathcal{L} = \sum_{h_1, \ldots, h_L} q(h_L) \prod_{l=1}^{L-1} p(h_l|h_{l+1}) \log p(y|h_1)
\]
3 hidden layers
(100-100-10)

- the generated examples from each value in the top layer
- in total 1024 examples
- columns encode first 5 bits.
- rows encode later 5 bits.
The variational lower bound can also be written as

$$
\log p(y|x) = \int_h q(h) \log \frac{p(y|h)p(h|x)}{q(h)} + \text{KL} \left( q(h) \parallel p(h|x, y) \right)
$$

With \( q(h) = p(h|x) \),

$$
\log p(y|x) = \int_h p(h|x) \log p(y|h) + \text{KL} \left( p(h|x) \parallel p(h|x, y) \right)
$$
What is the price?

The lower bound is exact only if the posterior is same as the prior.

\[
\log p(y|x) = \int p(h|x) \log p(y|h) + \text{KL} (p(h|x) \| p(h|x,y))
\]

\[
p(h|x,y) = \frac{p(y|h)p(h|x)}{\int_{h'} p(y|h')p(h'|x)}
\]

KL \( p(h|x) \| p(h|x,y) \) is zero, when

- \( y \) is independent of \( h \), i.e., \( p(y|h) = p(y) \).
- \( p(h|x) \) is a deterministic relation, i.e., \( p(h|x) = \delta(h(x)) \).
A bias towards being a deterministic function

This variational posterior introduces a bias towards being a deterministic function.

Assume the model is parameterized by $\theta$, i.e., $p(y|h, \theta)p(h|x, \theta)$.

Point Estimate:

$$\theta^* = \arg \max_{\theta} \mathcal{L}(\theta) = \arg \max_{\theta} (\log p(y|x, \theta) - \text{KL} (p(h|x, \theta) \parallel p(h|x, y, \theta)))$$
A bias towards being a deterministic function

Variational Inference:

$$
\hat{\mathcal{L}} = \int_{h,\theta} p(h|x, \theta) q(\theta) \log \frac{p(y|h, \theta)p(h|x, \theta)p(\theta)}{p(h|x, \theta)q(\theta)} \\
= \langle \log p(y|x, \theta) - \text{KL} (p(h|x, \theta) \parallel p(h|x, y, \theta)) \rangle_{q(\theta)} - \text{KL} (q(\theta) \parallel p(\theta)) \\
= \langle \log p(y|x, \theta) \rangle_{q(\theta)} - \text{KL} (q(\theta) \parallel p(\theta)) \\
- \langle \text{KL} (p(h|x, \theta) \parallel p(h|x, y, \theta)) \rangle_{q(\theta)}
$$
The bias in variational Sparse GP

Variational sparse GP often "under-fit".

\[ \mathcal{L} = \int_{\mathbf{f}, \mathbf{u}} p(\mathbf{f}|\mathbf{u}, \mathbf{X}, \mathbf{Z}) q(\mathbf{u}) \log \frac{p(\mathbf{y}|\mathbf{f}) p(\mathbf{f}|\mathbf{u}, \mathbf{X}, \mathbf{Z}) p(\mathbf{u}|\mathbf{Z})}{p(\mathbf{f}|\mathbf{u}, \mathbf{X}, \mathbf{Z}) q(\mathbf{u})} \]

\[ = \int_{\mathbf{f}, \mathbf{u}} p(\mathbf{f}|\mathbf{u}, \mathbf{X}, \mathbf{Z}) q(\mathbf{u}) \log p(\mathbf{y}|\mathbf{f}) - \text{KL} \left( q(\mathbf{u}) \parallel p(\mathbf{u}|\mathbf{Z}) \right) \]

\[ = \langle \log p(\mathbf{y}|\mathbf{X}, \mathbf{u}, \mathbf{Z}) \rangle_{q(\mathbf{u})} - \text{KL} \left( q(\mathbf{u}) \parallel p(\mathbf{u}|\mathbf{Z}) \right) \]

\[ - \langle \text{KL} \left( p(\mathbf{f}|\mathbf{X}, \mathbf{u}, \mathbf{Z}) \parallel p(\mathbf{f}|\mathbf{X}, \mathbf{y}, \mathbf{u}, \mathbf{Z}) \right) \rangle_{q(\mathbf{u})} \]
Flexible variational posterior

Consider a generic probabilistic model, e.g., \( p(y|f)p(f) \).

Variational lower bound:

\[
\mathcal{L} = \int q(f) \log \frac{p(y|f)p(f)}{q(f)}
\]

One flexible variational posterior:

\[
q(f) = \int q(f|u)q(u)
\]
It may not be tractable to compute $\int f q(f|u)q(u)$. One way to work around is to introduce $u$ as an auxiliary variable to the model:

$$p(y|f)p(f)p(u|f)$$

Note that we are free to choose the form of $p(u|f)$ and $q(u)$ and $q(f|u)$. 
A further lower bound

show the relation between two lower bounds.

\[
\int f q(f) \log \frac{p(y|f)p(f)}{q(f)} = \int_{f,u} q(u|f)q(f) \log \frac{p(y|f)p(f)p(u|f)}{q(u|f)q(f)} - \int f q(f) \int_u q(u|f) \log \frac{p(u|f)}{q(u|f)}
\]

\[
\mathcal{L} = \mathcal{L}_{f,u} - \langle \text{KL} (q(u|f) \| p(u|f)) \rangle_{q(f)} \geq \mathcal{L}_{f,u}
\]
In the case of spare GP

This leads back to the usual sparse GP bound that we know.

\[
\begin{align*}
p(y|f) &= \mathcal{N}(y|f, \sigma^2 I) \\
p(f|X) &= \mathcal{N}(f|0, K(X, X)) \\
p(u|Z, f, X) &= \mathcal{N}
\left(u|K_{uf}K_{ff}^{-1}f, K_{uu} - K_{fu}K_{ff}^{-1}K_{fu}\right) \\
q(u|f)q(f) &= q(f|u)q(u) = p(f|X, u, Z)q(u)
\end{align*}
\]
Parallel Sparse Gaussian Process

- Beyond Approximate the inference method, maybe we could exploit parallelization.
- For Gaussian process, it turns out to be very hard, because parallel Cholesky decomposition is very difficult.
- Dai et al. [2014] and Gal et al. [2014] proposes a parallel inference method for sparse GP.
Data Parallelism

- Consider a training set: \( \mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\} \).
- Assume there are \( C \) computational cores/machines.
- A data parallelism algorithm divides the data set into \( C \) partitions as evenly as possible: \( \mathcal{D} = \bigcup_{c=1}^{C} \mathcal{D}_c \).
- The parallelism happens in the way that the function running on each core only requiring the data from the local partition.
A simple example: neural network regression

\[ l = \sum_{n=1}^{N} \| y_n - f_\theta(x_n) \|^2 = \sum_{c=1}^{C} \sum_{n_c \in D_c} \| y_{n_c} - f_\theta(x_{n_c}) \|^2 \]

1. Each core computes its local objective
   \[ l_c = \sum_{n_c \in D_c} \| y_{n_c} - f_\theta(x_{n_c}) \|^2. \]

2. Each core computes the gradient of its local object \( \partial l_c/\partial \theta \).

3. Aggregate all the local objectives and gradients \( l = \sum_{c=1}^{C} l_c \) and \( \partial l/\partial \theta = \sum_{c=1}^{C} \partial l_c/\partial \theta \).

4. Take a step along the gradient following a gradient descent algorithm.

5. Repeat Step 1 until converge.
Data Parallelism for Sparse GP

The variational lower bound (after applying Woodbury formula) is

\[
\mathcal{L} = - \frac{N}{2} \log 2\pi \sigma^2 + \frac{1}{2} \log \frac{|K_{uu}|}{|K_{uu} + \sigma^{-2} \Phi|} - \frac{1}{2\sigma^2} y^\top y
+ \frac{1}{2\sigma^4} y^\top K_{fu} (K_{uu} + \Phi)^{-1} K_{fu}^\top y - \frac{1}{2\sigma^2} \phi + \frac{1}{2\sigma^2} \text{tr} (K_{uu}^{-1} \Phi)
\]

where \( \Phi = K_{fu}^\top K_{fu} \) and \( \phi = \text{tr} (K_{ff}) \).
Data Parallelism for Sparse GP

- The lower bound is not fully distributable like in the simple example.
- All the terms involving data can be written as a sum across data points:

\[
\begin{align*}
\mathbf{y}^\top \mathbf{y} &= \sum_{n=1}^{N} y_n^2, \\
\mathbf{y}^\top \mathbf{K}_{fu} &= \sum_{n=1}^{N} y_n \mathbf{K}_{fnu}, \\
\Phi &= \sum_{n=1}^{N} \mathbf{K}_{fnu}^\top \mathbf{K}_{fnu} \\
\phi &= \sum_{n=1}^{N} \mathbf{K}_{fnf_n}, \text{ where } \mathbf{K}_{fu} = \mathbf{K}(\mathbf{x}_n, Z), \quad \mathbf{K}_{fnf_n} = \mathbf{K}(\mathbf{x}_n, \mathbf{x}_n).
\end{align*}
\]
Data Parallelism for Sparse GP

1. [local] Compute all the data related terms locally: \( \mathbf{y}_c^\top \mathbf{y}_c, \mathbf{y}_c^\top \mathbf{K}_{f_c u}, \Phi_c \) and \( \phi_c \).

2. [global] Aggregate all the local terms and compute the lower bound \( \mathcal{L} \) on one node.

3. [global] Compute the gradient of the bound w.r.t. the model parameters.

4. [global] Compute the gradient w.r.t. the local terms \( \partial \mathcal{L} / \partial \mathbf{K}_{f_c u}, \partial \mathcal{L} / \partial \Phi_c \) and \( \partial \mathcal{L} / \partial \phi_c \) and broadcast to individual nodes.

5. [local] Compute the gradient contribution of the local terms and aggregate the local gradients into the final gradient.

Data Parallelism for Sparse GP

Zhenwen Dai (Amazon)
Scalable Gaussian Processes
September 4, 2018 @GPSS2018

The graphs illustrate the relationship between the number of data points and the average time per iteration, as well as the percentage of indistributable computational time, across different numbers of CPU cores and GPUs.
The emerge of deep learning platforms

- Deep learning platforms such as Theano, Tensorflow, Torch, Caffe, MXNet emerge in recent years.
- It standardizes deep neural networks programming.
- Auto-differentiation enables the flexible construction of DNNs.
- GPU acceleration enables scalability for real world applications.
GPU for machine learning

- Von Neumann architecture is not suitable for machine learning.
- Memory bandwidth:
  - GPU (NVidia V100, AWS P3): 900 GB/s
  - CPU (Intel Xeon E5-2660 v3, AWS C4): 68 GB/s

GTX 580 GPU has only 3GB of memory
Probabilistic Programming on deep learning platforms

- Edward http://edwardlib.org
- PyMC3 https://github.com/pymc-devs/pymc3
- pyprob https://github.com/probprog/pyprob
- Pyro https://github.com/uber/pyro
GP on deep learning platforms

GPflow https://github.com/GPflow/GPflow

GPyTorch https://github.com/cornellius-gp/gpytorch
MXFusion and GPy2

- Beyond GPU acceleration and auto-differentiation
- Use Gaussian process as a building block.
- MXFusion: modular probabilistic programming language
  https://github.com/amzn/MXFusion
- GPy2 (a new interface based on MXFusion):
  - writing new kernel with auto-differentiation
  - scalable inference on GPU
  - Construct hybrid GP, deep GP, recurrent GP by re-using GP module with scalable approximate inference.
Acknowledgement.


