Scalability of Gaussian Process

Zhenwen Dai

Spotify

14 September 2021 @GPSS 2021

Outline

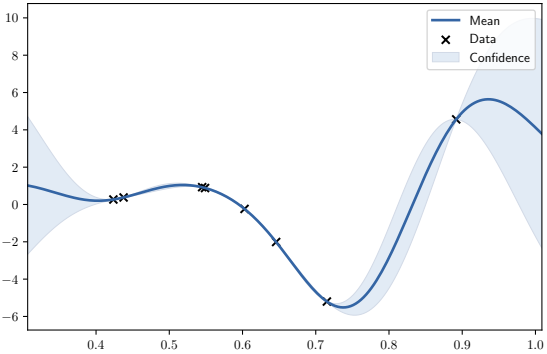
- What is the scalability issue of Gaussian Process?
- Numerical solution
- Model/Inference Approximation
- Mini-batch Training
- How to draw a function sample?

Gaussian Process Regression

Input and Output Data:

$$\mathbf{y} = (y_1, \ldots, y_N), \quad \mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^\top$$

$$p(\mathbf{y}|\mathbf{f}) = \mathcal{N}\left(\mathbf{y}|\mathbf{f}, \sigma^2\mathbf{I}\right), \quad p(\mathbf{f}|\mathbf{X}) = \mathcal{N}\left(\mathbf{f}|0, \mathbf{K}(\mathbf{X}, \mathbf{X})\right)$$



Behind a Gaussian process fit

- Maximum likelihood estimate of the hyper-parameters.

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

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

$$p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{y}, \mathbf{X}, \theta) = \mathcal{N}\left(\mathbf{f}_*|\mathbf{K}_*(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y}, \mathbf{K}_{**} - \mathbf{K}_*(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{K}_*^\top\right)$$

How to implement the log-likelihood (1)

- Compute the covariance matrix $\mathbf{K}$:

$$\mathbf{K} = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & \cdots & k(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix}$$

  where $k(\mathbf{x}_i, \mathbf{x}_j) = \gamma \exp\left(-\frac{1}{2l^2}(\mathbf{x}_i - \mathbf{x}_j)^\top (\mathbf{x}_i - \mathbf{x}_j)\right)$

- The complexity is $O(N^2 Q)$.

How to implement the log-likelihood (2)

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

$$\begin{aligned}\log p(\mathbf{y}|\mathbf{X}) &= \log \mathcal{N}\left(\mathbf{y}|0, \mathbf{K}+\sigma^2\mathbf{I}\right) \\ &= -\frac{1}{2}\log|2\pi(\mathbf{K}+\sigma^2\mathbf{I})| - \frac{1}{2}\mathbf{y}^\top(\mathbf{K}+\sigma^2\mathbf{I})^{-1}\mathbf{y} \\ &= -\frac{N}{2}N\log 2\pi - \sum_i \log \mathbf{L}_{ii} - \frac{1}{2}||\mathbf{L}^{-1}\mathbf{y}||^2\end{aligned}$$

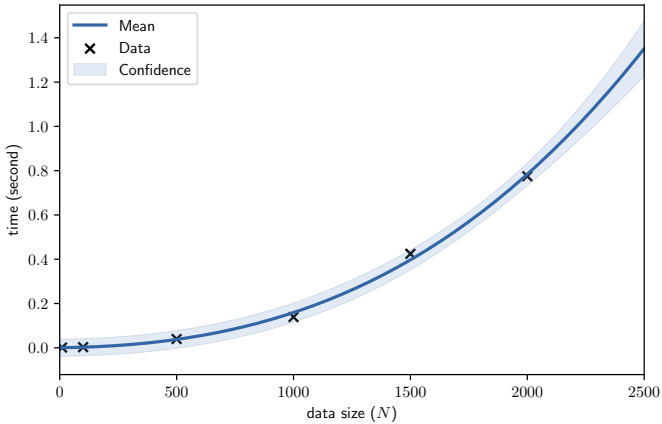
- Take a Cholesky decomposition: $\mathbf{L} = \texttt{chol}(\mathbf{K}+\sigma^2\mathbf{I})$, such that $\mathbf{K}+\sigma^2\mathbf{I} = \mathbf{L}\mathbf{L}^\top$.
- The computational complexity is $O(N^3+N^2+N)$. Therefore, the overall complexity including the computation of $\mathbf{K}$ is $O(N^3)$.

A quick profiling (N=1000, Q=10)

```
Line #    Time(ms)   % Time      Line Contents
 2                            def log_likelihood(kern, X, Y, sigma2):
 3          6.0       0.0          N = X.shape[0]
 4      55595.0      58.7          K = kern.K(X)
 5       4369.0       4.6          Ky = K + np.eye(N)*sigma2
 6      30012.0      31.7          L = np.linalg.cholesky(Ky)
 7       4361.0       4.6          LinvY = dtrtrs(L, Y, lower=1)[0]
 8         49.0       0.1          logL = N*np.log(2*np.pi)/-2.
 9         82.0       0.1          logL += np.square(LinvY).sum()/-2.
10        208.0       0.2          logL += -np.log(np.diag(L)).sum()
11          2.0       0.0          return logL
```

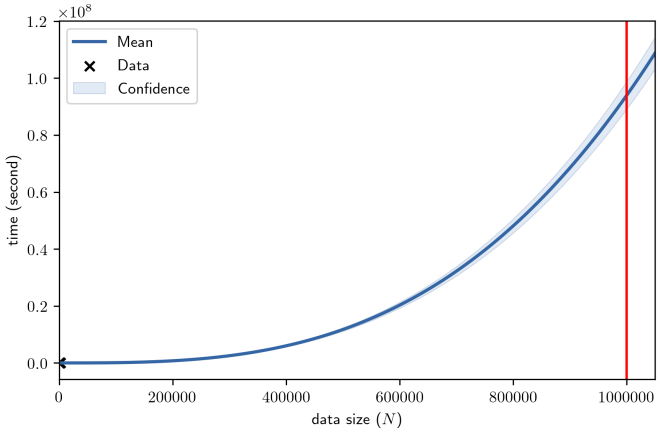
Empirical analysis of computational time

- I collect the run time for $N = \{10, 100, 500, 1000, 1500, 2000\}$.
- They take 1.3ms, 8.5ms, 28ms, 0.12s, 0.29s, 0.76s.



What if we have 1 million data points?

The mean of predicted computational time is 9.4×10^7 seconds ≈ 2.98 years.



Well, it is only a matrix inversion.

- The cubic complexity $O(N^3)$ only comes from $\mathbf{y}^\top(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y}$.
- There must be some *Numerical Linear Algebra* algorithms to speed it up!?

Quadratic Optimization Formulation

- Consider the problem:
$$\mathbf{v} = \hat{\mathbf{K}}^{-1}\mathbf{y}, \quad \hat{\mathbf{K}} = \mathbf{K} + \sigma^2\mathbf{I}$$
- Rewrite it as a linear system:
$$\hat{\mathbf{K}}\mathbf{v} - \mathbf{y} = 0$$
- This can be formulated as a quadratic optimization:
$$\mathbf{v}^* = \arg\min_{\mathbf{v}} \mathbf{v}^\top \hat{\mathbf{K}}\mathbf{v} - \mathbf{v}^\top \mathbf{y}$$

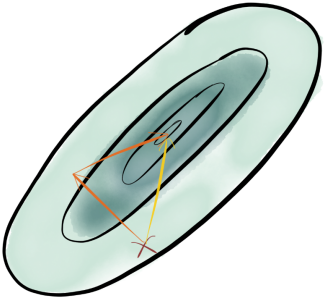
Conjugate Gradient Method (1)

- Conjugate Gradient (CG) method is an efficient solver for the quadratic problem:

$$\mathbf{v}^* = \arg\min_{\mathbf{v}} \mathbf{v}^\top \hat{\mathbf{K}} \mathbf{v} - \mathbf{v}^\top \mathbf{y}$$

- Solve it by finding n linearly independent vectors $\{\mathbf{d}_1, \mathbf{d}_N\}$ such that:

$$\mathbf{v}^* = \mathbf{v}_0 + \alpha_1 \mathbf{d}_1 + \ldots + \alpha_N \mathbf{d}_N$$



Conjugate Gradient (CG)

Figure taken from [Davies, 2015]

Conjugate Gradient Method (2)

- CG is an iterative algorithm.
- CG recovers the exact solution after N iterations.
- We get an approximate solution with #iterations $<< N$.
- Each iteration is $O(N^2)$.

Conjugate Gradient:

$$\mathbf{d}_0 = \mathbf{u}_0 = \mathbf{y} - \hat{\mathbf{K}}\mathbf{v}_0$$

$$\alpha_i = \frac{\mathbf{u}_i^\top \mathbf{u}}{\mathbf{d}_i^\top \hat{\mathbf{K}} \mathbf{d}_i}$$

$$\mathbf{v}_{i+1} = \mathbf{v}_i + \alpha_i \mathbf{d}_i$$

$$\mathbf{u}_{i+1} = \mathbf{u}_i - \alpha_i \hat{\mathbf{K}} \mathbf{d}_i$$

$$\beta_{i+1} = \frac{\mathbf{u}_{i+1}^\top \mathbf{u}_{i+1}}{\mathbf{u}_i^\top \mathbf{u}_i}$$

$$\mathbf{d}_{i+1} = \mathbf{u}_{i+1} + \beta_{i+1} \mathbf{d}_i$$

Convergence and Preconditioning

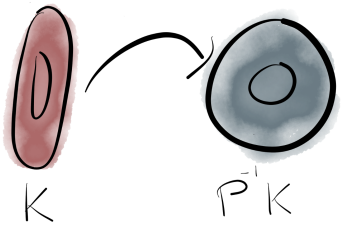
- Numerical stability and rate of convergence of CG are *sensitive* to the condition number:

$$\kappa(\hat{\mathbf{K}}) = \frac{\lambda_{\max}(\hat{\mathbf{K}})}{\lambda_{\min}(\hat{\mathbf{K}})}$$

- Improve the condition number by solving:

$$\mathbf{P}^{-1}\hat{\mathbf{K}}\mathbf{v} - \mathbf{P}^{-1}\mathbf{y} = 0$$

- Ideally $\mathbf{P}^{-1} = \hat{\mathbf{K}}^{-1}$ so that $\kappa(\mathbf{P}^{-1}\hat{\mathbf{K}}) = 1$.

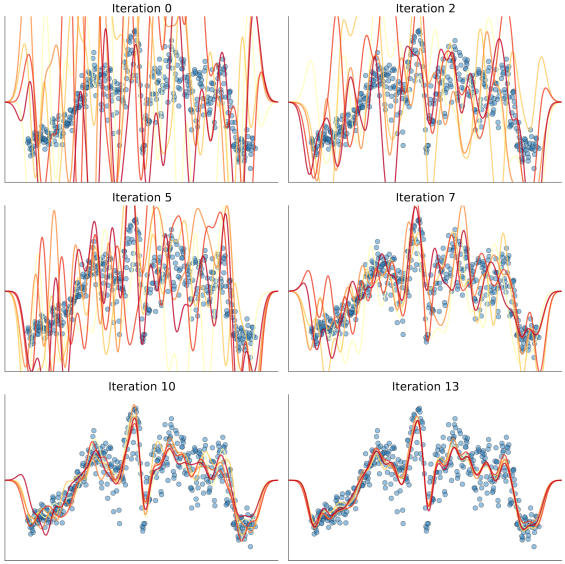


Preconditioning

Figure taken from [Davies, 2015]

Example of CG

- Example from [Davies, 2015].
- Estimate the posterior mean of GP.
- 5 separate runs ($N = 415$)
- CG is used in GPyTorch [Gardner et al., 2018].

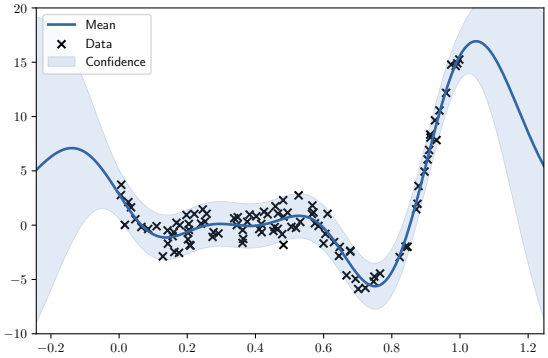


$O(N^2)$ is still slow!

Gaussian Process Model/Inference Approximation

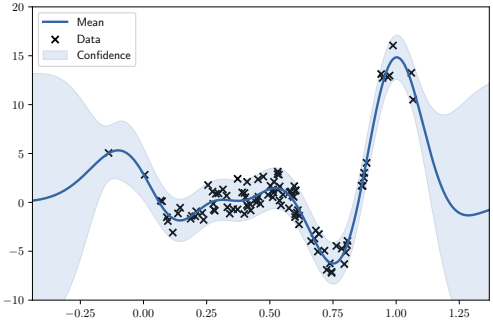
Big data (?)

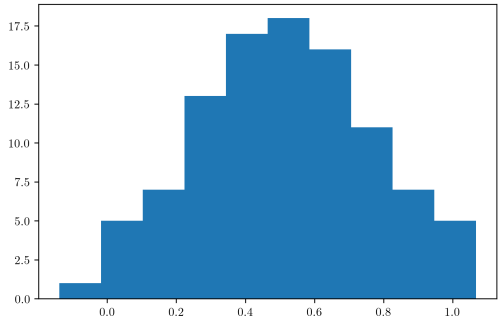
- lots of data $\neq$ complex function
- In real world problems, we often collect a lot of data for modeling relatively simple relations.



Data subsampling?

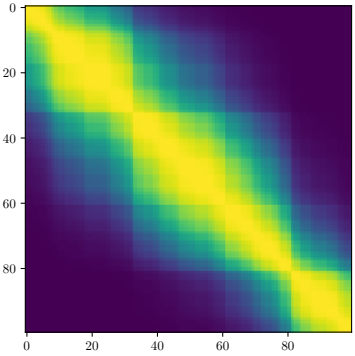
- Real data often do not evenly distributed.
- We tend to get a lot of data on common cases and very few data on rare cases.

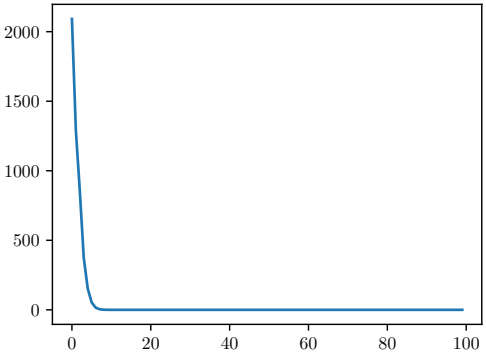




Covariance matrix of redundant data

- With redundant data, the covariance matrix becomes low rank.
- What about low rank approximation?





Low-rank approximation

- Let's recall the log-likelihood of GP:

$$\log p(\mathbf{y}|\mathbf{X}) = \log \mathcal{N}\left(\mathbf{y}|0, \mathbf{K} + \sigma^2\mathbf{I}\right),$$

  where $\mathbf{K}$ is the covariance matrix computed from $\mathbf{X}$ according to the kernel function $k(\cdot, \cdot)$ and σ^2 is the variance of the Gaussian noise distribution.
- Assume $\mathbf{K}$ to be low rank.
- This leads to Nyström approximation by Williams and Seeger [Williams and Seeger, 2001].

Approximation by subset

- Let's randomly pick a subset from the training data: $\mathbf{Z} \in \mathbb{R}^{M \times Q}$.
- Approximate the covariance matrix $\mathbf{K}$ by $\tilde{\mathbf{K}}$.

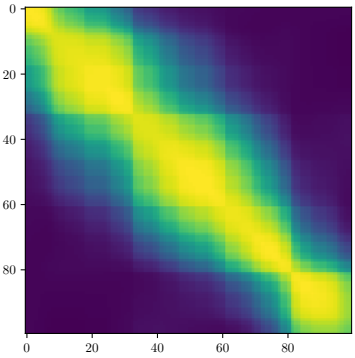
$$\tilde{\mathbf{K}} = \mathbf{K}_z \mathbf{K}_{zz}^{-1} \mathbf{K}_z^\top, \text{ where } \mathbf{K}_z = \mathbf{K}(\mathbf{X}, \mathbf{Z}) \text{ and } \mathbf{K}_{zz} = \mathbf{K}(\mathbf{Z}, \mathbf{Z}).$$

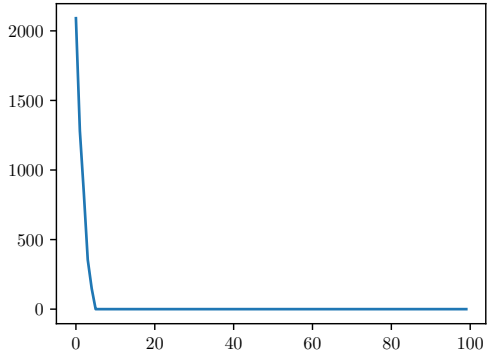
- Note that $\tilde{\mathbf{K}} \in \mathbb{R}^{N \times N}$, $\mathbf{K}_z \in \mathbb{R}^{N \times M}$ and $\mathbf{K}_{zz} \in \mathbb{R}^{M \times M}$.
- The log-likelihood is approximated by

$$\log p(\mathbf{y}|\mathbf{X}, \theta) \approx \log \mathcal{N}\left(\mathbf{y}|0, \mathbf{K}_z \mathbf{K}_{zz}^{-1} \mathbf{K}_z^\top + \sigma^2 \mathbf{I}\right).$$

Nyström approximation example

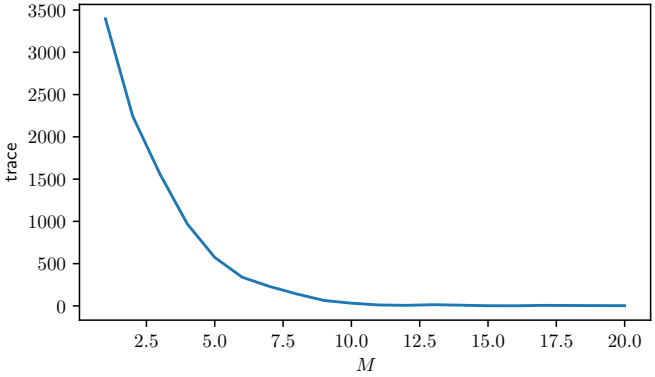
The covariance matrix with Nyström approximation using 5 random data points:





Nyström approximation example

Compute $\text{tr}\left(\mathbf{K} - \tilde{\mathbf{K}}\right)$ with different M.



Efficient computation using Woodbury formula

- The naive formulation does not bring any computational benefits.

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

- Apply the Woodbury formula:

$$(\mathbf{K}_z\mathbf{K}_{zz}^{-1}\mathbf{K}_z^\top + \sigma^2\mathbf{I})^{-1} = \sigma^{-2}\mathbf{I} - \sigma^{-4}\mathbf{K}_z(\mathbf{K}_{zz} + \sigma^{-2}\mathbf{K}_z^\top\mathbf{K}_z)^{-1}\mathbf{K}_z^\top$$

- Note that $(\mathbf{K}_{zz} + \sigma^{-2}\mathbf{K}_z^\top\mathbf{K}_z) \in \mathbb{R}^{M\times M}$.
- The computational complexity reduces to $O(NM^2)$.

Nyström approximation

- 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.*, $\mathbf{K}\mathbf{K}^{-1}\mathbf{K}^{\top} = \mathbf{K}$.
- The subset selection is done randomly.

Gaussian process with Pseudo Data (1)

- Snelson and Ghahramani [2006] proposes the idea of having pseudo data, which is later referred to as Fully independent training conditional (FITC).
- Augment the training data $(\mathbf{X}, \mathbf{y})$ with pseudo data $\mathbf{u}$ at location $\mathbf{Z}$.

$$p\left(\begin{bmatrix}\mathbf{y}\\ \mathbf{u}\end{bmatrix} \mid \begin{bmatrix}\mathbf{X}\\ \mathbf{Z}\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\mathbf{y}\\ \mathbf{u}\end{bmatrix} | 0, \begin{bmatrix}\mathbf{K}_{ff} + \sigma^2\mathbf{I} & \mathbf{K}_{fu}\\ \mathbf{K}_{fu}^\top & \mathbf{K}_{uu}\end{bmatrix}\right)$$

where $\mathbf{K}_{ff} = \mathbf{K}(\mathbf{X}, \mathbf{X})$, $\mathbf{K}_{fu} = \mathbf{K}(\mathbf{X}, \mathbf{Z})$ and $\mathbf{K}_{uu} = \mathbf{K}(\mathbf{Z}, \mathbf{Z})$.

Gaussian process with Pseudo Data (2)

- Thanks to the marginalization property of Gaussian distribution,

$$p(\mathbf{y}|\mathbf{X}) = \int_{\mathbf{u}} p(\mathbf{y}, \mathbf{u}|\mathbf{X}, \mathbf{Z}).$$

- Further re-arrange the notation:

$$p(\mathbf{y}, \mathbf{u}|\mathbf{X}, \mathbf{Z}) = p(\mathbf{y}|\mathbf{u}, \mathbf{X}, \mathbf{Z})p(\mathbf{u}|\mathbf{Z})$$

where $p(\mathbf{u}|\mathbf{Z}) = \mathcal{N}(\mathbf{u}|0, \mathbf{K}_{uu})$,
$p(\mathbf{y}|\mathbf{u}, \mathbf{X}, \mathbf{Z}) = \mathcal{N}\left(\mathbf{y}|\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{u}, \mathbf{K}_{ff} - \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{fu}^{\top} + \sigma^2\mathbf{I}\right)$.

FITC approximation (1)

- So far, $p(\mathbf{y}|\mathbf{X})$ has not been changed, but there is no speed-up.
- $\mathbf{K}_{ff} \in \mathbb{R}^{N \times N}$ in $\mathbf{K}_{ff} - \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{fu}^{\top} + \sigma^2\mathbf{I}$.
- The FITC approximation assumes

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

where $\mathbf{\Lambda} = (\mathbf{K}_{ff} - \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{fu}^{\top}) \circ \mathbf{I}$.

FITC approximation (2)

- Marginalize $\mathbf{u}$ from the model definition:

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

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

$$(\mathbf{K}_z\mathbf{K}_{zz}^{-1}\mathbf{K}_z^{\top} + \mathbf{\Lambda} + \sigma^2\mathbf{I})^{-1} = \mathbf{A} - \mathbf{A}\mathbf{K}_z(\mathbf{K}_{zz} + \mathbf{K}_z^{\top}\mathbf{A}\mathbf{K}_z)^{-1}\mathbf{K}_z^{\top}\mathbf{A},$$

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

FITC approximation (3)

- FITC allows the pseudo data not being a subset of training data.
- The inducing inputs $\mathbf{Z}$ 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:

$$\tilde{p}(\mathbf{y}|\mathbf{X}, \mathbf{Z} = \mathbf{X}) = \mathcal{N}\left(\mathbf{y}|0, \mathbf{K}_{ff} + \sigma^2\mathbf{I}\right)$$

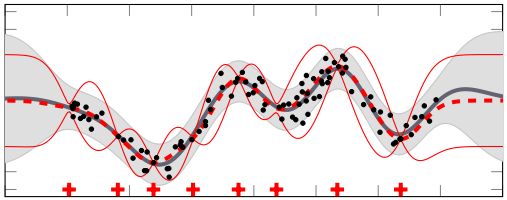
- FITC can be combined easily with expectation propagation (EP).
- Bui et al. [2017] provides an overview and a nice connection with variational sparse GP.

Model Approximation vs. Approximate Inference

FITC approximation changes the model definition.

- A better objective under FITC does not necessarily corresponds to a better approximation to the original GP.
- In fact, optimizing $\mathbf{Z}$ can lead to overfitting. [Quiñonero-Candela and Rasmussen, 2005, Bauer et al., 2016]

FITC (nlml $= 23.16$, $\sigma_n = 1.93 \cdot 10^{-4}$)



Optimal values for the exact GP: nlml $= 34.15$, $\sigma = 0.274$. [Bauer et al., 2016]

Model Approximation vs. Approximate Inference

Variational inference (VI) takes a different approach.

- VI keeps the model definition untouched.
- VI derives a lower bound of the log-marginal likelihood:

$$\log(y) \geq \int q(x) \log \frac{p(y,x)}{q(x)} \mathrm{d}x = \mathcal{L}$$

- Alternatively, it can be written as

$$\mathrm{KL}\left(q(x) \,\|\, p(x|y)\right) = \log p(y) - \mathcal{L}.$$

Variational Sparse Gaussian Process (1)

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

$$p(\mathbf{y}|\mathbf{X}) = \int_{\mathbf{f},\mathbf{u}} p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z})p(\mathbf{u}|\mathbf{Z})$$

where $p(\mathbf{u}|\mathbf{Z}) = \mathcal{N}(\mathbf{u}|0,\mathbf{K}_{uu})$,
$p(\mathbf{y}|\mathbf{u},\mathbf{X},\mathbf{Z}) = \mathcal{N}\left(\mathbf{y}|\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{u}, \mathbf{K}_{ff} - \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{fu}^{\top} + \sigma^2\mathbf{I}\right)$.

Variational Sparse Gaussian Process (2)

- Instead of approximate the model, Titsias [2009] derives a variational lower bound.
- Normally, a variational lower bound of a marginal likelihood looks like

$$\begin{aligned}\log p(\mathbf{y}|\mathbf{X}) &= \log \int_{\mathbf{f},\mathbf{u}} p(\mathbf{y}|\mathbf{f}) p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z}) p(\mathbf{u}|\mathbf{Z}) \\ &\geq \int_{\mathbf{f},\mathbf{u}} q(\mathbf{f},\mathbf{u}) \log \frac{p(\mathbf{y}|\mathbf{f}) p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z}) p(\mathbf{u}|\mathbf{Z})}{q(\mathbf{f},\mathbf{u})}.\end{aligned}$$

Special Variational Posterior

- Titsias [2009] defines an unusual variational posterior:

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

- Plug it into the lower bound:

$$\begin{aligned}
\mathcal{L} &= \int_{\mathbf{f},\mathbf{u}} p(\mathbf{f}|\mathbf{u}, \mathbf{X}, \mathbf{Z})q(\mathbf{u}) \log \frac{p(\mathbf{y}|\mathbf{f})\cancel{p(\mathbf{f}|\mathbf{u}, \mathbf{X}, \mathbf{Z})}p(\mathbf{u}|\mathbf{Z})}{\cancel{p(\mathbf{f}|\mathbf{u}, \mathbf{X}, \mathbf{Z})}q(\mathbf{u})} \\
&= \langle \log p(\mathbf{y}|\mathbf{f}) \rangle_{p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z})q(\mathbf{u})} - \mathsf{KL}\left(q(\mathbf{u}) \,\|\, p(\mathbf{u}|\mathbf{Z})\right) \\
&= \left\langle \log \mathcal{N}\left(\mathbf{y}|\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{u}, \sigma^2\mathbf{I}\right) \right\rangle_{q(\mathbf{u})} - \mathsf{KL}\left(q(\mathbf{u}) \,\|\, p(\mathbf{u}|\mathbf{Z})\right)
\end{aligned}$$

Special Variational Posterior

- There is no inversion of any big covariance matrices in the first term:

$$-\frac{N}{2}\log 2\pi\sigma^2 - \frac{1}{2\sigma^2}\left\langle(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{u}-\mathbf{y})^\top(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{u}-\mathbf{y})\right\rangle_{q(\mathbf{u})}$$

- The overall complexity of the lower bound is $O(NM^2)$.

Tighten the Bound

- Find the optimal parameters of $q(\mathbf{u})$:

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

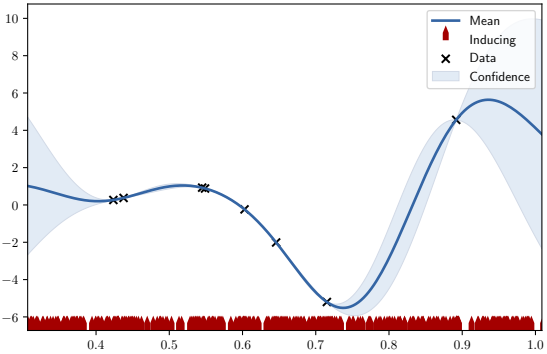
- Make the bound as tight as possible by plugging in μ^* and Σ^*:

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

- The 1st term is the same as in the Nyström approximation.
- The overall complexity of the lower bound remains $O(NM^2)$.

Variational sparse GP

- Note that $\mathcal{L}$ is not a valid log-pdf, $\int_{\mathbf{y}} \exp(\mathcal{L}(\mathbf{y})) \leq 1$, 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.

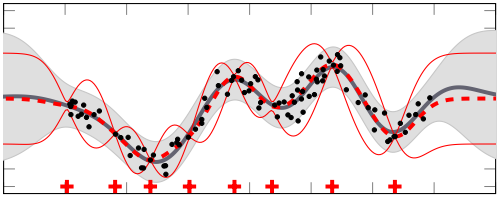


FITC vs. Variational sparse GP

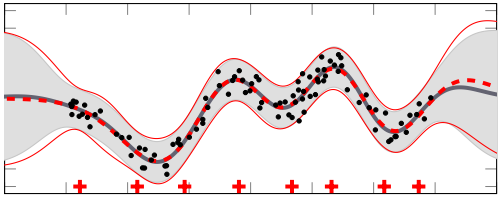
- model approximation vs. approximate inference (see [Bauer et al., 2016])
- Note that, when point estimating hyper-parameters, if the number of inducing points is too small, the model may "under-fit":

$$\mathcal{L} = \log p(y) - \mathsf{KL}\left(q(x) \,\|\, p(x|y)\right).$$

FITC (nlml $= 23.16$, $\sigma_n = 1.93 \cdot 10^{-4}$)



VFE (nlml $= 38.86$, $\sigma_n = 0.286$)



Optimal values for the exact GP: nlml $= 34.15$, $\sigma = 0.274$. [Bauer et al., 2016]

Limitations of Sparse GP

Variational sparse GP has computational complexity $O(NM^2)$.

The computation becomes infeasible under two scenarios:

- The number of data points N is very high, e.g., millions of data points.
- The function is very complex, which requires tens of thousands of inducing points.

Mini-batch Learning (1)

- Mini-batch learning allows DNNs to be trained on millions of data points.
- Given a set of inputs and labels, $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, $(\mathbf{x}_i, y_i) \sim p(\mathbf{x}, y)$, the true loss function is defined as

$$c_{\text{true}} = \int l(f_\theta(\mathbf{x}), y) p(\mathbf{x}, y) \mathrm{d}\mathbf{x} \mathrm{d}y \approx \frac{1}{N} \sum_{i=1}^{N} l(f_\theta(\mathbf{x}), y) = c,$$

  where $f_\theta(\cdot)$ is DNN and $l(\cdot, \cdot)$ is the loss function.
- Gradient descent (GD) updates the parameters by

$$\theta_{t+1} = \theta_t - \eta \frac{\mathrm{d}c}{\mathrm{d}\theta}.$$

Mini-batch Learning (2)

- Mini-batch learning approximates the loss by subsampling the data,

$$c_{\mathsf{MB}} = \frac{1}{B} \sum_{\mathbf{x}_i, y_i \sim \tilde{p}(\mathbf{x}, y)} l(f_\theta(\mathbf{x}_i), y_i).$$

- Stochastic gradient descent (SGD) updates the parameters by

$$\theta_{t+1} = \theta_t - \eta \frac{\mathsf{d}c_{\mathsf{MB}}}{\mathsf{d}\theta}.$$

- Can mini-batch learning be applied to GPs as well?

Mini-batch Learning for GPs

- Mini-batch learning relies on the objective being an expectation w.r.t. the data, *i.e.*, $\langle l(f_\theta(\mathbf{x}), y) \rangle_{p(\mathbf{x},y)}$.
- The log-marginal likelihood of GP:

$$\log \mathcal{N}\left(\mathbf{y}|0, \mathbf{K} + \sigma^2 \mathbf{I}\right)$$

- The variational lower bound of sparse GP:

$$\log \mathcal{N}\left(\mathbf{y}|0, \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{fu}^\top + \sigma^2 \mathbf{I}\right) - \frac{1}{2\sigma^2}\mathrm{tr}\left(\mathbf{K}_{ff} - \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{fu}^\top\right)$$

"Uncollapsed" Lower Bound

- Hensman et al. [2013] discovers that the "uncollapsed" variational lower bound of sparse GP can be used for mini-batch learning.
- The "uncollapsed" variational lower bound of sparse GP:

$$\mathcal{L} = \langle \log p(\mathbf{y}|\mathbf{f}) \rangle_{p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z})q(\mathbf{u})} - \mathrm{KL}\left(q(\mathbf{u}) \,\|\, p(\mathbf{u})\right)$$

- The 2nd term, $\mathrm{KL}\left(q(\mathbf{u}) \,\|\, p(\mathbf{u})\right)$, does not depend on the data.

"Uncollapsed" Lower Bound

- In the 1st term, as $p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{y}|\mathbf{f}, \sigma^2\mathbf{I})$,

$$\log p(\mathbf{y}|\mathbf{f}) = \sum_{n=1}^{N} \log \mathcal{N}\left(y_n|f_n, \sigma^2\right)$$

- Denote $q(\mathbf{f}|\mathbf{X}, \mathbf{Z}) = \int p(\mathbf{f}|\mathbf{u}, \mathbf{X}, \mathbf{Z}) q(\mathbf{u}) \mathrm{d}\mathbf{u}$.

$$\begin{aligned}\langle \log p(\mathbf{y}|\mathbf{f}) \rangle_{q(\mathbf{f}|\mathbf{X},\mathbf{Z})} &= \left\langle \sum_{n=1}^{N} \log \mathcal{N}\left(y_n|f_n, \sigma^2\right) \right\rangle_{q(\mathbf{f}|\mathbf{X},\mathbf{Z})} \\ &= \sum_{n=1}^{N} \left\langle \log \mathcal{N}\left(y_n|f_n, \sigma^2\right) \right\rangle_{q(f_n|\mathbf{x}_n,\mathbf{Z})}\end{aligned}$$

Stochastic Variational GP (SVGP)

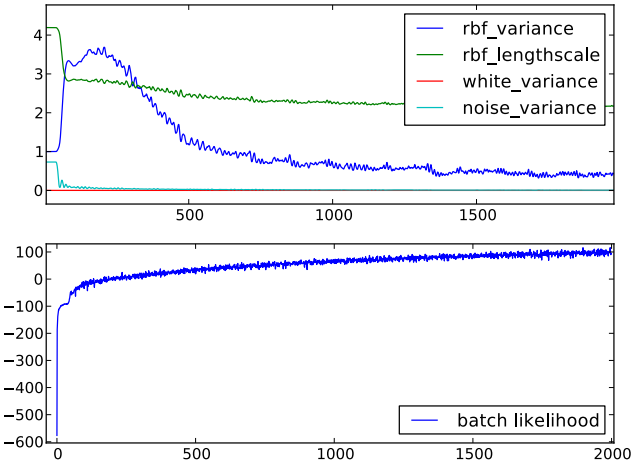
- The resulting lower bound can be written as the sum over the data,

$$\begin{aligned}\mathcal{L} =& \sum_{n=1}^{N} \left\langle \log \mathcal{N}\left(y_n | f_n, \sigma^2\right) \right\rangle_{q(f_n|\mathbf{x}_n,\mathbf{Z})} - \mathsf{KL}\left(q(\mathbf{u}) \,\|\, p(\mathbf{u})\right) \\ \approx& \frac{N}{B} \sum_{\mathbf{x}_i, y_i \sim \tilde{p}(\mathbf{x},y)} \left\langle \log \mathcal{N}\left(y_i | f_i, \sigma^2\right) \right\rangle_{q(f_i|\mathbf{x}_i,\mathbf{Z})} - \frac{N}{B}\mathsf{KL}\left(q(\mathbf{u}) \,\|\, p(\mathbf{u})\right) = \mathcal{L}_{\mathsf{MB}}\end{aligned}$$

- This allows us to do mini-batch learning with SGD,

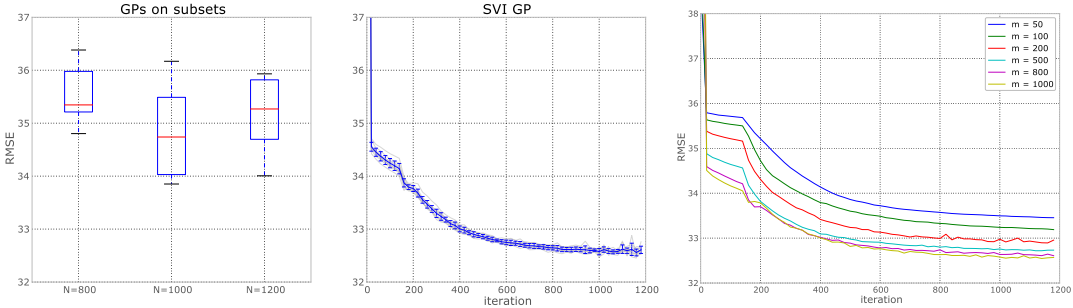
$$\theta_{t+1} = \theta_t - \eta \frac{\mathsf{d}\mathcal{L}_{\mathsf{MB}}}{\mathsf{d}\theta}.$$

2D Synthetic Data



Airline Delay Data

Flight delays for every commercial flight in the USA from January to April 2008.
700,000 train, 100,000 test



The pros and cons of SVGP

Pros

- With mini-batch learning, the computational complexity reduces from $O(NM^2)$ to $O(M^3)$.

Cons

- The variational distribution $q(\mathbf{u})$ needs to be explicitly optimized.
- The number of variational parameters increase from MQ to $(2M + M^2)Q$.
- Optimization relies on SGD methods and the methods like L-BFGS are no longer applicable.
- It can be challenging to initialize $q(\mathbf{u})$.

GP sampling?

- So far, we only consider parameter estimation and posterior inference.
- What about drawing a sample from GP posterior?
- Draw a sample for a set of new location $\mathbf{X}_*$:

$$\mathbf{f}_i \sim \mathcal{N}\left(\mathbf{f}_* | \mathbf{K}_*(\mathbf{K}+\sigma^2\mathbf{I})^{-1}\mathbf{y}, \mathbf{K}_{**} - \mathbf{K}_*(\mathbf{K}+\sigma^2\mathbf{I})^{-1}\mathbf{K}_*^\top\right)$$

Draw a finite sample

- A sample can be computed via the reparameterization trick.
- Compute the Cholesky factor:

$$\mathbf{L}_* = \texttt{chol}(\mathbf{K}_{**} - \mathbf{K}_*(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{K}_*^\top)$$

- Draw sample for an isotropic Gaussian $\epsilon_i \sim \mathcal{N}(0, \mathbf{I})$.
- The posterior sample can be generated by transforming ϵ_i:

$$\mathbf{f}_i = \mathbf{K}_*(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y} + \mathbf{L}_*\epsilon_i$$

What about a function sample?

- GP is a distribution over functions: $\mathcal{GP}(0, k(\cdot, \cdot))$.
- Can we draw a parametric function sample from a GP posterior?

$$f_i \sim \mathcal{GP}(f|0, k(\cdot, \cdot), \mathcal{D})$$

GP function sample

- GP function sample can be handy for downstream tasks.
- For example, Bayesian optimization with Thompson sampling.

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} f_i(\mathbf{x}), \quad f_i \sim \mathcal{GP}(f|0, k(\cdot,\cdot), \mathcal{D})$$

- The minimum of a function sample $=>$ a sample from the distribution of minima

Weight-space approximation to GP

- If the kernel function is degenerate, $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x})$.
- GP is written as a Bayesian Linear Model:

$$p(\mathbf{y}|\mathbf{w}, \mathbf{X}) = \mathcal{N}\left(\mathbf{y}|\mathbf{\Phi}\mathbf{w}, \sigma^2\mathbf{I}\right), \quad p(\mathbf{w}) = \mathcal{N}\left(\mathbf{w}|0, \mathbf{I}\right), \quad \mathbf{\Phi} = (\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_N))$$

- Get back GP formulation by marginalizing $\mathbf{w}$:

$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}\left(\mathbf{y}|\mathbf{\Phi}0, \mathbf{\Phi}\mathbf{I}\mathbf{\Phi}^\top + \sigma^2\mathbf{I}\right) = \mathcal{N}\left(\mathbf{y}|0, \mathbf{K} + \sigma^2\mathbf{I}\right)$$

Sample from Weight-space approximation

- The posterior of GP is encoded into $p(\mathbf{w}|\mathbf{y}, \mathbf{X})$ (a Gaussian distribution).
- First, we draw a sample of $\mathbf{w}$: $\mathbf{w}_i \sim p(\mathbf{w}|\mathbf{y}, \mathbf{X})$.
- Considering the noise-free observation $f = \mathbf{w}\phi(\mathbf{x})$, we get a function sample:

$$f_i(\mathbf{x}) = \mathbf{w}_i \phi(\mathbf{x})$$

How to approximate a kernel function? (1)

- For the above to work, we need the approximation $k(\mathbf{x}, \mathbf{x}') \approx \phi(\mathbf{x})^\top \phi(\mathbf{x})$.
- For stationary kernels, remember Bochner Theorem? (Markus' slides)

- Let's apply Fouriers to the kernel $K(\tau) := K(x, x')$, where $\tau = x - x'$ (instead of $f(x)$)

Theorem (Bochner)

Any stationary kernel $K : \mathbb{R}^D \mapsto \mathbb{R}$ and its spectral density $S : \mathbb{R}^D \mapsto \mathbb{R}$ are Fourier duals

$$K(\tau) = \int_{-\infty}^{\infty} S(\omega) e^{2\pi i \omega^T \tau} d\omega \qquad \text{(Inverse Fourier Transform)}$$

$$S(\omega) = \int_{-\infty}^{\infty} K(\tau) e^{-2\pi i \omega^T \tau} d\tau. \qquad \text{(Fourier Transform)}$$

How to approximate a kernel function? (2)

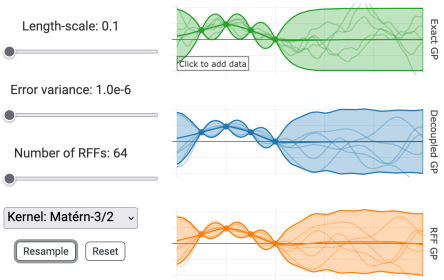
- $k(\mathbf{x}, \mathbf{x}') = \mathbb{E}_\omega[z_\omega(\mathbf{x}) z_\omega(\mathbf{x}')], z_\omega(\mathbf{x}) = \sqrt{2}\cos(\omega^\top \mathbf{x} + b)$
- Draw a sample $\omega_i \sim p(\omega)$.
- Draw a sample $b_i \sim \mathcal{U}[0, 2\pi]$.
- $\phi_i(\mathbf{x}) = \sqrt{2}\cos(\omega_i^\top \mathbf{x} + b_i)$

Kernel Name	$k(\Delta)$	$p(\omega)$
Gaussian	$e^{-\frac{\|\Delta\|_2^2}{2}}$	$(2\pi)^{-\frac{D}{2}} e^{-\frac{\|\omega\|_2^2}{2}}$
Laplacian	$e^{-\|\Delta\|_1}$	$\prod_d \frac{1}{\pi(1+\omega_d^2)}$
Cauchy	$\prod_d \frac{2}{1+\Delta_d^2}$	$e^{-\|\Delta\|_1}$

[Rahimi and Recht, 2008]

Example of GP sample

- Matérn-3/2 kernel
- A more efficient method [Wilson et al., 2020]



https://sml-group.cc/blog/2020-gp-sampling/

Q & A!

Matthias Bauer, Mark van der Wilk, and Carl Edward Rasmussen. Understanding probabilistic sparse gaussian process approximations. In *Advances in Neural Information Processing Systems 29*, pages 1533–1541. 2016.

Thang D Bui, Josiah Yan, and Richard E Turner. A unifying framework for gaussian process pseudo-point approximations using power expectation propagation. *Journal of Machine Learning Research*, 18:3649–3720, 2017.

Alexander Davies. *Effective implementation of Gaussian process regression for machine learning*. PhD thesis, Department of Engineering, University of Cambridge, 2015.

Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, 2018.

James Hensman, Nicolò Fusi, and Neil D. Lawrence. Gaussian processes for big data. page 282–290, 2013.

Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6:1939—-1959, 2005.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2008.

Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*, pages 1257–1264. 2006.

Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, pages 567–574, 2009.

Christopher K. I. Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, pages 682–688. 2001.

James T. Wilson, Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, and Marc Peter Deisenroth. Efficiently sampling functions from gaussian process posteriors. In *International Conference on Machine Learning*, 2020.