Scalability of Gaussian Process

Zhenwen Dai

Spotify

13 September 2022 @GPSS 2022

Outline

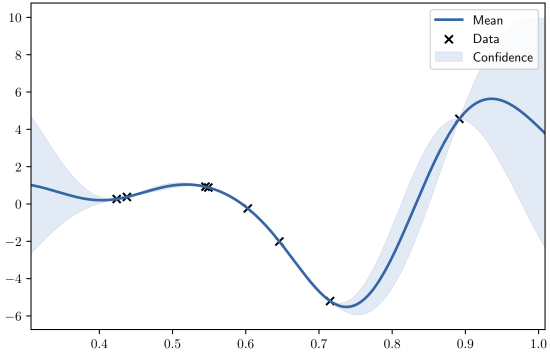
- What is the scalability issue of Gaussian Process?
- Numerical solution
- Model/Inference Approximation
- Mini-batch Training

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:

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

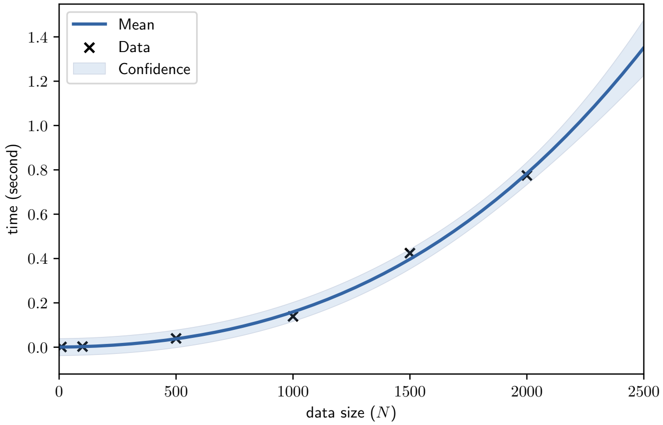
- Take a Cholesky decomposition: $\mathbf{L} = \texttt{chol}(\mathbf{K}+\sigma^2\mathbf{I})$, such that $\mathbf{K}+\sigma^2\mathbf{I} = \mathbf{L}\mathbf{L}^\top$.
- The computational complexity is $O(N^3 + N^2 + N)$. Therefore, the overall complexity including the computation of $\mathbf{K}$ is $O(N^3)$.

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
```

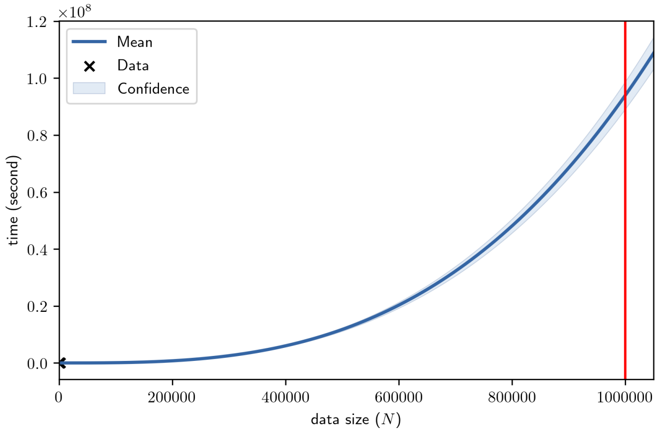
Empirical analysis of computational time

- 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 $O(N^3)$ mainly 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:

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

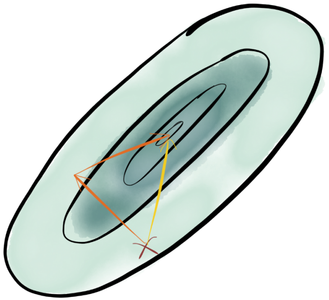

Conjugate Gradient (CG)

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

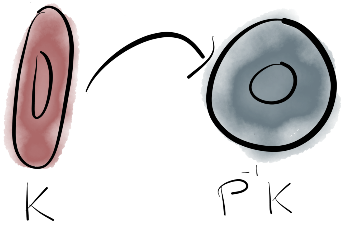
- Numerical stability and rate of convergence of CG are *sensitive* to the condition number:

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

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

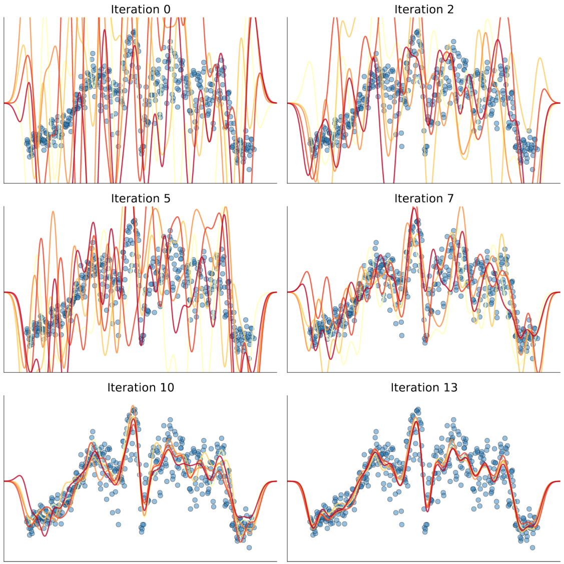


Preconditioning

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 implemented in GPyTorch [Gardner et al., 2018].

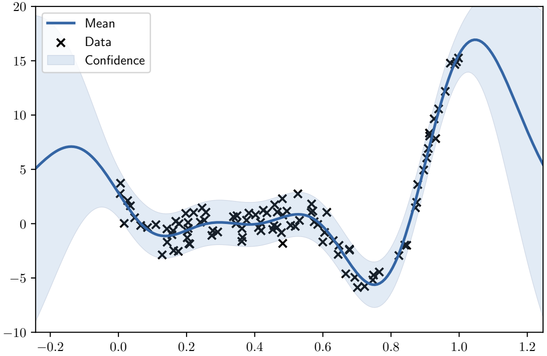


$O(N^2)$ is still slow!

Gaussian Process Model/Inference Approximation

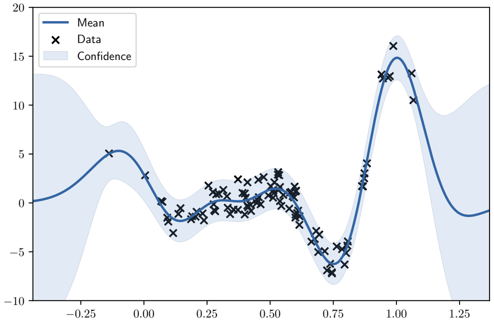
Big data (?)

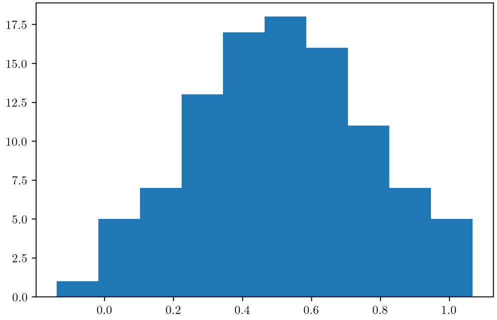
- lots of data $\neq$ complex function
- In real world problems, we often collect a lot of data for modeling relatively simple relations.



Data subsampling?

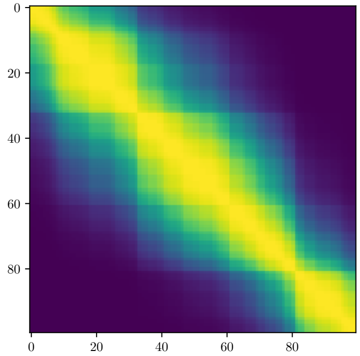
- Real data often do not evenly distributed.
- We tend to get a lot of data on common cases and very few data on rare cases.

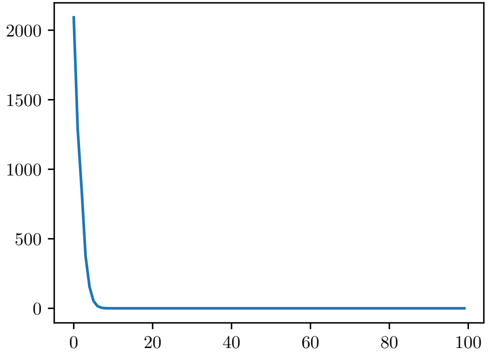




Covariance matrix of redundant data

- 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 $\tilde{\mathbf{K}}$.

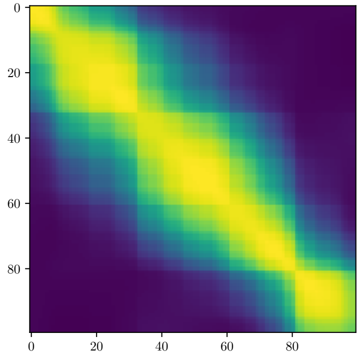
$\tilde{\mathbf{K}} = \mathbf{K}_z \mathbf{K}_{zz}^{-1} \mathbf{K}_z^\top$, where $\mathbf{K}_z = \mathbf{K}(\mathbf{X}, \mathbf{Z})$ and $\mathbf{K}_{zz} = \mathbf{K}(\mathbf{Z}, \mathbf{Z})$.

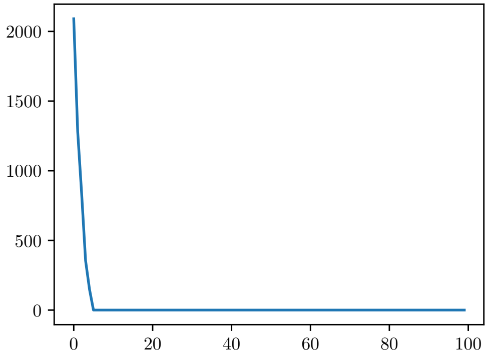
- Note that $\tilde{\mathbf{K}} \in \mathbb{R}^{N \times N}$, $\mathbf{K}_z \in \mathbb{R}^{N \times M}$ and $\mathbf{K}_{zz} \in \mathbb{R}^{M \times M}$.
- The log-likelihood is approximated by

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

Nyström approximation example

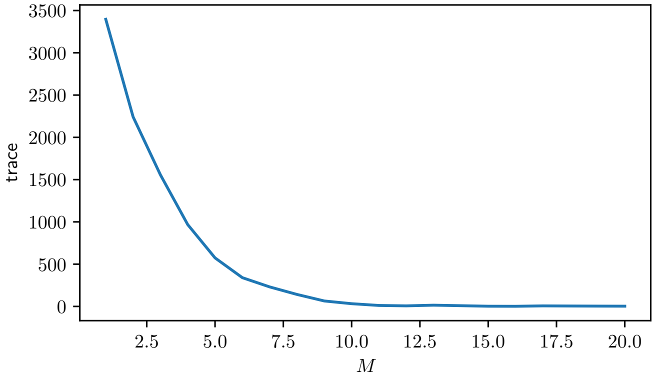
The covariance matrix with Nyström approximation using 5 random data points:





Nyström approximation example

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



Nyström approximation implementation

- The naïve formulation does **not** bring any computational benefits.

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

- $\tilde{\mathbf{K}} + \sigma^2\mathbf{I}$ is a $N \times N$ matrix!

Efficient computation using Woodbury formula

- Rewrite the log-likelihood

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

- by applying the Woodbury formula:

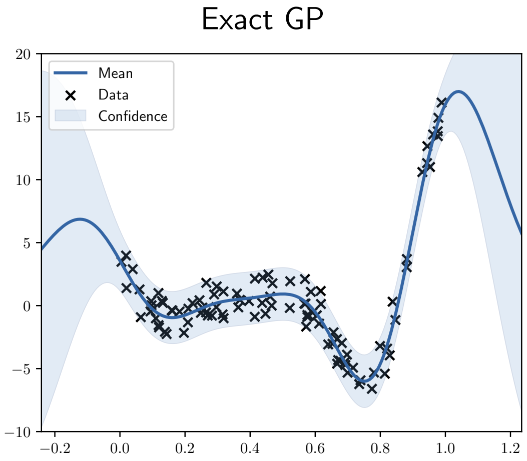
$$(\mathbf{K}_z\mathbf{K}_{zz}^{-1}\mathbf{K}_z^\top + \sigma^2\mathbf{I})^{-1} = \sigma^{-2}\mathbf{I} - \sigma^{-4}\mathbf{K}_z(\mathbf{K}_{zz} + \sigma^{-2}\mathbf{K}_z^\top\mathbf{K}_z)^{-1}\mathbf{K}_z^\top$$

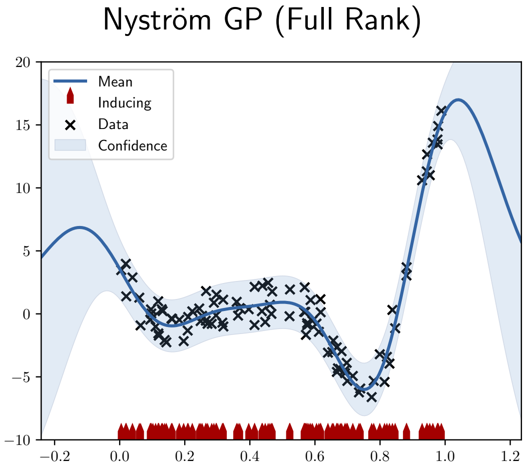
- Note that $(\mathbf{K}_{zz} + \sigma^{-2}\mathbf{K}_z^\top\mathbf{K}_z) \in \mathbb{R}^{M\times M}$.
- The computational complexity reduces to $O(NM^2)$.

Nyström approximation summary

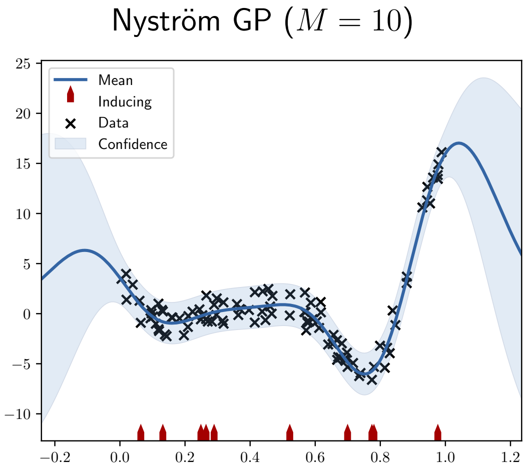
- The approximation is directly done on the covariance matrix without the concept of pseudo data.
- The approximation becomes exact if the whole data set is taken, *i.e.*, $\mathbf{K}\mathbf{K}^{-1}\mathbf{K}^\top = \mathbf{K}$.
- The subset selection is done randomly.

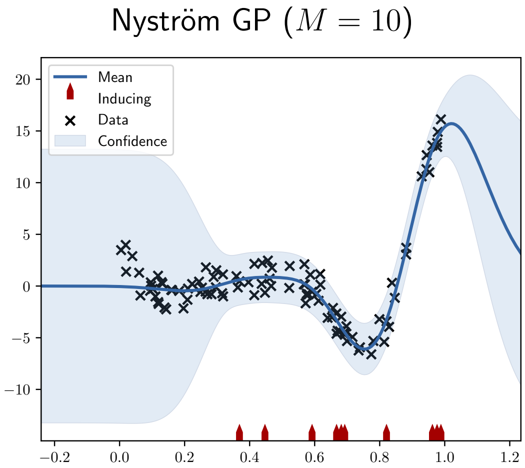
Examples of Nyström approximation (1)





Examples of Nyström approximation (2)





$N = 100$

Issues with random sampling

- Performance can be bad if unlucky.
- Areas with lots of data get more samples.
- *Can we do better?*

Take a step back

- In the log-likelihood of GP

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

- the covariance $\mathbf{K}$ is **computed** using the kernel function $k(\cdot, \cdot)$ on the inputs $\mathbf{X}$.
- Let's construct the Nyström approximation differently.

Pseudo data

- Imagine that there are a set of additional data points $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_M)^\top$.
- $\mathbf{z}$ lies in the **same** space as $\mathbf{x}$ does.
- The corresponding outputs $\mathbf{u} = (u_1, \ldots, u_M)$ are unknown.
- $\mathbf{Z}$ and $\mathbf{u}$ are referred to as **pseudo data**.
- $\mathbf{Z}$ are referred to as **inducing inputs**.
- $\mathbf{u}$ are referred to as **inducing variables**.

Pseudo data approximation

- With pseudo data $\mathbf{Z}$ and $\mathbf{u}$,
- the covariance of $\mathbf{u}$, $\mathbf{K}_{uu}$, can be computed $k(\cdot, \cdot)$ on the inputs $\mathbf{Z}$.
- The cross covariance between $\mathbf{f}$ and $\mathbf{u}$, $\mathbf{K}_{ff}$, can be computed as well.
- We can construct a similar approximation: $\tilde{\mathbf{K}} = \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{fu}^{\top}$.

Optimizing the pseudo data locations

- How does it differ from Nyström approximation?
- The inducing inputs are explicitly parameterized by $\mathbf{Z}$.
- Search for the optimal $\mathbf{Z}$ via optimization:

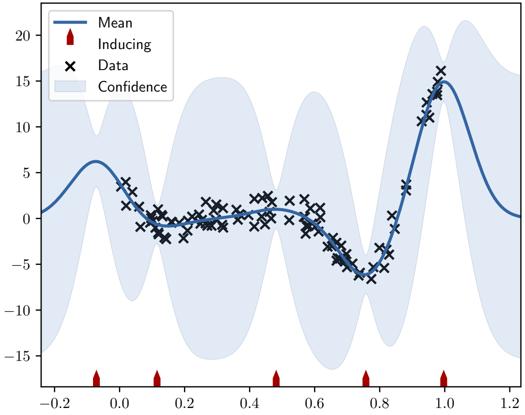
$$\mathbf{Z}^* = \arg\max_{\mathbf{Z}} \log \mathcal{N}\left(\mathbf{y}|0, \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{fu}^\top + \sigma^2\mathbf{I}\right).$$

- Does it work? Not really.

Deterministic Training Conditional (DTC)

- This formulation is known as Deterministic Training Conditional (DTC).
- Five inducing points moves out of scope.
- Overfits: Its logL is -156.73, while the logL of exact GP is -167.36.

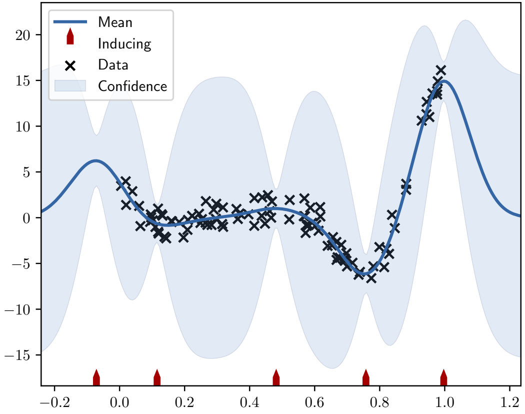
GP-DTC ($M = 10$)



Deterministic Training Conditional (DTC) (2)

- The inducing inputs adds a lot of parameters to the model.
- This model behaves much more like parametric model.
- The original DTC method [Seeger et al., 2003] greedily selects a subset of training data as the inducing points.

GP-DTC ($M = 10$)



Take a different approach

- Assume the pseudo data follow the same distribution as the observed data.
- In other words, $\mathbf{f}$ and $\mathbf{u}$ jointly follows the same GP:

$$p(\mathbf{f}, \mathbf{u}|\mathbf{X}, \mathbf{Z}).$$

- Compared to the original GP, the prior distribution is **not** changed because

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

- Alternatively, the prior distribution can be written as

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

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

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:

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

- Plug it into the lower bound:

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

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 $q(\mathbf{u})$:

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

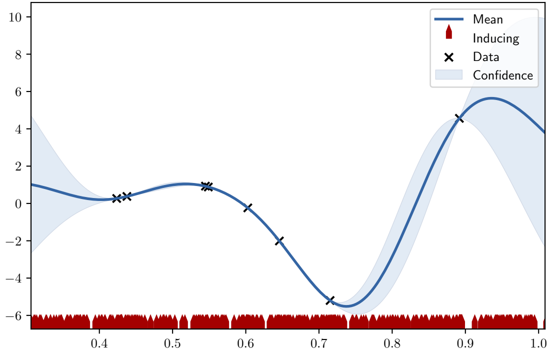
- Make the bound as tight as possible by plugging in μ^* and Σ^*:

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

- 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.



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

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

- 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:

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

“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)

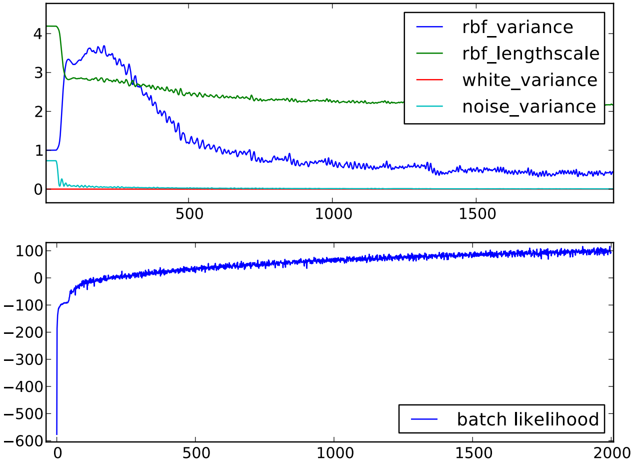
- The resulting lower bound can be written as the sum over the data,

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

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

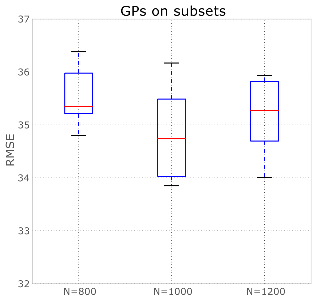
$$\theta_{t+1} = \theta_t - \eta \frac{\mathrm{d}\mathcal{L}_{\mathsf{MB}}}{\mathrm{d}\theta}.$$

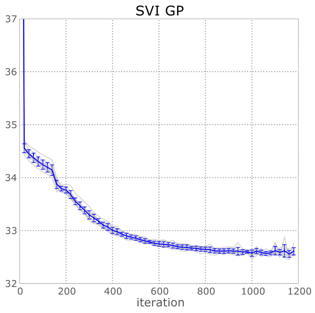
2D Synthetic Data

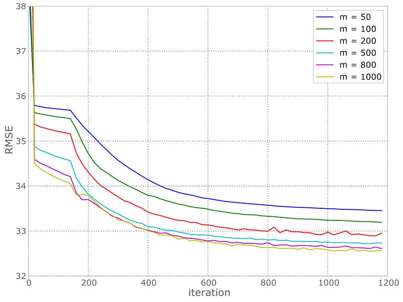


Airline Delay Data

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 $q(\mathbf{u})$.

Questions?

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.

Matthias W. Seeger, Christopher K. I. Williams, and Neil D. Lawrence. Fast forward selection to speed up sparse gaussian process regression. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, pages 254–261, 2003.

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.