G-Research Seminar

Imperial College London

Reliable training of GP approximations

Mark van der Wilk

Department of Computing
Imperial College London

@markvanderwilk
m.vdwilk@imperial.ac.uk

Jun 2, 2021

Goals

Main goal:

Uncertainty-aware regression for making **robust** predictions that you can **act** on.

Goals

Main goal:

Uncertainty-aware regression for making **robust** predictions that you can **act** on.

The goal of the work I will discuss today:

High-quality GP approximations for **big data** that you can train **without human intervention**.

Goals

Main goal:

Uncertainty-aware regression for making **robust** predictions that you can **act** on.

The goal of the work I will discuss today:

High-quality GP approximations for **big data** that you can train **without human intervention**.

... And when I mean "high-quality", I mean that we really want to squeeze every last bit of predictive performance out of the data.

Overview

The What & Why of Gaussian Processes

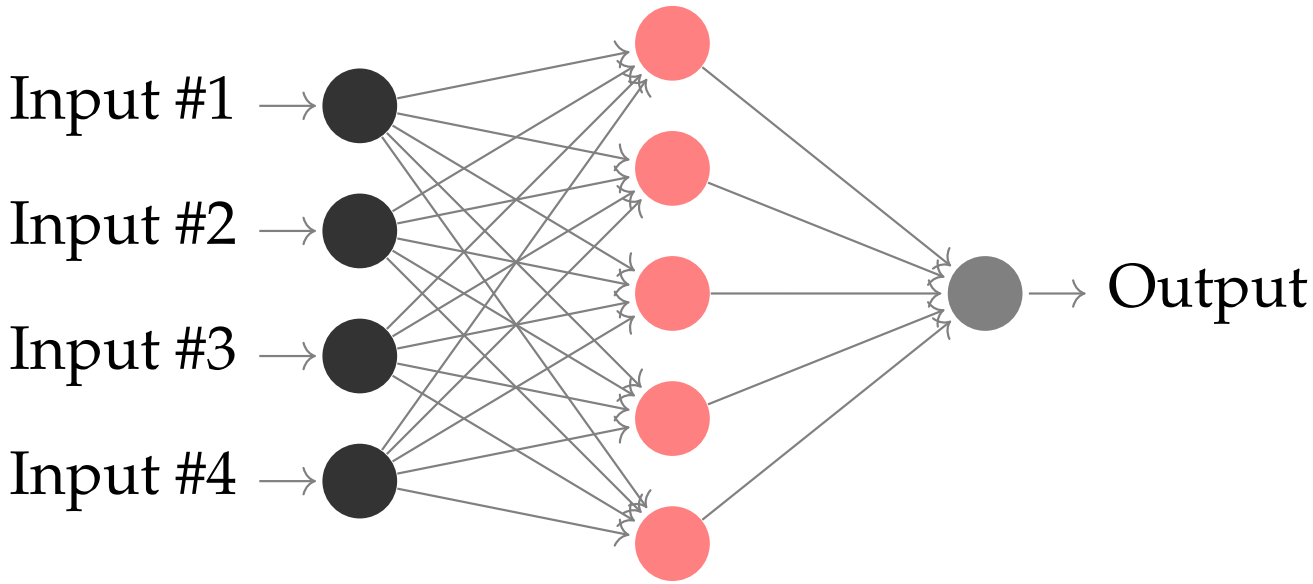
Variational Inference for Big Data

Theory and Practice of Training Variational GPs

Robust Conjugate Gradient Methods

Wrap-up

Neural networks are basis function models

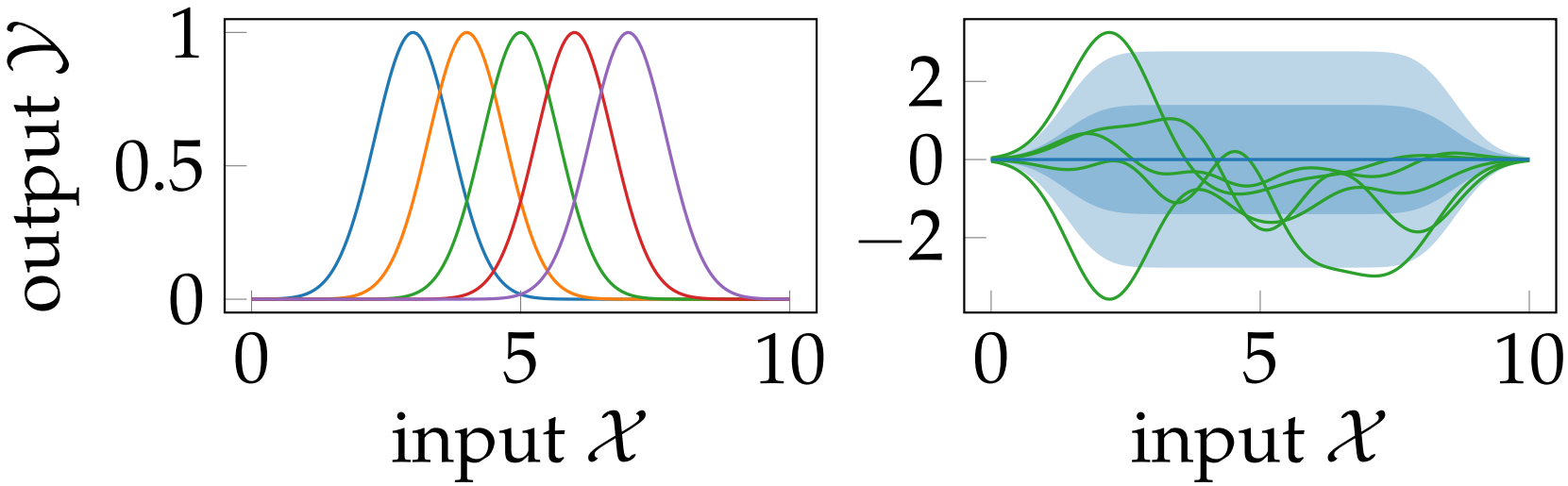


$$f(\mathbf{x}) = \sum_{b=1}^{B} w_b \phi_b(\mathbf{x}) = \mathbf{w}^\mathsf{T} \boldsymbol{\phi}(\mathbf{x})$$

$$\phi_b(\mathbf{x}) = \sigma\left(\sum_{d=1}^{D} \tilde{w}_d x_d \right) = \sigma(\tilde{\mathbf{w}}^\mathsf{T} \mathbf{x})$$

Bayesian Neural Networks are a prior over functions

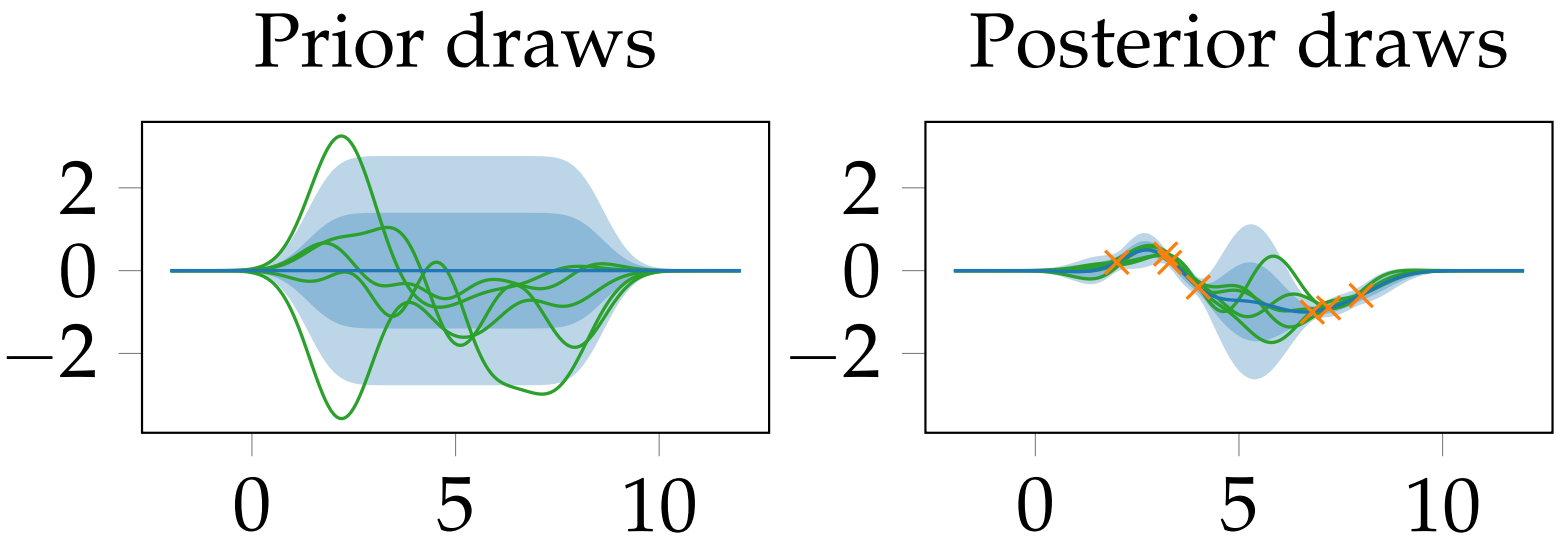
Placing priors on $\mathbf{w}$ gives us a distribution over functions:



$$f(\mathbf{x}) = \sum_{b=1}^{B} w_b \phi_b(\mathbf{x}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x})$$

$$\mathbf{w} \sim \mathcal{N}\left(\mathbf{w}; 0, \sigma_w^2 I\right)$$

Bayesian advantages



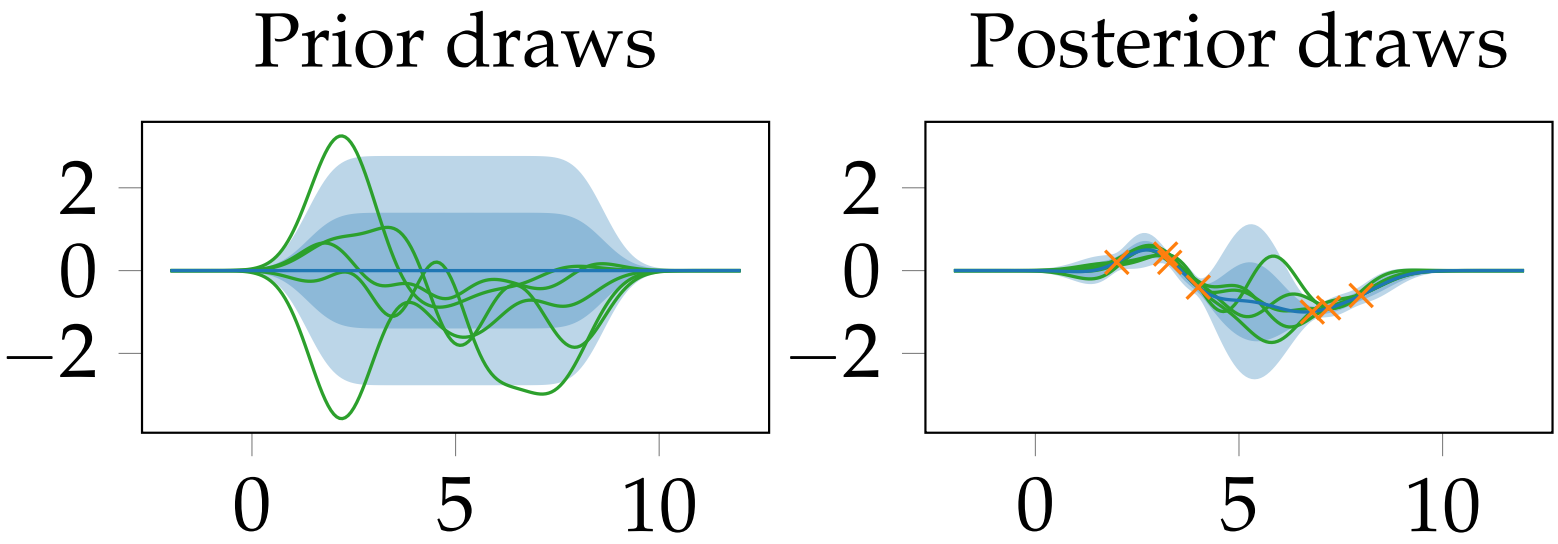
Using the prior, we can obtain the **posterior**:

$$p(\mathbf{w}|\mathbf{y},\theta) = \frac{\prod_n p(y_n|\mathbf{w},\theta)p(\mathbf{w}|\theta)}{p(\mathbf{y}|\theta)}$$

Advantages:

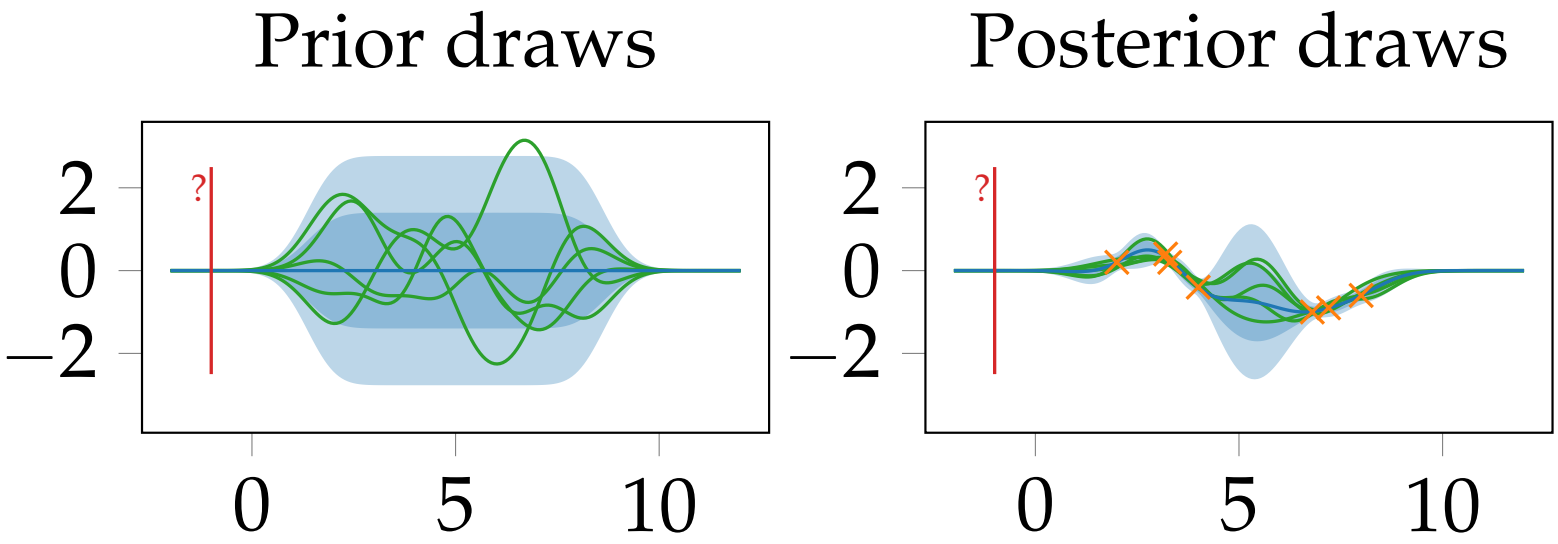
- Posterior quantifies our **uncertainty**
- **Marginal likelihood** $p(\mathbf{y}|\theta)$ gives us the hyperparameters

What is wrong with the model?



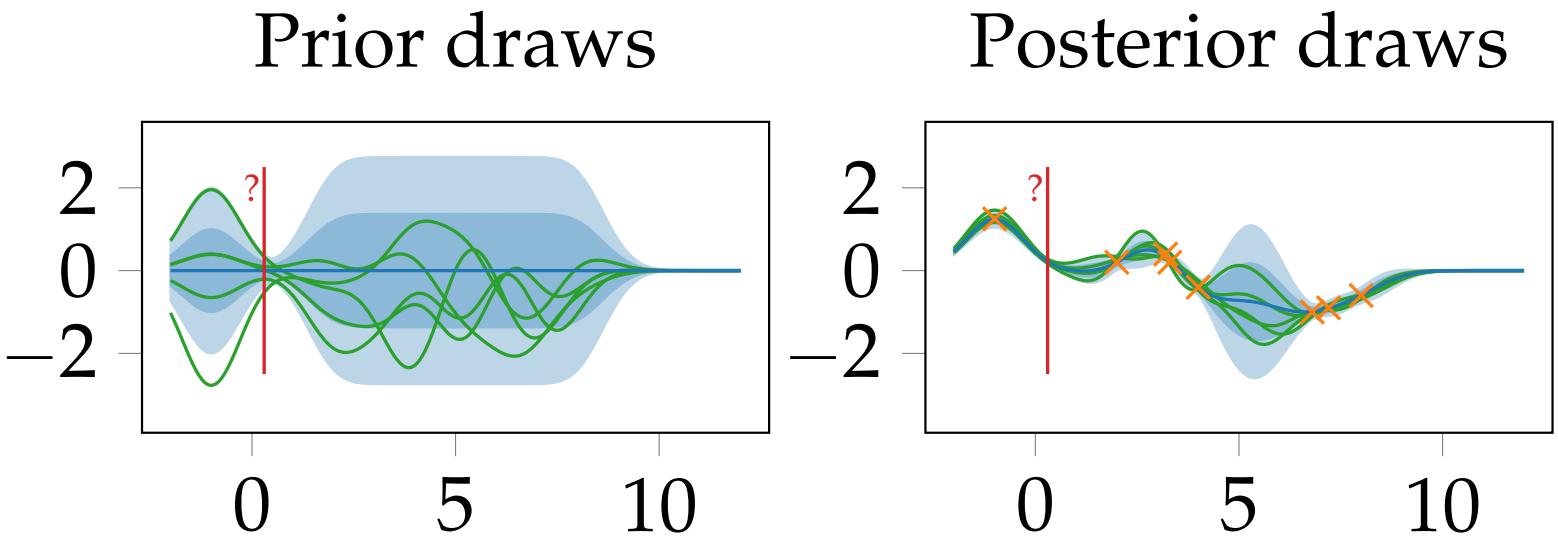
- How to set basis function locations?
- How many basis functions?
- Should we be so certain far from the data?

What is wrong with the model?



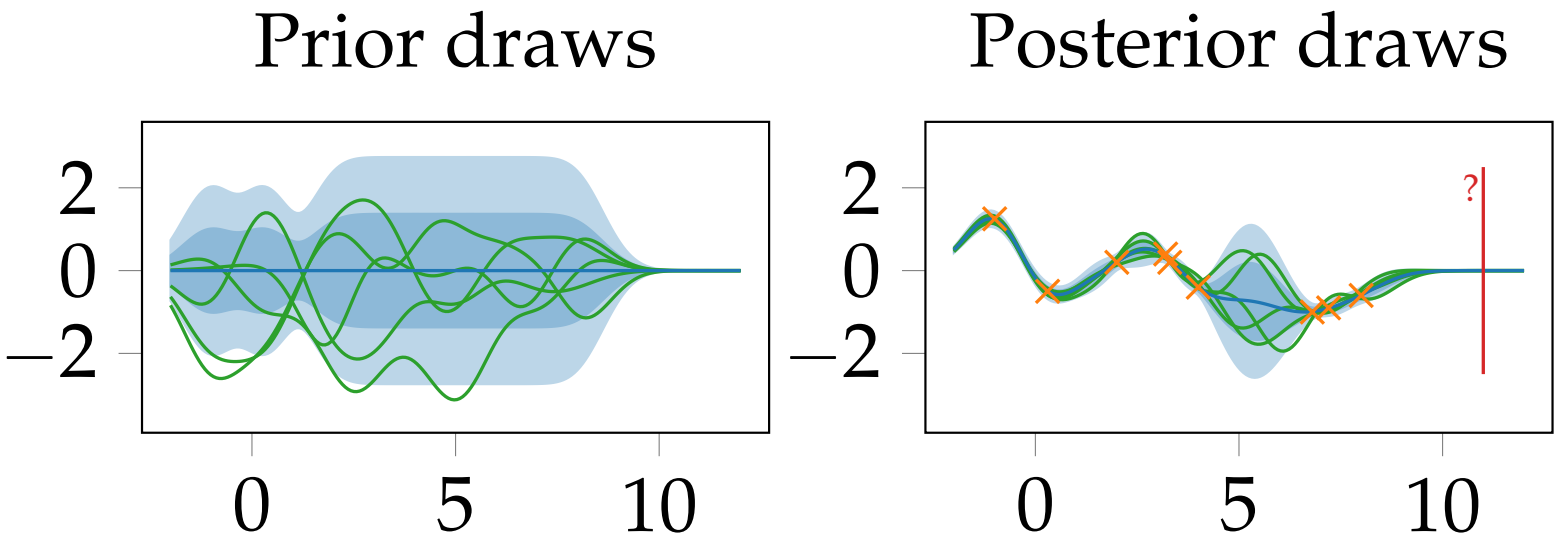
- How to set basis function locations?
- How many basis functions?
- Should we be so certain far from the data?

What is wrong with the model?



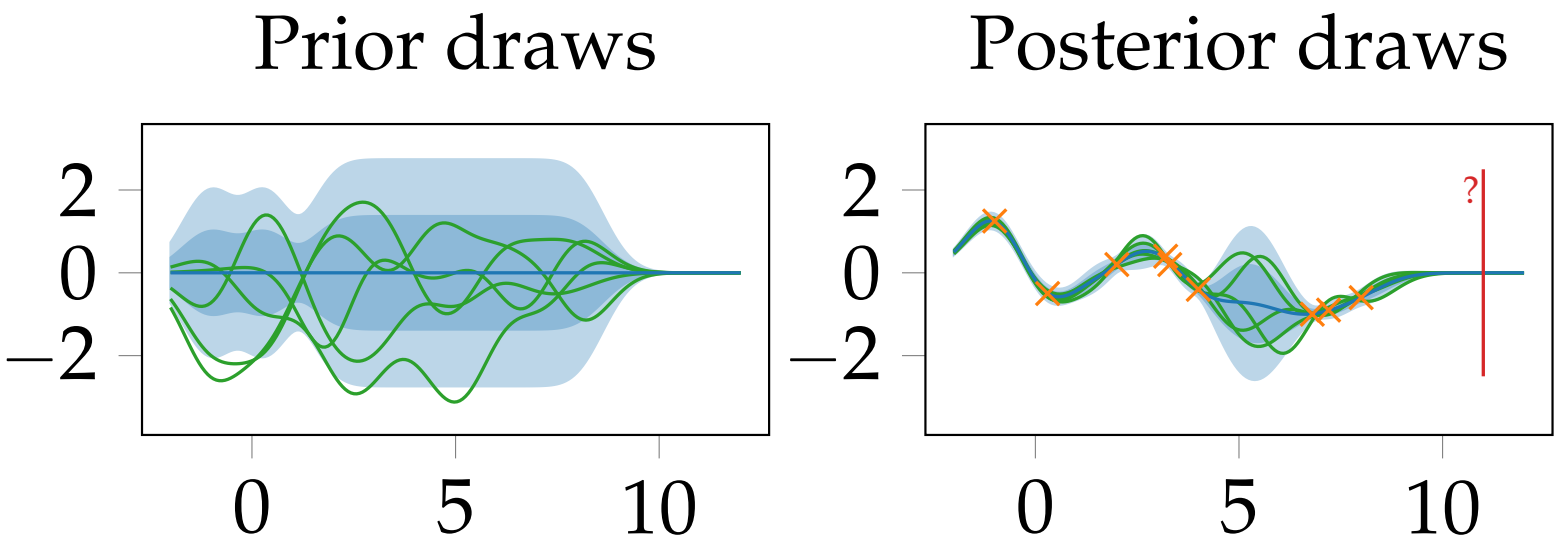
- How to set basis function locations?
- How many basis functions?
- Should we be so certain far from the data?

What is wrong with the model?



- How to set basis function locations?
- How many basis functions?
- Should we be so certain far from the data?

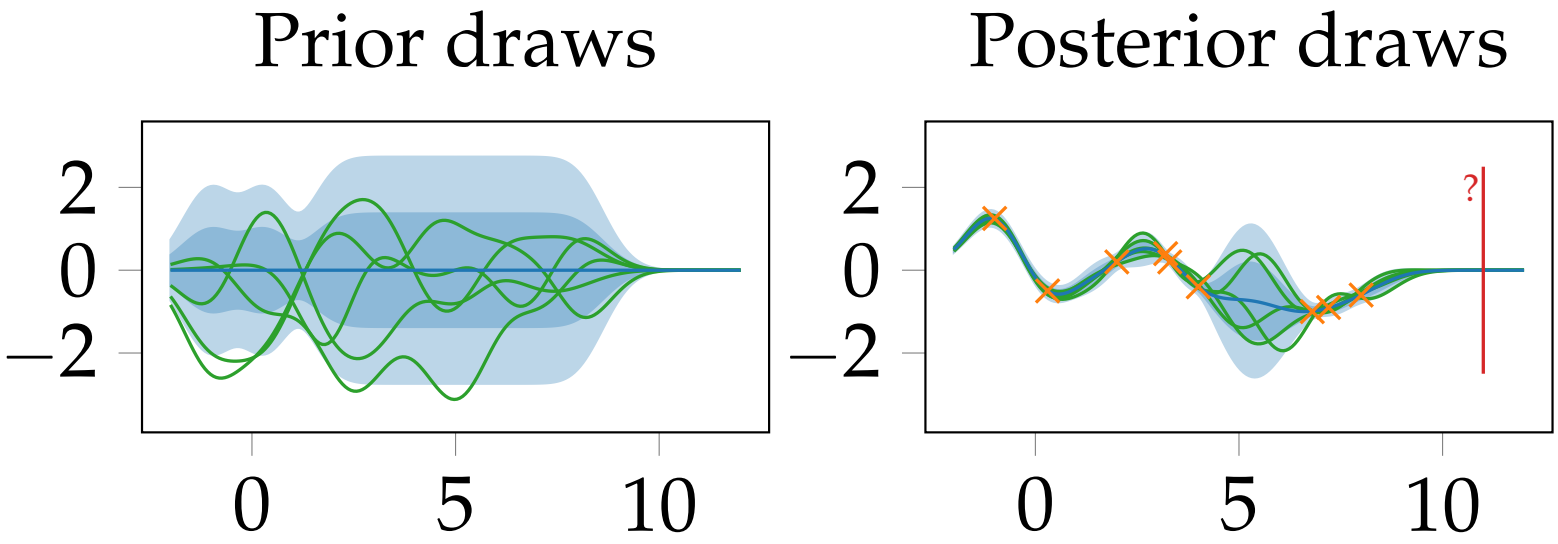
What is wrong with the model?



- How to set basis function locations?
- How many basis functions?
- Should we be so certain far from the data?

Solution: Use large number of basis functions

What is wrong with the model?



- How to set basis function locations?
- How many basis functions?
- Should we be so certain far from the data?

Solution: Use an infinite number of basis functions?

A different representation?

- Having many weights is expensive (cost is $\mathcal{O}\left(W^3\right)$)
- Our likelihood only really depends on the function values $f(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\mathsf{T}\mathbf{w}$
- Weights are only a means to specify the prior over functions

A different representation?

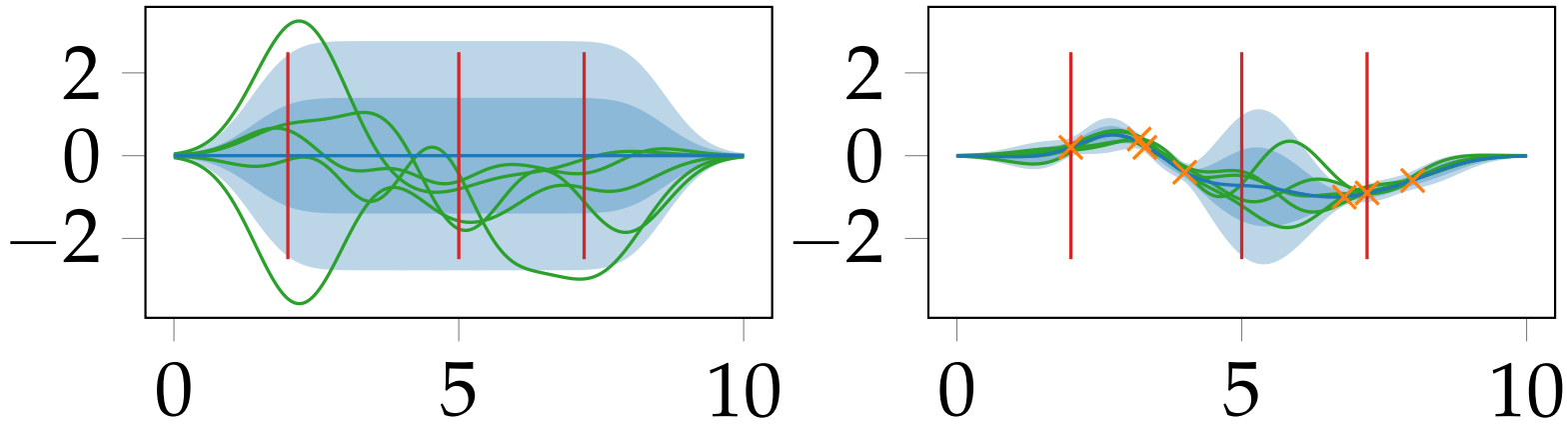
- Having many weights is expensive (cost is $\mathcal{O}(W^3)$)
- Our likelihood only really depends on the function values $f(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\mathsf{T}\mathbf{w}$
- Weights are only a means to specify the prior over functions

Can we specify the distribution over functions directly?

Distributions over functions

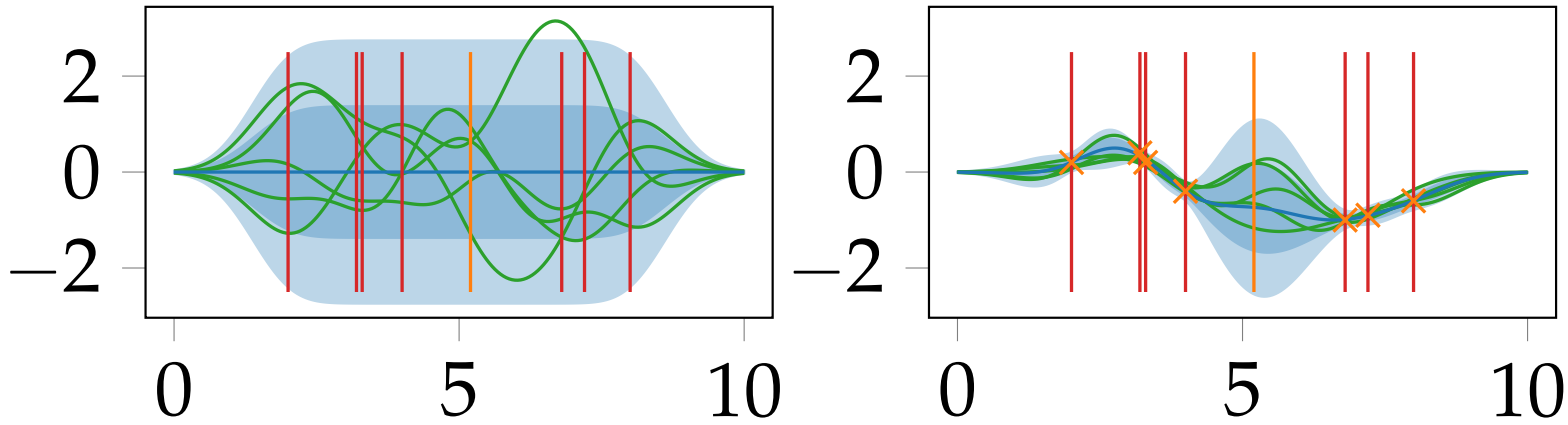
We can not write densities for entire functions "$p\left(f_{(\cdot)}\right)$"
However, a **density over the values** of $f_{(\cdot)}$ at **arbitrary input locations** is just as good.

$$p\left(f_{(\mathbf{x}_1)}, f_{(\mathbf{x}_2)}, f_{(\mathbf{x}_3)}\right)$$



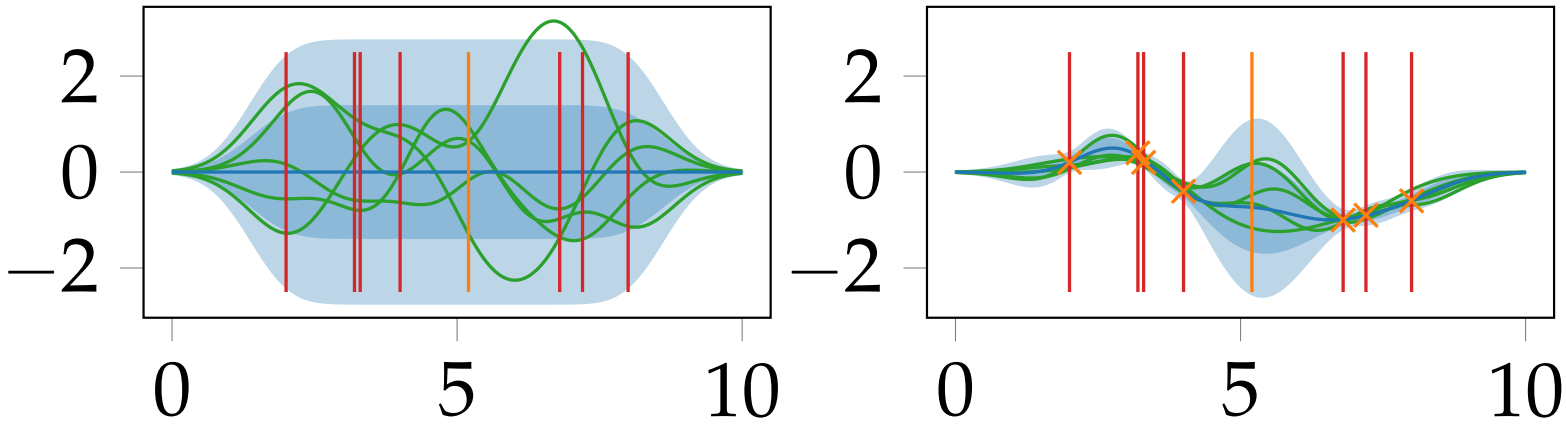
Distributions over functions

This allows us to always find the posterior of observations and any points to predict at.



Distributions over functions

This allows us to always find the posterior of observations and any points to predict at.



$$
\begin{aligned}
p\left(f_{(X)}, f_{(\cdot)} | \mathbf{y}\right) &= \frac{p\left(\mathbf{y}|f_{(X)}\right) p\left(f_{(X)}, f_{(\cdot)}\right)}{p(\mathbf{y})} \\
&= \frac{p\left(\mathbf{y}|f_{(X)}\right) p\left(f_{(X)}\right)}{p(\mathbf{y})} p\left(f_{(\cdot)}|f_{(X)}\right) \\
&= p\left(f_{(X)}|\mathbf{y}\right) p\left(f_{(\cdot)}|f_{(X)}\right)
\end{aligned}
$$

A different representation of the neural net prior

Concentrating on the function values at input locations $X \in \mathbb{R}^{N \times D}$:

$$p(f(X)) = \int \delta(f(X) - \Phi(X)\mathbf{w}) p(\mathbf{w}) \mathrm{d}\mathbf{w}$$
$$= \mathcal{N}\left(f(X); 0, \sigma^2 \Phi(X)\Phi(X)^\mathsf{T}\right)$$

$$\left[\Phi(X)\Phi(X)^\mathsf{T}\right]_{nn'} = \boldsymbol{\phi}(\mathbf{x})^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}')$$

A different representation of the neural net prior

Concentrating on the function values at input locations $X \in \mathbb{R}^{N \times D}$:

$$p(f(X)) = \int \delta(f(X) - \Phi(X)\mathbf{w}) p(\mathbf{w}) \mathrm{d}\mathbf{w}$$
$$= \mathcal{N}\left(f(X); 0, \sigma^2 \Phi(X)\Phi(X)^\mathsf{T}\right)$$

$$\left[\Phi(X)\Phi(X)^\mathsf{T}\right]_{nn'} = \boldsymbol{\phi}(\mathbf{x})^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}')$$

- We only need to define the **kernel function**.

A different representation of the neural net prior

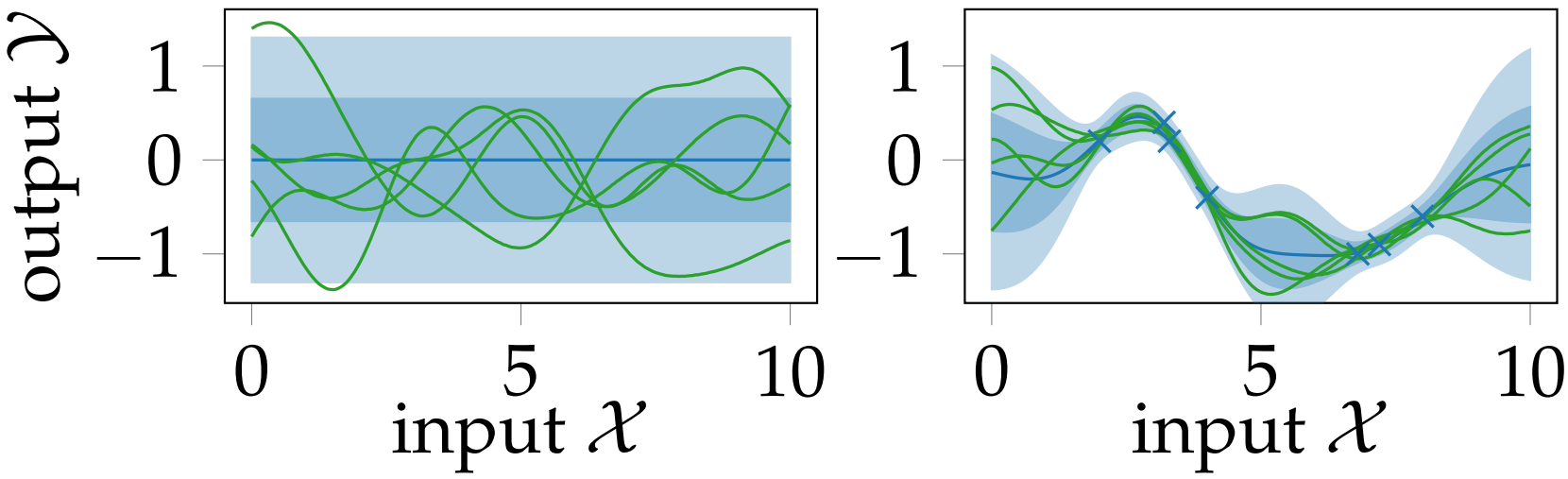
Concentrating on the function values at input locations $X \in \mathbb{R}^{N \times D}$:

$$p(f(X)) = \int \delta(f(X) - \Phi(X)\mathbf{w}) p(\mathbf{w}) \mathrm{d}\mathbf{w}$$
$$= \mathcal{N}\left(f(X); 0, \sigma^2 \Phi(X)\Phi(X)^\mathsf{T}\right)$$

$$\left[\Phi(X)\Phi(X)^\mathsf{T}\right]_{nn'} = \boldsymbol{\phi}(\mathbf{x})^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}')$$

- We only need to define the **kernel function**.
- There are kernels which correspond to **infinite** basis functions.

Three problems solved



- How to set basis function locations? → **everywhere!**
- How may basis functions? → **infinite!**
- Should we be so certain far from the data? → **No, and we don't have to be!**

Scalability

Gaussian processes are the "**gold-standard**" uncertainty-aware regression method. Training requires:

1. Selecting kernel hyperparameters

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \log p(\mathbf{y}|\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \log \mathcal{N}\left(\mathbf{y}; 0, \mathbf{K_{ff}} + \sigma^2 \mathbf{I}_N\right), \quad (1)$$

$$[\mathbf{K_{ff}}]_{nn'} = k(\mathbf{x}_n, \mathbf{x}_{n'}). \quad (2)$$

2. Calculate predictive mean at new point $\mathbf{x}_*$

$$\mu_* = \mathbf{k}_{*\mathbf{f}}(\mathbf{K_{ff}} + \sigma^2)^{-1}\mathbf{y} \quad (3)$$

3. Calculate predictive variance at new point

$$\sigma_*^2 = k_{**} - \mathbf{k}_{*\mathbf{f}}\mathbf{K_{ff}}^{-1}\mathbf{k}_{*\mathbf{f}}^\top \quad (4)$$

Overview

The What & Why of Gaussian Processes

Variational Inference for Big Data

Theory and Practice of Training Variational GPs

Robust Conjugate Gradient Methods

Wrap-up

Two problems

Our general model:

$$\text{Prior:} \qquad f_{(\cdot)} \sim \mathcal{GP}\left(0, k(\mathbf{x}, \mathbf{x}')\right)$$

$$\text{Likelihood:} \qquad y_n \sim p\left(y_n | f_{(\mathbf{x}_n)}\right)$$

We have two problems:

- Non-conjugacy
  When $p(f_{(X)}|\mathbf{y}) = \frac{\prod_n p\left(y_n|f_{(\mathbf{x}_n)}\right)p\left(f_{(X)}\right)}{p(\mathbf{y})}$ is non-Gaussian.
- Scale
  Prior and posterior require $\mathcal{O}\left(N^2\right)$ space and $\mathcal{O}\left(N^3\right)$ time.

Single solution

Variational inference:

1. Define a tractable set of posteriors $q_\eta(f)$
2. Minimise the KL divergence to the true posterior $\mathrm{KL}\big[q_\eta(f)||p(f|\mathbf{y})\big]$

Single solution

Variational inference:

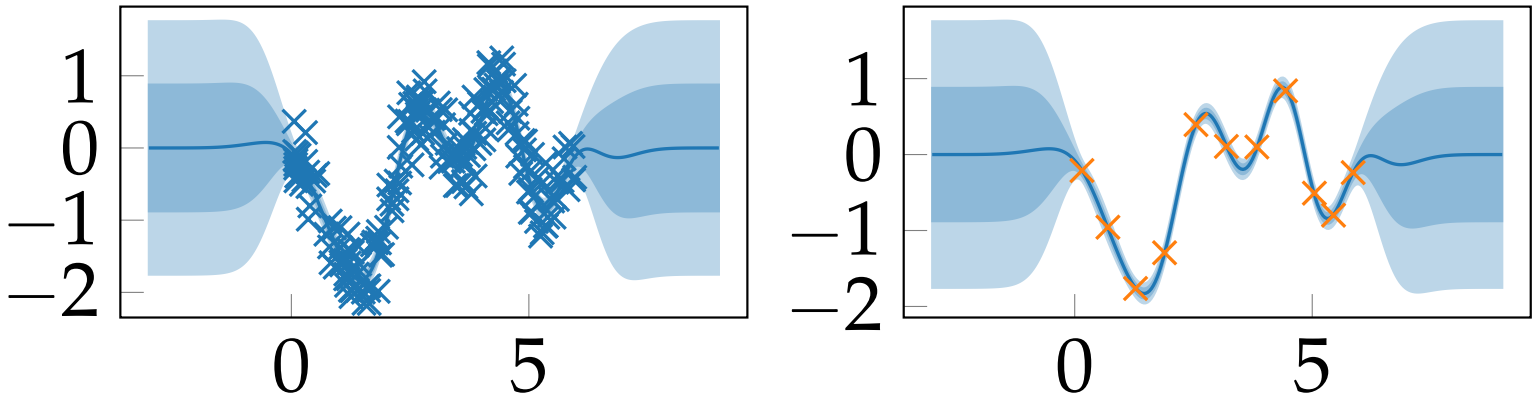
1. Define a tractable set of posteriors $q_\eta(f)$
2. Minimise the KL divergence to the true posterior $\mathrm{KL}\big[q_\eta(f)||p(f|\mathbf{y})\big]$

Procedure:

1. Formulate the lower bound $\mathcal{L} \leqslant \log p(\mathbf{y}|\theta)$, such that $\mathcal{L} + \mathrm{KL}\big[q_\eta||p\big] = \log p(\mathbf{y}|\theta)$
2. Maximise w.r.t. variational parameters η
3. Also maximise w.r.t. hyperparameters θ, for approximate maximum likelihood

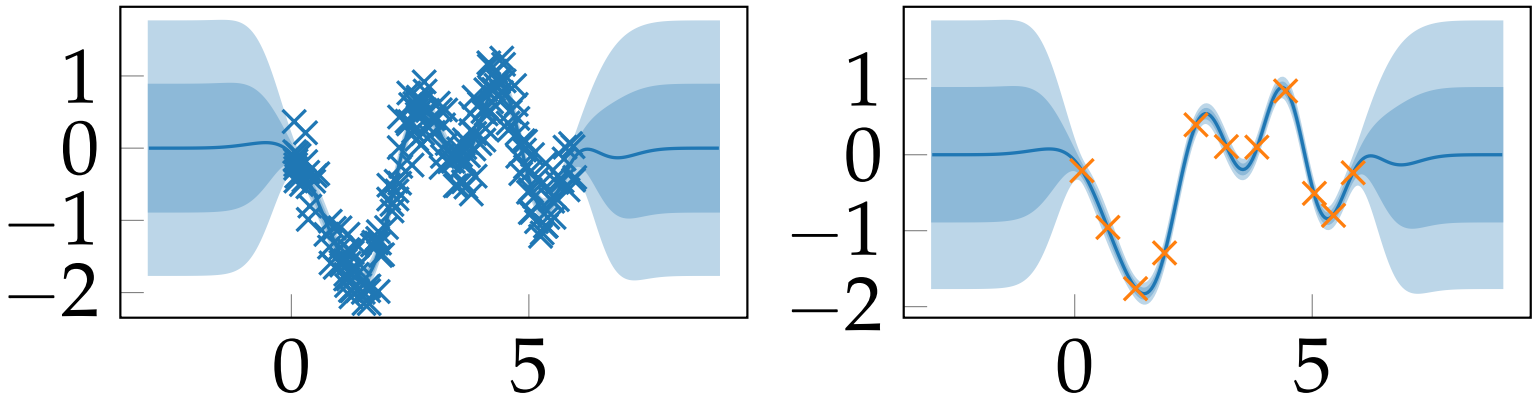
Set of approximate posteriors

A small GP is a tractable GP:



- A single noiseless point constrains the prior similarly to many noisy ones
- Without the redundancy, we can get **very close** with **far fewer points**

Set of approximate posteriors

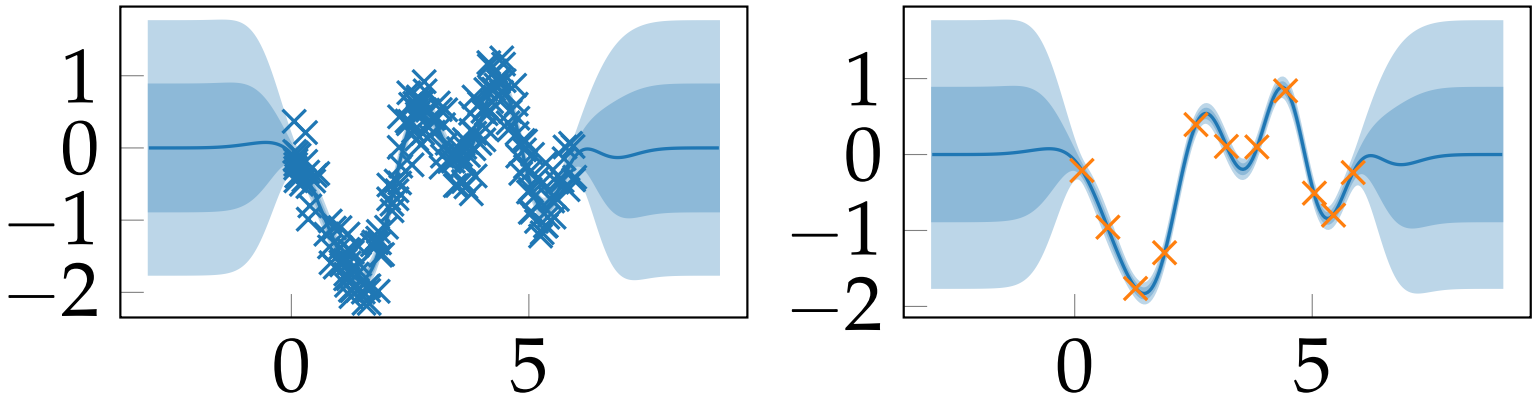


Can choose a posterior conditioned on arbitrary inputs $Z = \{\mathbf{z}_m\}$ and observations $\{\tilde{y}_m\}$:

$$
\begin{aligned}
q_\eta(f) &= \frac{\prod_m \mathcal{N}\left(\tilde{y}_m; f_{(\mathbf{z}_m)}, \tilde{\sigma}^2\right) p(f)}{p(\tilde{\mathbf{y}})} \\
&= \frac{\prod_m \mathcal{N}\left(\tilde{y}_m; f_{(\mathbf{z}_m)}, \tilde{\sigma}^2\right) p\left(f_{(Z)}\right)}{p(\tilde{\mathbf{y}})} p\left(f | f_{(Z)}\right) \\
&= p\left(f | f_{(Z)}\right) p\left(f_{(Z)} | \tilde{\mathbf{y}}\right)
\end{aligned}
$$

$\eta = \{Z, \tilde{y}_m, \tilde{\sigma}\}$

Set of approximate posteriors



Can further relax the assumption and choose an arbitrary fully correlated Gaussian likelihood:

$$q_\eta(f) = \frac{\prod_m \mathcal{N}\left(\tilde{y}_m; f_{(\mathbf{z}_m)}, \Sigma\right) p(f)}{p(\tilde{\mathbf{y}})}$$
$$= \mathcal{N}\left(f_{(Z)}; \mu, \Sigma\right) p(f|f_{(Z)}) = p\left(f|f_{(Z)}\right) q(f_{(Z)})$$

$$\eta = \{Z, \boldsymbol{\mu}, \Sigma\}$$

Variational lower bound

$$\mathrm{KL}[q(f)||p(f|\mathbf{y})] = \mathrm{KL}\Big[p\Big(f|f_{(X)},f_{(Z)}\Big)q\Big(f_{(X)},f_{(Z)}\Big) \quad || \\ p\Big(f|f_{(X)},f_{(Z)}\Big)p\Big(f_{(X)},f_{(Z)}|\mathbf{y}\Big)\Big]$$

Variational lower bound

$$\mathrm{KL}[q(f)||p(f|\mathbf{y})] = \mathrm{KL}\Big[\cancel{p\Big(f|f_{(X)},f_{(Z)}\Big)}q\Big(f_{(X)},f_{(Z)}\Big) \quad || \\ \cancel{p\Big(f|f_{(X)},f_{(Z)}\Big)}p\Big(f_{(X)},f_{(Z)}|\mathbf{y}\Big)\Big]$$

Variational lower bound

$$\mathrm{KL} = \log p(\mathbf{y}) - \mathbb{E}_{q\left(f_{(\mathrm{X})}, f_{(\mathrm{Z})}\right)}\left[\log \frac{\prod_n p\left(y_n | f_{(\mathbf{x}_n)}\right) p\left(f_{(\mathrm{X})} | f_{(\mathrm{Z})}\right) p\left(f_{(\mathrm{Z})}\right)}{p\left(f_{(\mathrm{X})} | f_{(\mathrm{Z})}\right) q\left(f_{(\mathrm{Z})}\right)}\right]$$

Variational lower bound

$$\mathrm{KL} = \log p(\mathbf{y}) - \mathbb{E}_{q\left(f_{(\mathrm{X})}, f_{(\mathrm{Z})}\right)}\left[\log \frac{\prod_n p\left(y_n | f_{(\mathbf{x}_n)}\right) \cancel{p\left(f_{(\mathrm{X})} | f_{(\mathrm{Z})}\right)} p\left(f_{(\mathrm{Z})}\right)}{\cancel{p\left(f_{(\mathrm{X})} | f_{(\mathrm{Z})}\right)} q\left(f_{(\mathrm{Z})}\right)}\right]$$

Variational lower bound

$$\mathrm{KL} = \log p(\mathbf{y}) - \mathbb{E}_{q\left(f_{(\mathrm{X})}, f_{(\mathrm{Z})}\right)}\left[\log \frac{\prod_n p\left(y_n | f_{(\mathbf{x}_n)}\right) \cancel{p\left(f_{(\mathrm{X})} | f_{(\mathrm{Z})}\right)} p\left(f_{(\mathrm{Z})}\right)}{\cancel{p\left(f_{(\mathrm{X})} | f_{(\mathrm{Z})}\right)} q\left(f_{(\mathrm{Z})}\right)}\right]$$

$$\therefore \mathcal{L} = \sum_n \mathbb{E}_{p\left(f_{(\mathbf{x}_n)} | f_{(\mathrm{Z})}\right) q\left(f_{(\mathrm{Z})}\right)}\left[\log p\left(y_n | f_{(\mathbf{x}_n)}\right)\right] - \mathrm{KL}\left[q\left(f_{(\mathrm{Z})}\right) || p\left(f_{(\mathrm{Z})}\right)\right]$$

Variational lower bound

$$\mathrm{KL} = \log p(\mathbf{y}) - \mathbb{E}_{q\left(f_{(\mathrm{X})}, f_{(\mathrm{Z})}\right)}\left[\log \frac{\prod_n p\left(y_n | f_{(\mathbf{x}_n)}\right) \cancel{p\left(f_{(\mathrm{X})} | f_{(\mathrm{Z})}\right)} p\left(f_{(\mathrm{Z})}\right)}{\cancel{p\left(f_{(\mathrm{X})} | f_{(\mathrm{Z})}\right)} q\left(f_{(\mathrm{Z})}\right)}\right]$$

$$\therefore \mathcal{L} = \sum_n \mathbb{E}_{p\left(f_{(\mathbf{x}_n)} | f_{(\mathrm{Z})}\right) q\left(f_{(\mathrm{Z})}\right)}\left[\log p\left(y_n | f_{(\mathbf{x}_n)}\right)\right] - \mathrm{KL}\left[q\left(f_{(\mathrm{Z})}\right) || p\left(f_{(\mathrm{Z})}\right)\right]$$

- Maximising $\mathcal{L}$ minimises the KL (gets us closer to the true posterior).
- Analytically tractable (with 1D quadrature)

Variational lower bound

$$\mathrm{KL} = \log p(\mathbf{y}) - \mathbb{E}_{q\left(f_{(\mathrm{X})}, f_{(\mathrm{Z})}\right)}\left[\log \frac{\prod_n p\left(y_n | f_{(\mathbf{x}_n)}\right) \cancel{p\left(f_{(\mathrm{X})} | f_{(\mathrm{Z})}\right)} p\left(f_{(\mathrm{Z})}\right)}{\cancel{p\left(f_{(\mathrm{X})} | f_{(\mathrm{Z})}\right)} q\left(f_{(\mathrm{Z})}\right)}\right]$$

$$\therefore \mathcal{L} = \sum_n \mathbb{E}_{p\left(f_{(\mathbf{x}_n)} | f_{(\mathrm{Z})}\right) q\left(f_{(\mathrm{Z})}\right)}\left[\log p\left(y_n | f_{(\mathbf{x}_n)}\right)\right] - \mathrm{KL}\left[q\left(f_{(\mathrm{Z})}\right) || p\left(f_{(\mathrm{Z})}\right)\right]$$

- Maximising $\mathcal{L}$ minimises the KL (gets us closer to the true posterior).
- Analytically tractable (with 1D quadrature)
- Low-rank approximate GP (only needs $M \times M$ inversion)

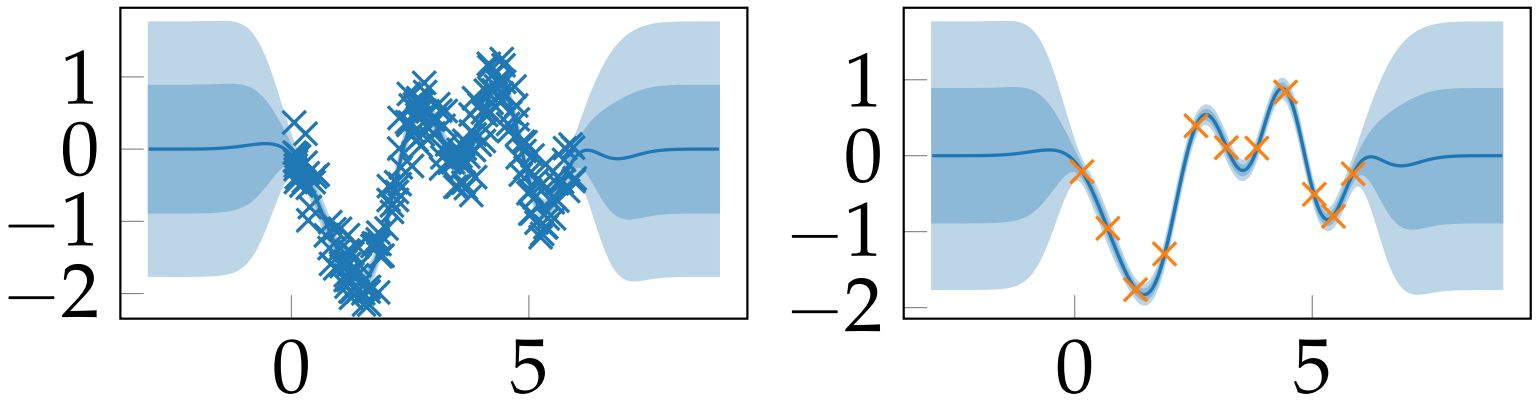
Variational lower bound

$$\mathrm{KL} = \log p(\mathbf{y}) - \mathbb{E}_{q\left(f_{(\mathrm{X})}, f_{(\mathrm{Z})}\right)}\left[\log \frac{\prod_n p\left(y_n | f_{(\mathbf{x}_n)}\right) \cancel{p\left(f_{(\mathrm{X})} | f_{(\mathrm{Z})}\right)} p\left(f_{(\mathrm{Z})}\right)}{\cancel{p\left(f_{(\mathrm{X})} | f_{(\mathrm{Z})}\right)} q\left(f_{(\mathrm{Z})}\right)}\right]$$

$$\therefore \mathcal{L} = \sum_n \mathbb{E}_{p\left(f_{(\mathbf{x}_n)} | f_{(\mathrm{Z})}\right) q\left(f_{(\mathrm{Z})}\right)}\left[\log p\left(y_n | f_{(\mathbf{x}_n)}\right)\right] - \mathrm{KL}\left[q\left(f_{(\mathrm{Z})}\right) || p\left(f_{(\mathrm{Z})}\right)\right]$$

- Maximising $\mathcal{L}$ minimises the KL (gets us closer to the true posterior).
- Analytically tractable (with 1D quadrature)
- Low-rank approximate GP (only needs $M \times M$ inversion)

What should we do in practice?



Practical training requires finding θ, Z, μ, Σ by maximising $\mathcal{L}(\theta, Z, \mu, \Sigma)$.

1. How to initialise the parameters? How many inducing points?
2. How should we optimise $\mathcal{L}$?
3. How small can we make the KL gap to the true posterior?

Overview

The What & Why of Gaussian Processes

Variational Inference for Big Data

Theory and Practice of Training Variational GPs

Robust Conjugate Gradient Methods

Wrap-up

Two methods

Approach 1 (Titsias, 2009):

- Notice that the optimisation for μ, Σ is quadratic, so solve for this in closed form.
- This defines a new bound $\mathcal{L}'(\theta, Z)$, with cost $O(NM^2)$.
- Gradient-based optimisation w.r.t. θ and perhaps Z, using BFGS.

Approach 2 (Hensman et al., 2013):

- Compute $\mathcal{L}$ and its gradients on a minibatch, with cost $O(BM^2 + M^3)$.
- Optimise θ, Z, μ, Σ with stochastic optimiser (e.g. Adam).

Training guidelines

Before our work, the guidelines were:

- Initialise Z by K-means or a random subset of the training inputs.
- List of "tricks" to try (human in the loop).
- **No guarantees** on how many inducing points are needed.

In practice:

- Optimisation would never converge. Waiting $10\times$ longer would always give marginally better results.
- We never got *very* close to the true posterior with a sparse model.
- Approach 2 worked better, because you could do many iterations more quickly.

What was the problem?

- Problem existed in both approaches, so the problem wasn't μ, Σ.
- $\implies$ Problem must be selection of inducing inputs Z!

Draw on board: Impact of selecting bad inducing points (far from data, nearby each other).

Theory to the Rescue

Rates of Convergence for Sparse Variational Gaussian Process Regression

David R. Burt [1] **Carl Edward Rasmussen** [1 2] **Mark van der Wilk** [2]

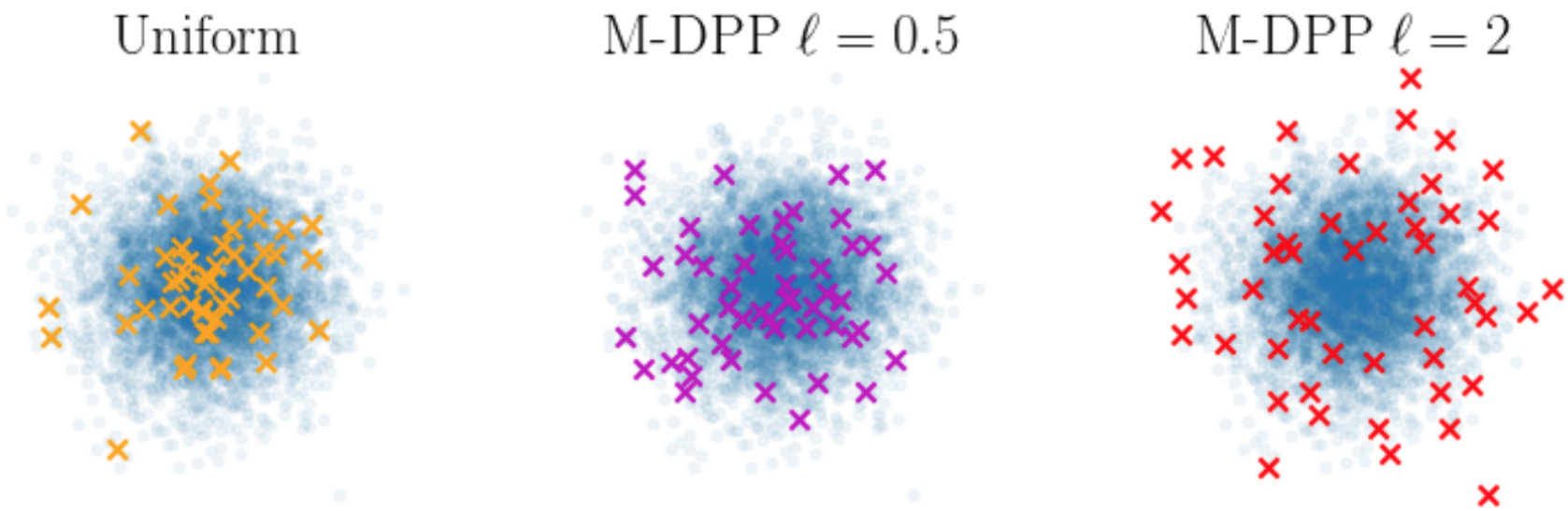
- We set out to analyse how many inducing inputs were needed to get KL $\to 0$ as $N \to \infty$.
- Could not prove bounds for uniform subsampling.
- *Could* prove bounds for DPP selection!

Result (assuming inputs in a bounded region):

- Select inducing inputs Z with (approximate) M-DPP.
- We give a rate for M to increase with N, so that KL $\to 0$.
- Reasonable computational cost, e.g. $O(N(\log N)^{2D}(\log\log N)^2))$ for SqExp, barely above linear[1]

[1] Recall that $O(N(\log N)^D) = O(N^{1+\epsilon})$ for any $D \in \mathbb{N}$ and $\epsilon > 0$, and that D is fixed in our problem.

Theory to the Rescue



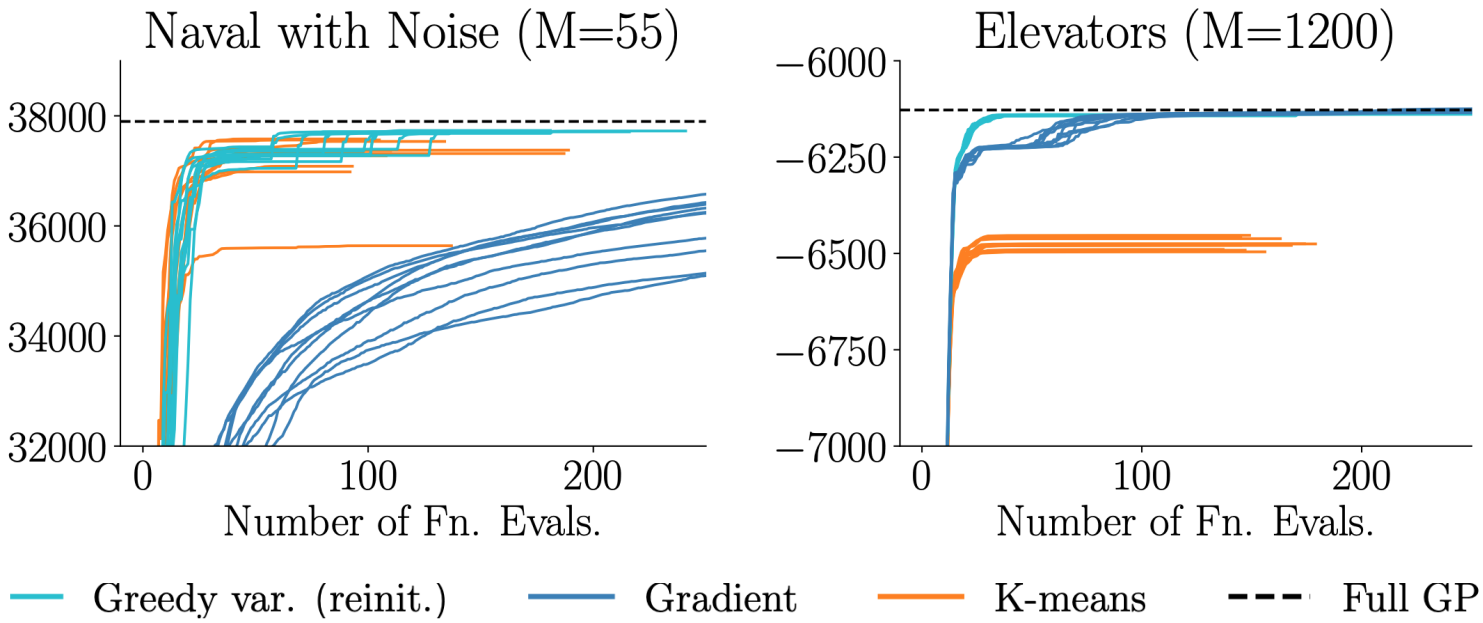
- *M*-DPP depends on kernel of the GP.
- *M*-DPP initialisation leads to more equally spread out inducing points than uniform.
- Theory says **you don't need to optimise inducing inputs**.

This suggests a different training procedure!

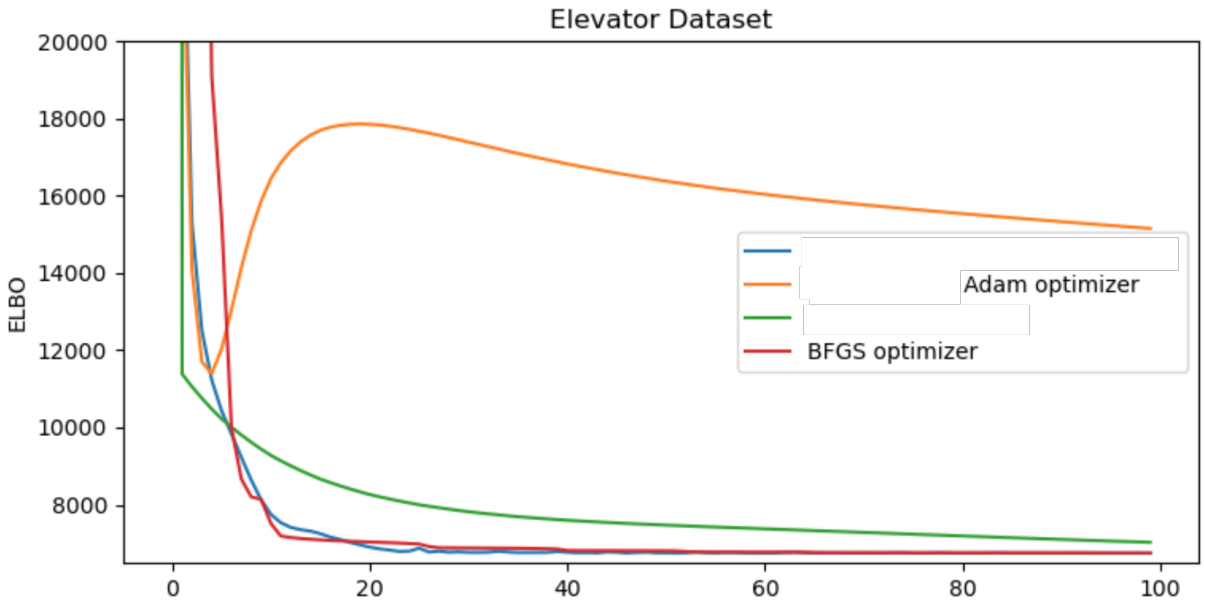
New Training Procedure

In the journal version (Burt et al., 2020) we recommend:

1. Use Approach 1 (Titsias, 2009), so you don't have to worry about optimising μ, Σ.
2. Initialise inducing inputs with M-DPP approximation ("greedy variance" method).
3. Use BFGS to optimise *only* hyperparameters θ.
4. Once BFGS converges[2], repeat from step 2.



What about minibatching?



Courtesy of Maëlhann Rozé, MEng student

If you correctly initialise your inducing points:

- Stochastic optimisation is supposed to be faster than full batch... but isn't!
- Adam becomes a terrible stochastic optimiser for GPs!
- Past recommendations for SVGP were likely due to **all** methods performing sub-optimally due to bad inducing input placement.

Takeaways

- We have a *proof* that inducing points are *arbitrarily exact* at *reasonable cost* as $N \to \infty$.
- This leads to a new training procedure which **reliably** obtains **better** results.
- Particularly important if you care about getting every last bit of performance out of your method.

Is GP inference solved?

Takeaways

- We have a *proof* that inducing points are *arbitrarily exact* at *reasonable cost* as $N \to \infty$.
- This leads to a new training procedure which **reliably** obtains **better** results.
- Particularly important if you care about getting every last bit of performance out of your method.

Is GP inference solved? **No!**

- Given a dataset of fixed size, required number of inducing points may still be too large.

Takeaways

- We have a *proof* that inducing points are *arbitrarily exact* at *reasonable cost* as $N \to \infty$.
- This leads to a new training procedure which **reliably** obtains **better** results.
- Particularly important if you care about getting every last bit of performance out of your method.

Is GP inference solved? **No!**

- Given a dataset of fixed size, required number of inducing points may still be too large.
- Inducing points may not give the best speed-accuracy trade-off.

Takeaways

- We have a *proof* that inducing points are *arbitrarily exact* at *reasonable cost* as $N \to \infty$.
- This leads to a new training procedure which **reliably** obtains **better** results.
- Particularly important if you care about getting every last bit of performance out of your method.

Is GP inference solved? **No!**

- Given a dataset of fixed size, required number of inducing points may still be too large.
- Inducing points may not give the best speed-accuracy trade-off.
- Bounded/iid inputs are only one special case.

Takeaways

- We have a *proof* that inducing points are *arbitrarily exact* at *reasonable cost* as $N \to \infty$.
- This leads to a new training procedure which **reliably** obtains **better** results.
- Particularly important if you care about getting every last bit of performance out of your method.

Is GP inference solved? **No!**

- Given a dataset of fixed size, required number of inducing points may still be too large.
- Inducing points may not give the best speed-accuracy trade-off.
- Bounded/iid inputs are only one special case.
- We need to make running this **super easy** for people.

Overview

The What & Why of Gaussian Processes

Variational Inference for Big Data

Theory and Practice of Training Variational GPs

Robust Conjugate Gradient Methods

Wrap-up

Conjugate Gradients

Alternative approach to inducing points. Tries to find $\mathbf{K}^{-1}\mathbf{v}$ by solving

$$\underset{\mathbf{x}}{\operatorname{argmin}} \frac{1}{2}\mathbf{x}^\top \mathbf{K}\mathbf{x} - \mathbf{x}^\top \mathbf{b} \tag{5}$$

- **Conjugate Gradients** gives iterative solution that is exact in the limit.
- May give better speed-accuracy trade-off than inducing points.
- Has given genuinely impressive results[3]:

[3] The results and the scale are genuinely impressive. However I disagree that they can be called exact.

Conjugate Gradients

Alternative approach to inducing points. Tries to find $\mathbf{K}^{-1}\mathbf{v}$ by solving

$$\underset{\mathbf{x}}{\operatorname{argmin}} \frac{1}{2}\mathbf{x}^\top \mathbf{K}\mathbf{x} - \mathbf{x}^\top \mathbf{b} \tag{5}$$

- **Conjugate Gradients** gives iterative solution that is exact in the limit.
- May give better speed-accuracy trade-off than inducing points.
- Has given genuinely impressive results[3]:

Exact Gaussian Processes on a Million Data Points

Ke Alexander Wang[1*] **Geoff Pleiss**[1*] **Jacob R. Gardner**[2]
Stephen Tyree[3] **Kilian Q. Weinberger**[1] **Andrew Gordon Wilson**[1,4]
[1]Cornell University, [2]Uber AI Labs, [3]NVIDIA, [4]New York University

[3]The results and the scale are genuinely impressive. However I disagree that they can be called exact.

Conjugate Gradients

One should expect the training procedure of an “exact” method, to also be “exact”.

Conjugate Gradients

One should expect the training procedure of an "exact" method, to also be "exact".
Open questions about CG approach of Wang et al. (2019):

Conjugate Gradients

One should expect the training procedure of an "exact" method, to also be "exact".

Open questions about CG approach of Wang et al. (2019):

- How is (the convergence of) hyperparameters affected by error in the estimation of the gradients?

Conjugate Gradients

One should expect the training procedure of an "exact" method, to also be "exact".

Open questions about CG approach of Wang et al. (2019):

- How is (the convergence of) hyperparameters affected by error in the estimation of the gradients?
  - Wang et al. (2019) directly approximate the gradients with CG. With error introduced, it is not clear whether following them will lead to convergence.

Conjugate Gradients

One should expect the training procedure of an "exact" method, to also be "exact".

Open questions about CG approach of Wang et al. (2019):

- How is (the convergence of) hyperparameters affected by error in the estimation of the gradients?
  - Wang et al. (2019) directly approximate the gradients with CG. With error introduced, it is not clear whether following them will lead to convergence.
- What convergence tolerances should be used to obtain good accuracy-speed tradeoff?

Conjugate Gradients

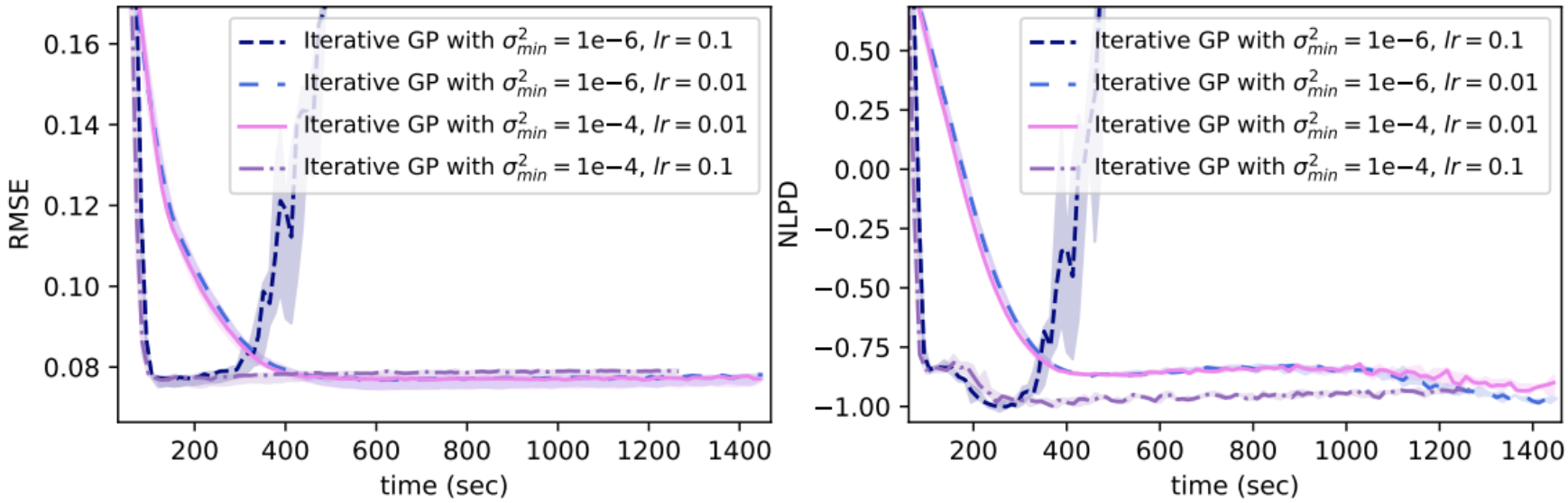
One should expect the training procedure of an "exact" method, to also be "exact".

Open questions about CG approach of Wang et al. (2019):

- How is (the convergence of) hyperparameters affected by error in the estimation of the gradients?
  - Wang et al. (2019) directly approximate the gradients with CG. With error introduced, it is not clear whether following them will lead to convergence.
- What convergence tolerances should be used to obtain good accuracy-speed tradeoff?
  - An "arbitrarily exact" method would require a well-defined rule, a proof, and an asymptotic computational cost analysis, as inducing points currently has (Burt et al., 2019, 2020)

Conjugate Gradients

This has practical consequences, with behaviour that you would not expect from an exact method:



This said, CG is a good idea, and inspired by the results of Wang et al. (2019), we (+ Artem Artemev, David Burt) propose some improvements.

Conjugate Gradient Lower Bound

- VI measures quality of approximation and hyperparameters in a single objective: The ELBO.

Conjugate Gradient Lower Bound

- VI measures quality of approximation and hyperparameters in a single objective: The ELBO.
- Makes it easy to design convergent algorithms: Just optimise the ELBO!

Conjugate Gradient Lower Bound

- VI measures quality of approximation and hyperparameters in a single objective: The ELBO.
- Makes it easy to design convergent algorithms: Just optimise the ELBO!
- Q: How to set parameters? A: Just optimise the ELBO!

Conjugate Gradient Lower Bound

- VI measures quality of approximation and hyperparameters in a single objective: The ELBO.
- Makes it easy to design convergent algorithms: Just optimise the ELBO!
- Q: How to set parameters? A: Just optimise the ELBO!

Conjugate Gradient Lower Bound

- VI measures quality of approximation and hyperparameters in a single objective: The ELBO.
- Makes it easy to design convergent algorithms: Just optimise the ELBO!
- Q: How to set parameters? A: Just optimise the ELBO!

Can we do something similar for Conjugate Gradient methods?

Conjugate Gradient Lower Bound

- VI measures quality of approximation and hyperparameters in a single objective: The ELBO.
- Makes it easy to design convergent algorithms: Just optimise the ELBO!
- Q: How to set parameters? A: Just optimise the ELBO!

Can we do something similar for Conjugate Gradient methods? **Yes!**

Conjugate Gradient Lower Bound

- VI measures quality of approximation and hyperparameters in a single objective: The ELBO.
- Makes it easy to design convergent algorithms: Just optimise the ELBO!
- Q: How to set parameters? A: Just optimise the ELBO!

Can we do something similar for Conjugate Gradient methods? **Yes!**

We develop the **Conjugate Gradient Lower Bound** (CGLB).

Conjugate Gradient Lower Bound

- VI measures quality of approximation and hyperparameters in a single objective: The ELBO.
- Makes it easy to design convergent algorithms: Just optimise the ELBO!
- Q: How to set parameters? A: Just optimise the ELBO!

Can we do something similar for Conjugate Gradient methods? **Yes!**

We develop the **Conjugate Gradient Lower Bound** (CGLB).

- The partial solution $\mathbf{v}$ to the CG optimisation problem to find $\mathbf{K}^{-1}\mathbf{y}$ is unified with the hyperparameter objective
  $\implies$ easier to guarantee convergence

$$\boldsymbol{\theta}^*, \mathbf{v}^* = \text{argmax}_{\boldsymbol{\theta},\mathbf{v}}\, L(\boldsymbol{\theta}, \mathbf{v}), \text{ with } \mathbf{v}^* = \text{argmax}_{\mathbf{v}}\, L(\boldsymbol{\theta}, \mathbf{v}) = \mathbf{K}^{-1}\mathbf{y}\,, \forall \boldsymbol{\theta}.$$

Conjugate Gradient Lower Bound

- VI measures quality of approximation and hyperparameters in a single objective: The ELBO.
- Makes it easy to design convergent algorithms: Just optimise the ELBO!
- Q: How to set parameters? A: Just optimise the ELBO!

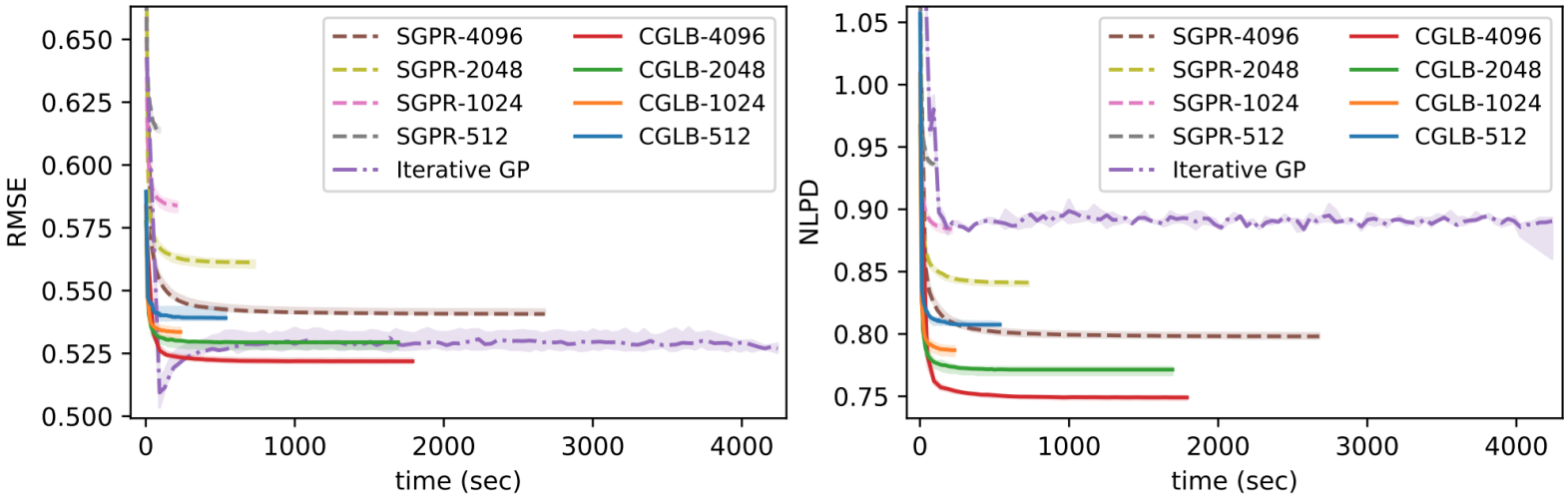
Can we do something similar for Conjugate Gradient methods? **Yes!**

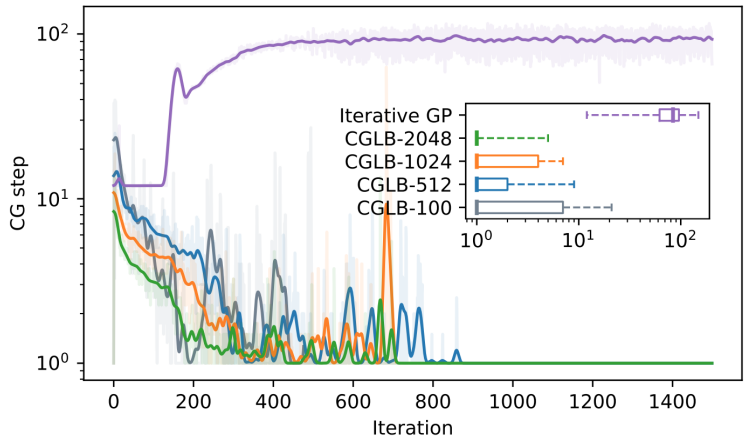
We develop the **Conjugate Gradient Lower Bound** (CGLB).

- The partial solution $\mathbf{v}$ to the CG optimisation problem to find $\mathbf{K}^{-1}\mathbf{y}$ is unified with the hyperparameter objective
  $\implies$ easier to guarantee convergence
- Additional upper bound to automatically determine number of CG iterations.

$\boldsymbol{\theta}^*, \mathbf{v}^* = \text{argmax}_{\boldsymbol{\theta},\mathbf{v}} L(\boldsymbol{\theta}, \mathbf{v})$, with $\mathbf{v}^* = \text{argmax}_{\mathbf{v}} L(\boldsymbol{\theta}, \mathbf{v}) = \mathbf{K}^{-1}\mathbf{y}\,, \forall \boldsymbol{\theta}$.

Conjugate Gradient Lower Bound





- Fewer iterations of CG $\implies$ faster.
- More guarantees for optimisation $\implies$ better performance.

Takeaways

- If you don't manually adjust parameters in "Iterative GP" (Wang et al., 2019), you can get sub-optimal performance.
- We (Artemev et al., 2021) introduce a CG-based approximation where **all parameters can be automatically set through optimisation**.
- This gives better and faster predictions, without requiring human intervention.

Overview

The What & Why of Gaussian Processes

Variational Inference for Big Data

Theory and Practice of Training Variational GPs

Robust Conjugate Gradient Methods

Wrap-up

Conclusion

- Inducing point methods have theoretically-backed methods for selecting all parameters.
- With our new training methods, inducing point methods can be **highly accurate**.
- There are cases though, where sparse approximations simply do not exist.

- Conjugate Gradient methods can help in these situations.
- Be careful with existing methods, even if branded as "exact" (Wang et al., 2019).
- We (Artemev et al., 2021) introduce automatic solutions, with better performance.

Future work

There is a lot more to do!

- Goal: Make GP approximations completely transparent.
  E.g. we don't want to need to manually choose between inducing points or CG. Software is needed for *real* impact.

Future work

There is a lot more to do!

- Goal: Make GP approximations completely transparent.
  E.g. we don't want to need to manually choose between inducing points or CG. Software is needed for *real* impact.
- Need better stochastic optimisation for GPs
  This is necessary for truly big data, or if we want to combine them with deep learning.

Future work

There is a lot more to do!

- Goal: Make GP approximations completely transparent.
  E.g. we don't want to need to manually choose between inducing points or CG. Software is needed for *real* impact.
- Need better stochastic optimisation for GPs
  This is necessary for truly big data, or if we want to combine them with deep learning.
- Deep GPs and DNNs may converge! Recent work: Dutordoir et al. (2021)

Future work

There is a lot more to do!

- Goal: Make GP approximations completely transparent.
  E.g. we don't want to need to manually choose between inducing points or CG. Software is needed for *real* impact.
- Need better stochastic optimisation for GPs
  This is necessary for truly big data, or if we want to combine them with deep learning.
- Deep GPs and DNNs may converge! Recent work: Dutordoir et al. (2021)

I'm currently growing a team at Imperial to work on this, and I'm looking for grants, or industry support for grant applications.

Summary

Some papers I contributed to on robust automatic GPs

- **Understanding Probabilistic Sparse Gaussian Process Approximations**
  Bauer et al. (2016), NeurIPS
- **GPflow: A Gaussian process library using TensorFlow**
  Matthews et al. (2017), JMLR
- **Convergence of Sparse Variational Inference in GPR**
  Burt et al. (2019, 2020), ICML, JMLR
- **Tighter Bounds on the LML of GPR Using CG**
  Artemev et al. (2021), ICML
- **Software for automatic & robust training of GPs**
  Work in Progress

References I

Artemev, A., Burt, D. R., and van der Wilk, M. (2021). Tighter bounds on the log marginal likelihood of gaussian process regression using conjugate gradients.

Bauer, M. S., van der Wilk, M., and Rasmussen, C. E. (2016). Understanding probabilistic sparse gaussian process approximations. In Advances in neural information processing systems.

Burt, D., Rasmussen, C. E., and Van Der Wilk, M. (2019). Rates of convergence for sparse variational Gaussian process regression. In Chaudhuri, K. and Salakhutdinov, R., editors, Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pages 862–871. PMLR.

Burt, D. R., Rasmussen, C. E., and van der Wilk, M. (2020). Convergence of sparse variational inference in gaussian processes regression. Journal of Machine Learning Research, 21(131):1–63.

References II

Dutordoir, V., Hensman, J., van der Wilk, M., Ek, C. H., Ghahramani, Z., and Durrande, N. (2021). Deep neural networks as point estimates for deep gaussian processes.

Hensman, J., Fusi, N., and Lawrence, N. D. (2013). Gaussian processes for big data. In Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI), pages 282–290.

Matthews, A. G. d. G., van der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. (2017). GPflow: A Gaussian process library using TensorFlow. Journal of Machine Learning Research, 18(40):1–6.

Titsias, M. K. (2009). Variational learning of inducing variables in sparse Gaussian processes. In Proceedings of the 12th International Conference on Artificial Intelligence and Statistics, pages 567–574.

References III

Wang, K., Pleiss, G., Gardner, J., Tyree, S., Weinberger, K. Q., and Wilson, A. G. (2019). Exact gaussian processes on a million data points. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc.