


# Challenges for Gaussian processes

**Mark van der Wilk**

Department of Computing  
Imperial College London

@markvanderwilk  
m.vdwilk@imperial.ac.uk

February 4, 2022

# Methods for Regression in ML

Most tasks in ML are regression! Deep learning is 99% regression!

Remember where GPs excel:

- ▶ are low dimensional (e.g. tens of dimensions rather than 100s),
- ▶ have little data (or data is expensive to obtain),
- ▶ are noisy (random fluctuations that obscure the signal),
- ▶ require uncertainty estimates.

Today: Where GPs struggle, and why.

# Learning objectives

Know how to perform the computations necessary for GPs

- ▶ Computing the marginal likelihood
- ▶ Computational complexity
- ▶ Low-rank approximations

Understand the computational and modelling limitations of GPs

- ▶ Limitations of stationary kernels
- ▶ Limitations of local kernels in high-dimensions

# Training a GP: Computations

To train, we need the marginal likelihood and its gradient:

$$\log p(\mathbf{y} | \theta) = \text{const} - \frac{1}{2} \log |\mathbf{K}_\theta + \sigma^2 \mathbf{I}| - \frac{1}{2} \mathbf{y}^\top (\mathbf{K}_\theta + \sigma^2 \mathbf{I})^{-1} \mathbf{y} \quad (1)$$

$$\frac{\partial}{\partial \theta} \log p(\mathbf{y} | \theta) = \frac{1}{2} \text{Tr} \left[ \left( \mathbf{K}_\theta^{-1} \mathbf{y} \mathbf{y}^\top \mathbf{K}_\theta - \mathbf{K}_\theta^{-1} \right) \frac{\partial \mathbf{K}_\theta}{\partial \theta} \right] \quad (2)$$

$(\dots)^{-1}$  and  $|\dots|$  are calculated from a **matrix decomposition**.

- ▶ Decompositions are expensive
- ▶ But make follow-on operations cheap
- ▶ The correct decomposition helps numerical stability<sup>1</sup>
- ▶ Directly computing the inverse (`np.linalg.inv()`) is a bad thing to do!

---

<sup>1</sup><https://nhigham.com/2020/08/04/what-is-numerical-stability/>

# Eigenvalue decomposition

We can compute the terms using the **eigenvalue decomposition**:

$$\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top = \mathbf{K}_\theta + \sigma^2\mathbf{I} \quad (3)$$

$$\log|\mathbf{K}_\theta + \sigma^2\mathbf{I}| = \log|\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top| = \log|\mathbf{Q}| + \log|\mathbf{\Lambda}| = \sum_{n=1}^N \log \lambda_n(\mathbf{K}_\theta + \sigma^2\mathbf{I}) \quad (4)$$

$$\mathbf{y}^\top (\mathbf{K}_\theta + \sigma^2\mathbf{I})^{-1} \mathbf{y} = \mathbf{y}^\top \mathbf{Q}^\top \mathbf{\Lambda}^{-1} \mathbf{Q} \mathbf{y} \quad (5)$$

Eigenvalue decomposition is mostly applied for **theoretical analysis**.

# Cholesky decomposition

Or alternatively the **Cholesky decomposition**:

$$\mathbf{L}\mathbf{L}^\top = \mathbf{K}_\theta + \sigma^2\mathbf{I}, \quad \text{where } \mathbf{L} \text{ is lower triangular.} \quad (6)$$

$$\log|\mathbf{K}_\theta + \sigma^2\mathbf{I}| = \log|\mathbf{L}\mathbf{L}^\top| = 2 \sum_{n=1}^N \log[\mathbf{L}]_{nn} \quad (7)$$

$$\mathbf{y}^\top (\mathbf{K}_\theta + \sigma^2\mathbf{I})^{-1} \mathbf{y} = \mathbf{y}^\top \mathbf{L}^\top{}^{-1} \mathbf{L}^{-1} \mathbf{y} \quad (8)$$

Cholesky decomposition is used for **practical implementation**.  
(For the coursework, all implementations are fine.)

# Computational complexity

1. Computing kernel matrix  $O(N^2)$  time and  $O(N^2)$  space
2. Eigendecomposition and Cholesky are both often quoted to be  $O(N^3)$  time.<sup>2</sup>
3. Logdet or inverse given the decomposition are fast
  - ▶ Logdet for both are  $O(N)$
  - ▶ Inverse is  $O(N^2)$

Can we take advantage of structure in the kernel matrix to do better?

---

<sup>2</sup>Algorithms do exist with better asymptotic complexity, but they are definitely slower than  $O(N^2)$ .

## Low-rank kernels

For kernel matrices that are low-rank (have some zero eigenvalues),  
i.e. with  $M \ll N$

$$\mathbf{K} = \mathbf{P}\mathbf{P}^\top, \quad \text{where } \mathbf{K} \in \mathbb{R}^{N \times N}, \mathbf{P} \in \mathbb{R}^{N \times M} \quad (9)$$

we can compute the marginal likelihood more cheaply:<sup>3</sup>

$$\log|\mathbf{K} + \sigma^2\mathbf{I}_N| = \log|\sigma^2\mathbf{I}_N| + \log|\mathbf{I}_M + \sigma^{-2}\mathbf{P}^\top\mathbf{P}| \quad (10)$$

$$\mathbf{y}^\top(\mathbf{K} + \sigma^2\mathbf{I}_N)^{-1}\mathbf{y} = \mathbf{y}^\top\left(\sigma^{-2}\mathbf{I}_N - \sigma^{-2}\mathbf{P}(\sigma^2\mathbf{I}_M + \mathbf{P}^\top\mathbf{P})^{-1}\mathbf{P}^\top\right)\mathbf{y} \quad (11)$$

Can be computed in  $O(NM^2 + M^3)$ .

---

<sup>3</sup>Can be proved by applying the Woodbury Matrix Identity, and the similar Matrix Determinant Lemma.



# Decaying eigenvalues

For example, the squared exponential kernel

$$k(x, x') = \sigma_f^2 \exp\left(\frac{|x - x'|^2}{2\ell^2}\right), \quad \theta = \{\sigma_f, \ell\} \quad (12)$$

with 1D inputs i.i.d. from<sup>4</sup>  $\mathcal{N}(0, \sigma_D^2)$ , we have that the eigenvalues of  $\mathbf{K}$  decay exponentially

$$\lambda_m = \sqrt{\frac{2a}{A}} B^m \quad \text{with } B < 1 \quad (13)$$

$$a^{-1} = 4\sigma_D^2, \quad b^{-1} = 2\ell^2, \quad c = \sqrt{a^2 + 2ab}, \quad A = a + b + c, \quad B = b/A$$

as the number of data  $N \rightarrow \infty$ .

---

<sup>4</sup>Conditions can be much weaker, but this is easier.

# Low-rank approximations

So we know that for large  $N$  our  $\mathbf{K}$  will be **approximately low-rank**.  
Can we approximate it with a low-rank matrix?

$$\mathbf{K} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top = [\mathbf{Q}_1 \quad \mathbf{Q}_2] \begin{bmatrix} \mathbf{\Lambda}_1 & \\ & \mathbf{\Lambda}_2 \end{bmatrix} \begin{bmatrix} \mathbf{Q}_1^\top \\ \mathbf{Q}_2^\top \end{bmatrix} = \mathbf{Q}_1\mathbf{\Lambda}_1\mathbf{Q}_1^\top + \mathbf{Q}_2\mathbf{\Lambda}_2\mathbf{Q}_2^\top \quad (14)$$

$$\hat{\mathbf{K}} = \mathbf{Q}_1\mathbf{\Lambda}_1\mathbf{Q}_1^\top + \cancel{\mathbf{Q}_2\mathbf{\Lambda}_2\mathbf{Q}_2^\top} \overset{\approx 0}{=} \mathbf{P}\mathbf{P}^\top, \quad \text{with } \mathbf{P} = \mathbf{Q}_1\mathbf{\Lambda}_1^{\frac{1}{2}}. \quad (15)$$

This gives the approximation with the minimal Frobenius norm:

$$\|\mathbf{K} - \hat{\mathbf{K}}\|_F = \sqrt{\sum_{i=M+1}^N \lambda_i} \quad (16)$$

# Low-rank approximation: Data fit

Starting with the data-fit term:<sup>5</sup>

$$\begin{aligned} \mathbf{y}^\top (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{y} &= \mathbf{y}^\top (\hat{\mathbf{K}} + \sigma^2 \mathbf{I} + \mathbf{K} - \hat{\mathbf{K}})^{-1} \mathbf{y} \\ &= \mathbf{y}^\top (\hat{\mathbf{K}} + \sigma^2 \mathbf{I})^{-1} \mathbf{y} - \underbrace{\mathbf{y}^\top (\hat{\mathbf{K}} + \sigma^2 \mathbf{I})^{-1} (\mathbf{K} - \hat{\mathbf{K}}) (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{y}}_{\rightarrow 0 \text{ as } \|\mathbf{K} - \hat{\mathbf{K}}\|_F \rightarrow 0} \end{aligned} \quad (17)$$

You can place bounds on the error using knowledge of  $\lambda_M$  (the final included eigenvalue).

---

<sup>5</sup>Applying the identity  $(A + B)^{-1} = A^{-1} - A^{-1}B(A + B)^{-1}$ .

# Low-rank approximation: Complexity penalty

And finishing with the complexity penalty:

$$\begin{aligned}\log|\mathbf{K} + \sigma^2\mathbf{I}| &= \sum_{n=1}^N \log(\lambda_n(\mathbf{K}) + \sigma^2) \\ &= \sum_{m=1}^M \log(\lambda_m(\mathbf{K}) + \sigma^2) + \sum_{r=M+1}^N \underbrace{\log(\lambda_r(\mathbf{K}) + \sigma^2)}_{\approx \sigma^2} \quad (18)\end{aligned}$$

$$\approx \sum_{m=1}^M \log(\lambda_m(\mathbf{K}) + \sigma^2) + \sum_{r=M+1}^N \log(\sigma^2) \quad (19)$$

$$= \log|\hat{\mathbf{K}} + \sigma^2\mathbf{I}| \quad (20)$$

# Low-rank approximations overview

- ▶ Learning with exact matrix decompositions is expensive —  $O(N^3)$  time,  $O(N^2)$  memory
- ▶ Low-rank kernels improve things when  $N \gg M$  —  $O(NM^2)$  time,  $O(NM)$  memory
- ▶ Low-rank approximations do exist!
- ▶ We used an impractical method (eigendecomposition) to find **which** low-rank columns to use, with a cost of  $O(N^2M)$  time.

For certain approximations, we have 1) fast methods for finding  $\mathbf{P}$ , and 2) proofs for how large  $M$  needs to be.

E.g. Burt et al [2019] show that for regression and the Squared Exponential kernel, we can get arbitrarily good approximations in

$$O(N(\log N)^{2D}(\log \log N)^2) \quad (21)$$

$$O(N(\log N)^{2D}(\log \log N)^2) = O(N^{1+\epsilon}), \quad \forall \epsilon > 0 \quad (22)$$

# Practical constraints and implementation

GPs are slow, but often not even because of asymptotic complexity!

- ▶ Memory is often the main bottleneck (storing large matrices).  
This is poorly suited to modern-day hardware (i.e. GPUs)
- ▶ Decompositions require serial computations.  
This is poorly suited to modern-day hardware (i.e. GPUs)
- ▶ Decompositions need high floating point precision.  
This is poorly suited to modern-day hardware (i.e. GPUs)

# There is hope

Approximations can make the computations look more similar to a neural network by

- ▶ Reduce the size of the matrix decomposition [Titsias, 2009]
- ▶ Improve data parallelism [Hensman et al, 2013]
- ▶ Increase reliance on fast matrix-matrix multiplications [Gardner et al, 2018; Wang et al, 2019]
- ▶ Remove matrix inverses altogether [van der Wilk et al, 2019]

This slide for reference only, not examinable.

# Implementation matters!

We have good asymptotic bounds on the computational complexity!

Constants cause slowness  $\implies$  implementation matters

E.g. *Exact Gaussian Processes on a Million Data Points*, Wang et al [2019] tailors a very good approximation to GPU hardware, with impressive results.



# Conclusions on Computational Issues

- ▶ Gaussian processes are computationally expensive to train
- ▶ Low-rank approximations to the kernel matrix  $\mathbf{K}$  do exist
- ▶ Low-rank approximations improve the asymptotic complexity
- ▶ Still need work to optimise algorithms and tailor to modern hardware

# Modelling limitations

We know that the generalisation capability of a model depends strongly on the properties of its prior.

Do we really have good priors?

Some criticisms...

# Stationary kernels

We have seen many examples of **stationary** kernels:

$$k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{x}') \quad (23)$$

e.g. the squared exponential.

- ▶ Same generalisation characteristic is applied through the entire input space.
- ▶ See `stationary-kernel-failure.ipynb` in the `inference-plots` GitHub repo.

# Local kernels in high dimensions

Many kernels depend on some form of the Euclidean distance:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp \left[ -\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2} \right] \quad (24)$$

This behaves badly in high dimensions (even MNIST,  $D = 28^2 = 784$ ):

- ▶ Small differences in each dimension add up to large distances  
→ low correlation
- ▶ Function is allowed to vary independently along each input dimension
- ▶ Prior with low lengthscale is too flexible, large lengthscale is too inflexible
- ▶ No middle ground

# Conclusions

Gaussian processes have limitations for both

- ▶ low-dimensional inputs (e.g. stationarity)
- ▶ high-dimensional inputs (e.g. locality)

We need to either make much better priors, or incorporate GPs into more complex models.

# References I

- [1] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7576–7586. Curran Associates, Inc., 2018.