Exact Gaussian Processes on a Million Data Points

(NeurIPS, 2019)

Ke Alexander Wang, Geoff Pleiss, Jacob R. Gardner

Eric Han

National University of Singapore

Introduction - Gaussian Processes (GP)

GP is a stochastic process, that is flexible and non-parametric.

Advantages (Pedregosa et al. 2011):

1. Prediction interpolates the observations
2. Prediction is probabilistic - uncertainty representation
3. Modelling versatility - able to specify different kernels

Disadvantages (Pedregosa et al. 2011):

1. Non-sparse, difficult scaling with more data
2. Poor efficiency in higher dimensions

Introduction - Gaussian Processes (GP)

Typical implementation of GP Regression (GPR)

1. Input data X, Y and kernel $\kappa(x, x)$
2. Training – Compute log marginal likelihood for all θ

$$\frac{\partial \mathcal{L}}{\partial \theta} \propto \operatorname{Tr}\left[\left(\boldsymbol{\alpha}\boldsymbol{\alpha}^\top - \boxed{\widehat{\mathbf{K}}_{XX}^{-1}}\right)\frac{\partial \widehat{\mathbf{K}}_{XX}}{\partial \theta}\right], \quad \text{where } \boldsymbol{\alpha} = \boxed{\widehat{\mathbf{K}}_{XX}^{-1}\mathbf{y}}$$

3. Prediction – Compute Posterior

$$\bar{f}_* = \mathbf{k}_*^\top \boxed{\widehat{\mathbf{K}}_{XX}^{-1}\mathbf{y}}, \quad \mathbb{V}\left[\bar{f}_*\right] = \kappa(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top \boxed{\widehat{\mathbf{K}}_{XX}^{-1}\mathbf{k}_*}$$

Problem - Inefficient computation for $\boxed{\text{boxed}}$ parts.
Note: Equation (2) in the paper is incorrect.

Application

GPR are successful in many applications, and many more:

1. Bayesian Optimization
2. Reinforcement Learning
3. Time-series forecasting

But the inefficiency of the $\boxed{\text{boxed}}$ parts, prevent:

1. Scaling to large datasets

Goal of this work

Train GP on over a million data points,
performing predictions without approximations.

Motivation

Matrix Inversion complexity is known to be $O(n^3)$, but different methods solve with different performance and space characteristics.

Practical exact solutions to compute $\boxed{\text{boxed}}$ parts

- LU Decomposition
- Cholesky Decomposition (assume positive-definite)
  - Scikit-Learn (via Numpy, Blas/Lapack)
  - GPy (via Scipy, Blas/Lapack)
  - GPflow (via Tensorflow)

Approximate solutions to reduce the impact of $\boxed{\text{boxed}}$ parts

- Mixture-of-experts
- Inducing points – $O(nm^2)$ time
- Random feature expansions

Effects on prediction error and uncertainty quantification.

Background I

Popular practical exact solutions to solving the $\boxed{\text{boxed}}$ parts:

Cholesky Decomposition

Matrix $\widehat{\mathbf{K}}_{XX}$ is positive-definite, can be factorized

$$\widehat{\mathbf{K}}_{XX} = \mathbf{L}\mathbf{L}^\top, \text{where } \mathbf{L} \text{ is lower triangular}$$

- via distributed computing (Nguyen, Filippone, and Michiardi 2019), but quadratic communication and quadratic memory.
- recursive algorithm, unsuitable for GPU
- very stable, widely used - errors $||\mathbf{E}||_2 \leq c_n \epsilon ||\mathbf{A}||_2$
- Time $O(n^3)$, exact inference limit to $n > 10^4$
- Memory $O(n^2)$ needed to store L

$$L_{jj} = \sqrt{A_{jj} - \sum_{k=1}^{j-1} L_{jk}^2}, \quad L_{ij} = \frac{1}{L_{jj}} \left(A_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk} \right) : i > j$$

Background II

Alternative solutions to solving the $\boxed{\text{boxed}}$ parts - $\widehat{\mathbf{K}}_{XX}^{-1}\mathbf{y} = \mathbf{x}$:

Conjugate Gradients (CG)

Key Idea: Instead of solving for $\widehat{\mathbf{K}}_{XX}^{-1}$ to find $\mathbf{x}$, we find $\mathbf{x}$ directly:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}}\Big(\frac{1}{2}\mathbf{x}^\top\widehat{\mathbf{K}}_{XX}\mathbf{x} - \mathbf{x}^\top\mathbf{y}\Big)$$

- Unique minimizer $\mathbf{x}$ exist due to positive definite $\widehat{\mathbf{K}}_{XX}$
- Iterative algorithm with matrix-vector multiplication
- Iterations can be speed up with preconditioning
- Able to decide tolerence of solution $\epsilon = ||\widehat{\mathbf{K}}_{XX}\mathbf{x}^* - \mathbf{y}||/||\mathbf{y}||$
- Similar to gradient descent, the difference is that CG enforces that the search direction $\mathbf{p}_i$ is conjugate to each other.

Note: In the absence of round-off error, it converge to exact solution after n steps.

Background II

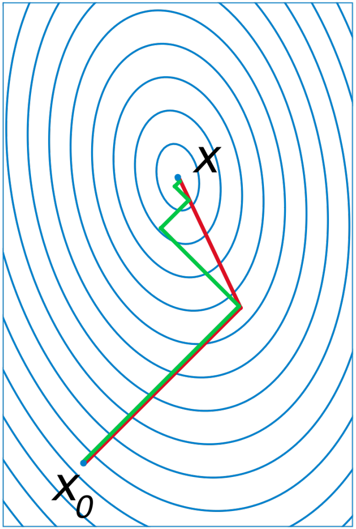
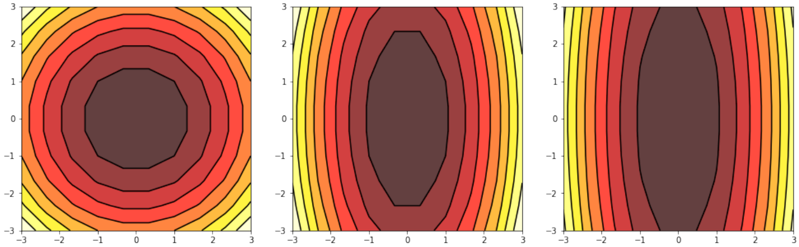


Figure 1: Conjugate Gradients (red) vs Gradient Descent (green)[1]

[1]Image from Conjugate Gradient Method.

Background II

Preconditioned Conjugate Gradients (PCG)



Images from the left[2] - $\kappa(\mathbf{A}) = 1, 5, 10$

Key Idea: Convergence speed to a good enough approximation is determined by condition number $\kappa(\mathbf{A}) = \frac{|\lambda_{\max}(\mathbf{A})|}{|\lambda_{\min}(\mathbf{A})|}$.

$$\widehat{\mathbf{K}}_{XX}\mathbf{x} - \mathbf{y} \quad \rightarrow \quad \mathbf{P}^{-1}(\widehat{\mathbf{K}}_{XX}\mathbf{x} - \mathbf{y})$$

Iterations required would depend on the eigenvalue distribution of $\mathbf{P}^{-1}\widehat{\mathbf{K}}_{XX}$ and not $\widehat{\mathbf{K}}_{XX}$ while preserving same solution $\mathbf{x}^*$.

[2]Image generated from Preconditioned Gradient Descent.

Method

PCG used here is tailored for GP inference (Gardner et al. 2018):

- Specialized preconditioner
- Include stochastic trace estimate of $\text{Tr}\left[\widehat{\mathbf{K}}_{XX}^{-1}\frac{\partial\widehat{\mathbf{K}}_{XX}}{\partial\theta}\right]$
- Include estimate of $log|\widehat{\mathbf{K}}_{XX}|$
- Each iteration requires one matrix multiplication with $\widehat{\mathbf{K}}_{XX}$
- Matrix multiplication is treated as a blackbox.

Algorithm:

1. Specify matrix multiplication routine $\widehat{\mathbf{K}}_{XX}\mathbf{u}$
2. For each $i \in \{1, \cdots\}$ iteration , update
   2.1 Current solution $\mathbf{u}_i$
   2.2 Current error $\mathbf{r}_i$
   2.3 Search direction $\mathbf{p}_i$
   2.4 Preconditioned error term $\mathbf{z}_i$

Method

Reduce memory requirement of $\widehat{\mathbf{K}}_{XX}\mathbf{u}$ to $O(n)$

Normally (Naive): Compute $\widehat{\mathbf{K}}_{XX}$, then $\widehat{\mathbf{K}}_{XX}\mathbf{u}$; mem $O(n^2)$.
Key Idea: Partition the kernel matrix to perform all matrix-vector multiplications without forming kernel matrix explicitly.

1. Partition $\mathbf{X} = \left[\mathbf{X}^{(1)}; \cdots; \mathbf{X}^{(l)}; \cdots; \mathbf{X}^{(p)}\right]$ to p parts (row)
2. For each l part:
   2.1 Compute $\widehat{\mathbf{K}}_{X^{(l)}X}$ with $\mathbf{X}^{(l)}, \mathbf{X}$
   2.2 Compute $\widehat{\mathbf{K}}_{X^{(l)}X}\mathbf{u}$

Memory requirement: $O(n^2/p)$
Memory requirement when $p \to n$: $O(n)$

Distribute to w nodes (GPUs)

For each iteration i of PCG

- $\mathbf{u}_i$ needs to be broadcasted; $O(n)$ comms
- $\widehat{\mathbf{K}}_{X^{(l)}X}\mathbf{u}$ needs to be collected from every device

Method

Predictions

- Linear solve $\widehat{\mathbf{K}}_{XX}^{-1}\mathbf{y}$ can be cached, dependent on training data
- Can be computed efficiently on single GPU (< 1 second)
- $\widehat{\mathbf{K}}_{XX}^{-1}\mathbf{y}$ can be computed with tighter tolerance ϵ

Preconditioning

- (Gardner et al. 2018) used conditioner of $k = 20$ but $k = 100$ improved wall-clock timing for large dataset.
- Impact of increasing conditioner is limited as its computed only once before PCG.

PCG Convergence

- PCG is not an approximate method
- Prediction uses $\epsilon \leq 0.01$ for good predictive performance
- Hyperparameter training uses $\epsilon = 1$; which is surprising.

Related Methods

1. CG-related algorithms
   1.1 used when kernel matrix is structured
   1.2 General purpose GP approximation using inducing points, designed for CG
   1.3 CG used to train up to 50k points using approximations and ‘off-the-shelf’ preconditioners.
2. Popular approximate GP methods via inducing points:
   2.1 Sparse Gaussian process regression (SGPR) selects inducing points Z using a regularized objective.
   2.2 Stochastic variational Gaussian processes (SVGP) introduces a set of variational parameters that can be optimized using minibatch training.

Experimental Setup

Metric

- $RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\left(y_i^* - y_i\right)^2}$

Key details

- Compared against:
  - SGPR with $m = 512$
  - SVGP with $m = 1024$
- Data split - 4/9 train, 2/9 valid, 3/9 test
- Matern 3/2 kernel
- *Pretraining - Randomly subset 10K training points to fit an exact GP whose hyperparameters will be used as initalization*

Hardware / Software

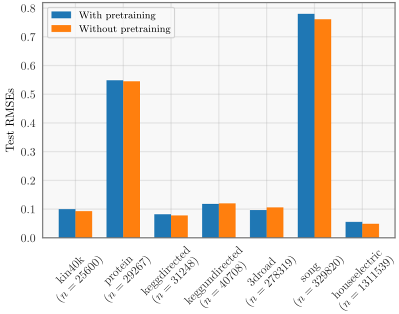
- GPyTorch backend
- 8 NVIDIA Tesla V100 - SGD 15K * 8 = $\boxed{\text{SGD 120K \$\$\$}}$

Experiments

The authors conducted the following experiments:

- **Effect of Pretraining** - effects of initalizing using pretrained hyperparameters
- **Accuracy** - compare accuracy against approximate methods
  - Single Lengthscale
  - Independent Lengthscale (Appendix)
- **Training Speedup** - investigates the scalability of the current method with more GPU compute
- **Ablation** - investigates subsampling of data vs increasing inducing points
- **Exact GPs with Adam (Appendix)** - Train exact GPs using Adam instead

Experiment - Effect of Pretraining



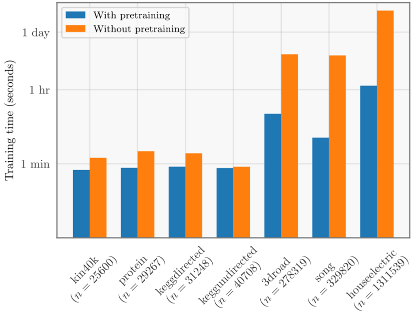
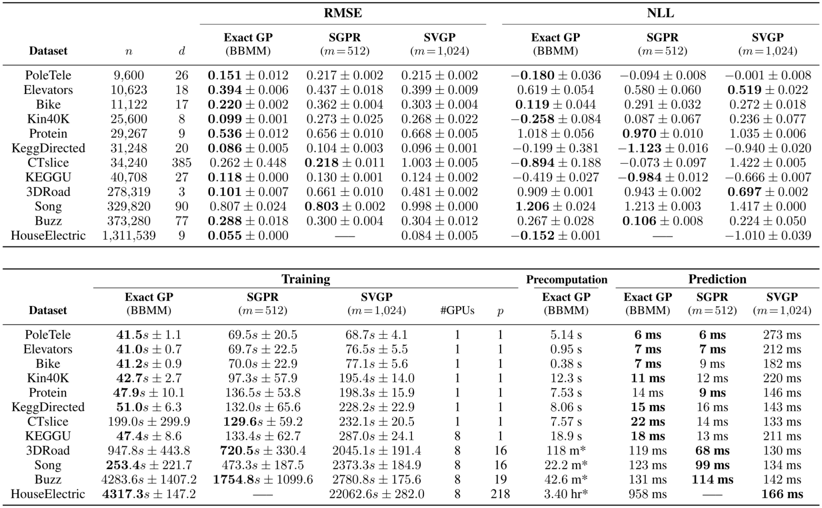


Figure 2: Exact GPs trained using initalization procedure

Note: No pretrain for SGPR and SVGP models as they required a significant number of fine-tuning steps after pretraining due to their increased number of model parameters.

Experiment - Accuracy (Single Lengthscale)



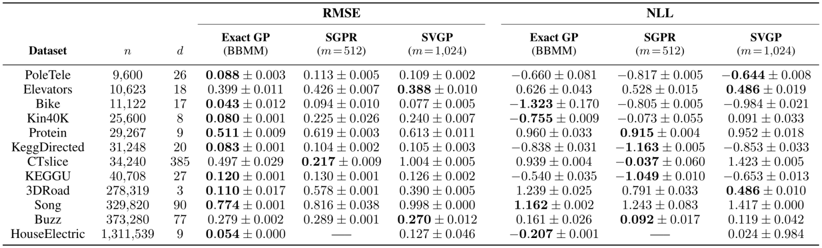
			RMSE			NLL		
Dataset	n	d	**Exact GP** (BBMM)	**SGPR** ($m=512$)	**SVGP** ($m=1{,}024$)	**Exact GP** (BBMM)	**SGPR** ($m=512$)	**SVGP** ($m=1{,}024$)
PoleTele	9,600	26	$\mathbf{0.151} \pm 0.012$	0.217 ± 0.002	0.215 ± 0.002	$\mathbf{-0.180} \pm 0.036$	-0.094 ± 0.008	-0.001 ± 0.008
Elevators	10,623	18	$\mathbf{0.394} \pm 0.006$	0.437 ± 0.018	0.399 ± 0.009	0.619 ± 0.054	0.580 ± 0.060	$\mathbf{0.519} \pm 0.022$
Bike	11,122	17	$\mathbf{0.220} \pm 0.002$	0.362 ± 0.004	0.303 ± 0.004	$\mathbf{0.119} \pm 0.044$	0.291 ± 0.032	0.272 ± 0.018
Kin40K	25,600	8	$\mathbf{0.099} \pm 0.001$	0.273 ± 0.025	0.268 ± 0.022	$\mathbf{-0.258} \pm 0.084$	0.087 ± 0.067	0.236 ± 0.077
Protein	29,267	9	$\mathbf{0.536} \pm 0.012$	0.656 ± 0.010	0.668 ± 0.005	1.018 ± 0.056	$\mathbf{0.970} \pm 0.010$	1.035 ± 0.006
KeggDirected	31,248	20	$\mathbf{0.086} \pm 0.005$	0.104 ± 0.003	0.096 ± 0.001	-0.199 ± 0.381	$\mathbf{-1.123} \pm 0.016$	-0.940 ± 0.020
CTslice	34,240	385	0.262 ± 0.448	$\mathbf{0.218} \pm 0.011$	1.003 ± 0.005	$\mathbf{-0.894} \pm 0.188$	-0.073 ± 0.097	1.422 ± 0.005
KEGGU	40,708	27	$\mathbf{0.118} \pm 0.000$	0.130 ± 0.001	0.124 ± 0.002	-0.419 ± 0.027	$\mathbf{-0.984} \pm 0.012$	-0.666 ± 0.007
3DRoad	278,319	3	$\mathbf{0.101} \pm 0.007$	0.661 ± 0.010	0.481 ± 0.002	0.909 ± 0.001	0.943 ± 0.002	$\mathbf{0.697} \pm 0.002$
Song	329,820	90	0.807 ± 0.024	$\mathbf{0.803} \pm 0.002$	0.998 ± 0.000	$\mathbf{1.206} \pm 0.024$	1.213 ± 0.003	1.417 ± 0.000
Buzz	373,280	77	$\mathbf{0.288} \pm 0.018$	0.300 ± 0.004	0.304 ± 0.012	0.267 ± 0.028	$\mathbf{0.106} \pm 0.008$	0.224 ± 0.050
HouseElectric	1,311,539	9	$\mathbf{0.055} \pm 0.000$	—	0.084 ± 0.005	$\mathbf{-0.152} \pm 0.001$	—	-1.010 ± 0.039

	Training					Precomputation	Prediction		
Dataset	**Exact GP** (BBMM)	**SGPR** ($m=512$)	**SVGP** ($m=1{,}024$)	#GPUs	p	**Exact GP** (BBMM)	**Exact GP** (BBMM)	**SGPR** ($m=512$)	**SVGP** ($m=1{,}024$)
PoleTele	$\mathbf{41.5}s \pm 1.1$	$69.5s \pm 20.5$	$68.7s \pm 4.1$	1	1	5.14 s	**6 ms**	**6 ms**	273 ms
Elevators	$\mathbf{41.0}s \pm 0.7$	$69.7s \pm 22.5$	$76.5s \pm 5.5$	1	1	0.95 s	**7 ms**	**7 ms**	212 ms
Bike	$\mathbf{41.2}s \pm 0.9$	$70.0s \pm 22.9$	$77.1s \pm 5.6$	1	1	0.38 s	**7 ms**	9 ms	182 ms
Kin40K	$\mathbf{42.7}s \pm 2.7$	$97.3s \pm 57.9$	$195.4s \pm 14.0$	1	1	12.3 s	**11 ms**	12 ms	220 ms
Protein	$\mathbf{47.9}s \pm 10.1$	$136.5s \pm 53.8$	$198.3s \pm 15.9$	1	1	7.53 s	14 ms	**9 ms**	146 ms
KeggDirected	$\mathbf{51.0}s \pm 6.3$	$132.0s \pm 65.6$	$228.2s \pm 22.9$	1	1	8.06 s	**15 ms**	16 ms	143 ms
CTslice	$199.0s \pm 299.9$	$\mathbf{129.6}s \pm 59.2$	$232.1s \pm 20.5$	1	1	7.57 s	**22 ms**	14 ms	133 ms
KEGGU	$\mathbf{47.4}s \pm 8.6$	$133.4s \pm 62.7$	$287.0s \pm 24.1$	8	1	18.9 s	**18 ms**	13 ms	211 ms
3DRoad	$947.8s \pm 443.8$	$\mathbf{720.5}s \pm 330.4$	$2045.1s \pm 191.4$	8	16	118 m*	119 ms	**68 ms**	130 ms
Song	$\mathbf{253.4}s \pm 221.7$	$473.3s \pm 187.5$	$2373.3s \pm 184.9$	8	16	22.2 m*	123 ms	**99 ms**	134 ms
Buzz	$4283.6s \pm 1407.2$	$\mathbf{1754.8}s \pm 1099.6$	$2780.8s \pm 175.6$	8	19	42.6 m*	131 ms	**114 ms**	142 ms
HouseElectric	$\mathbf{4317.3}s \pm 147.2$	—	$22062.6s \pm 282.0$	8	218	3.40 hr*	958 ms	—	**166 ms**

Figure 3: Single lengthscale across dims for kernel; averaged over 3 trials.

Note: High Variance with CTslice 0.262 ± 0.448, DV: $[0, 180]$

Experiment - Accuracy (Independent Lengthscale)



			RMSE			NLL		
Dataset	n	d	**Exact GP** (BBMM)	**SGPR** ($m=512$)	**SVGP** ($m=1{,}024$)	**Exact GP** (BBMM)	**SGPR** ($m=512$)	**SVGP** ($m=1{,}024$)
PoleTele	9,600	26	$\mathbf{0.088} \pm 0.003$	0.113 ± 0.005	0.109 ± 0.002	-0.660 ± 0.081	-0.817 ± 0.005	$\mathbf{-0.644} \pm 0.008$
Elevators	10,623	18	0.399 ± 0.011	0.426 ± 0.007	$\mathbf{0.388} \pm 0.010$	0.626 ± 0.043	0.528 ± 0.015	$\mathbf{0.486} \pm 0.019$
Bike	11,122	17	$\mathbf{0.043} \pm 0.012$	0.094 ± 0.010	0.077 ± 0.005	$\mathbf{-1.323} \pm 0.170$	-0.805 ± 0.005	-0.984 ± 0.021
Kin40K	25,600	8	$\mathbf{0.080} \pm 0.001$	0.225 ± 0.026	0.240 ± 0.007	$\mathbf{-0.755} \pm 0.009$	-0.073 ± 0.055	0.091 ± 0.033
Protein	29,267	9	$\mathbf{0.511} \pm 0.009$	0.619 ± 0.003	0.613 ± 0.011	0.960 ± 0.033	$\mathbf{0.915} \pm 0.004$	0.952 ± 0.018
KeggDirected	31,248	20	$\mathbf{0.083} \pm 0.001$	0.104 ± 0.002	0.105 ± 0.003	-0.838 ± 0.031	$\mathbf{-1.163} \pm 0.005$	-0.853 ± 0.033
CTslice	34,240	385	0.497 ± 0.029	$\mathbf{0.217} \pm 0.009$	1.004 ± 0.005	0.939 ± 0.004	$\mathbf{-0.037} \pm 0.060$	1.423 ± 0.005
KEGGU	40,708	27	$\mathbf{0.120} \pm 0.001$	0.130 ± 0.001	0.126 ± 0.002	-0.540 ± 0.035	$\mathbf{-1.049} \pm 0.010$	-0.653 ± 0.013
3DRoad	278,319	3	$\mathbf{0.110} \pm 0.017$	0.578 ± 0.001	0.390 ± 0.005	1.239 ± 0.025	0.791 ± 0.033	$\mathbf{0.486} \pm 0.010$
Song	329,820	90	$\mathbf{0.774} \pm 0.001$	0.816 ± 0.038	0.998 ± 0.000	$\mathbf{1.162} \pm 0.002$	1.243 ± 0.083	1.417 ± 0.000
Buzz	373,280	77	0.279 ± 0.002	0.289 ± 0.001	$\mathbf{0.270} \pm 0.012$	0.161 ± 0.026	$\mathbf{0.092} \pm 0.017$	0.119 ± 0.042
HouseElectric	1,311,539	9	$\mathbf{0.054} \pm 0.000$	—	0.127 ± 0.046	$\mathbf{-0.207} \pm 0.001$	—	0.024 ± 0.984

Figure 4: Independent lengthscales for kernel; averaged over 3 trials.

- Exact GPs achieve lower error
- Approximate methods are not affected by size or dimensionality
- Approximate methods perform similarly
- $n < 35000$ fits on 32GB GPU
- Fast predictions after precompute, predictions on RTX2080Ti

Experiment - Training Speedup

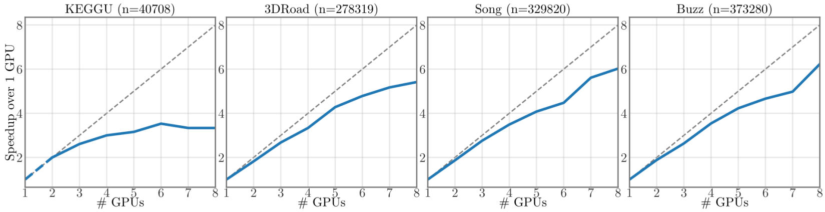


Figure 5: Speedup for datasets $d = \{27, 3, 90, 77\}$

- Sublinear speedup; almost linear for $1 \to 2$
- Larger datasets need more kernel partioning ($p = \{1, 16, 16, 19\}$)

Note: Having discussion on p will be nice; show HouseElectric will be better

Experiment - Ablation

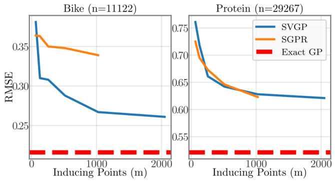


Figure 6: Approximate GP with more inducing points

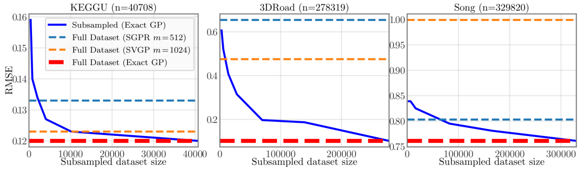
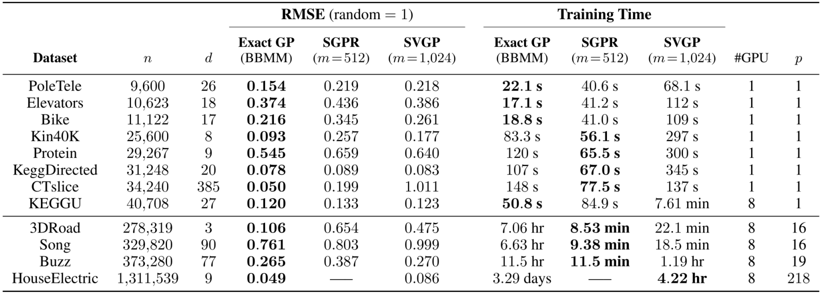


Figure 7: Exact GP on subsampled dataset

- Exact GP with less than 1/4 training data outperformed approximate GPs trained on the entire training set
- Train an exact GP on more data is preferable

Experiment - Exact GPs with Adam



			RMSE (random = 1)			Training Time				
Dataset	n	d	**Exact GP** (BBMM)	**SGPR** ($m=512$)	**SVGP** ($m=1{,}024$)	**Exact GP** (BBMM)	**SGPR** ($m=512$)	**SVGP** ($m=1{,}024$)	#GPU	p
PoleTele	9,600	26	**0.154**	0.219	0.218	**22.1 s**	40.6 s	68.1 s	1	1
Elevators	10,623	18	**0.374**	0.436	0.386	**17.1 s**	41.2 s	112 s	1	1
Bike	11,122	17	**0.216**	0.345	0.261	**18.8 s**	41.0 s	109 s	1	1
Kin40K	25,600	8	**0.093**	0.257	0.177	83.3 s	**56.1 s**	297 s	1	1
Protein	29,267	9	**0.545**	0.659	0.640	120 s	**65.5 s**	300 s	1	1
KeggDirected	31,248	20	**0.078**	0.089	0.083	107 s	**67.0 s**	345 s	1	1
CTslice	34,240	385	**0.050**	0.199	1.011	148 s	**77.5 s**	137 s	1	1
KEGGU	40,708	27	**0.120**	0.133	0.123	**50.8 s**	84.9 s	7.61 min	8	1
3DRoad	278,319	3	**0.106**	0.654	0.475	7.06 hr	**8.53 min**	22.1 min	8	16
Song	329,820	90	**0.761**	0.803	0.999	6.63 hr	**9.38 min**	18.5 min	8	16
Buzz	373,280	77	**0.265**	0.387	0.270	11.5 hr	**11.5 min**	1.19 hr	8	19
HouseElectric	1,311,539	9	**0.049**	—	0.086	3.29 days	—	**4.22 hr**	8	218

Figure 8: Exact GPs with Adam

- Fair comparison aginst SGPR and SVGPs that were trained with Adam.

Note: Missing error term, number of trials, results better than using log marginal likelihood with gradient desent.

Summary

Authors' summary

- Large datasets need scalable aproximations; Not anymore.
- Is CG an exact method?; CG can be more precise than Cholesky approaches (Gardner et al. 2018).
- Approximation can still be useful with limited resources.

Some of my thoughts

- CG is in theory, direct, it produces the exact solution after n iterations; the exact solution is never obtained in practice due to round off errors. CG is unstable with small perturbations.
- Effect of ϵ on hyperparameter - Is it really true that $\epsilon = 1$ has little impact on final model performance?
  - Plot the $\epsilon \in [0.001, 0.01, 0.1, 1]$ for hyperparameters
- Solving ϵ for prediction with even tighter tolerence and investigate its impact.

Summary

Some of my thoughts

- Log marginal likelihood is actually an estimation (Gardner et al. 2018); see (4) and (5)
  - (Artemev, Burt, and Wilk 2021) Bounds on LML for CG[3].
  - (Artemev, Burt, and Wilk 2021) 'In contrast, when using Iterative GP any bias introduced due to an insufficient number of iterations of CG may lead to either over or under estimation of the log marginal likelihood.'
- High variance CTslice
  - (Artemev, Burt, and Wilk 2021) 'However, Iterative GP training and predictive performance became unstable in later optimisation steps.'
- GPs with Adam optimizer?
  - GPs with Adam is not common
  - How about with others?

[3] https://arxiv.org/pdf/2102.08314.pdf

Bibliography

Artemev, Artem, David R Burt, and Mark van der Wilk. 2021. "Tighter Bounds on the Log Marginal Likelihood of Gaussian Process Regression Using Conjugate Gradients." *arXiv Preprint arXiv:2102.08314*.

Gardner, Jacob R, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson. 2018. "Gpytorch: Blackbox Matrix-Matrix Gaussian Process Inference with Gpu Acceleration." *arXiv Preprint arXiv:1809.11165*.

Nguyen, Duc-Trung, Maurizio Filippone, and Pietro Michiardi. 2019. "Exact Gaussian Process Regression with Distributed Computations." In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 1286–95. SAC '19. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3297280.3297409.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. 2011. "Scikit-Learn: Machine Learning in Python." *Journal of Machine Learning Research* 12: 2825–30.