Supplementary Materials for Low Precision Arithmetic for Fast Gaussian Processes

Wesley J. Maddox[1,*] **Andres Potapczynski**[1,*] **Andrew Gordon Wilson**[1]

[1]Center for Data Science, New York University
[*]Equal contribution.

Our Appendix is structured as follows:

- In Appendix A, we further describe related work, including on conjugate gradients.
- In Appendix B, we show several other experiments on both the properties of half precision kernel matrices and half precision conjugate gradients.
- In Appendix C, we outline experimental details for all of our experiments.
- In Appendix D, we give some detailed theoretical analysis of half precision kernel matrices, focusing on the quantized effective dimension and the effect of finite precision on the support of the kernel.

A EXTENDED RELATED WORK

Conjugate Gradients: A description of the conjugate gradients algorithm is given in Alg. 1 while using preconditioning [Nocedal and Wright, 2006, Golub and Loan, 2018]. Gardner et al. [2018] propose a variant of conjugate gradients that they call modified batched CG (mBCG) which we use in our work. The primary difference between mBCG and CG is that mBCG enables solving several linear systems at once by performing all computations in batch mode so that linear operators such as $\mathbf{K}(\mathbf{v})$ are actually matrix matrix multiplications rather than matrix vector products. Then, an individual set of learning rates α_k and β_k is used for each system. Our stable CG implementation (Alg. 1) is actually based off of mBCG, but for didactic purposes we display only the standard CG version.

Algorithm 1 CG

Input: MVM function $\mathbf{K}(\cdot)$, initial solution guess $\mathbf{x}_0$, linear system right hand side $\mathbf{b}$, tolerance ϵ, preconditioner function $\mathbf{P}(\cdot)$
Initialize: $k \leftarrow 0$, $\mathbf{r}_0 \leftarrow \mathbf{K}(x_0) - \mathbf{b}$, $\mathbf{d}_0 \leftarrow -\mathbf{r}_0$, $\mathbf{z}_0 = \mathbf{P}(\mathbf{r}_0)$ and $\gamma_0 = \mathbf{r}_0^T \mathbf{z}_0$.
while $\|\mathbf{r}_k\|_2 < \epsilon$ **do**
  $\alpha_k = \frac{\gamma_k}{\mathbf{d}_k^T \mathbf{K}(\mathbf{d}_k)}$
  $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$
  $\mathbf{r}_{k+1} = \mathbf{r}_k + \alpha_k \mathbf{K}(\mathbf{d}_k)$
  $\mathbf{z}_{k+1} = \mathbf{P}(\mathbf{r}_{k+1})$
  $\gamma_{k+1} = \mathbf{r}_{k+1}^T \mathbf{z}_{k+1}$
  $\beta_{k+1} = \frac{\gamma_{k+1}}{\gamma_k}$
  $\mathbf{d}_{k+1} = -\mathbf{r}_{k+1} + \beta_{k+1} \mathbf{d}_k$
end while

Other Scalable Gaussian Processes We note that our matrix-free schemes can be used to scale up approximate kernel methods such as Nystrom style approximations [Smola and Schölkopf, 2000, Williams and Seeger, 2000]; indeed, Meanti et al. [2020] use Nystrom approximations and KeOps for their kernel ridge regression approach. However, Zhang et al.

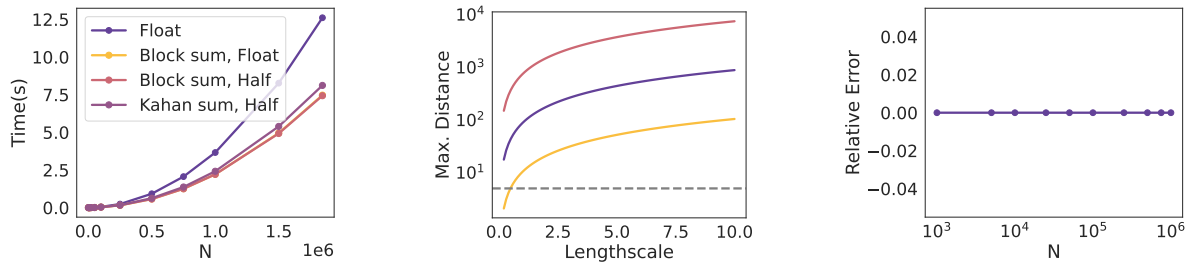
Accepted for the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2022).

[2019] found limited speedups when quantizing (which is slightly distinct from half precision) Nystrom approximations. Similarly, Pleiss et al. [2020] used iterative methods (in their cast MINRES) to speed up variational Gaussian processes [Titsias, 2009, Hensman et al., 2013], and we hope to speed up their approach as well. Chen et al. [2013], Nguyen et al. [2019] proposed parallel direct Cholesky based GP schemes for more scalable GP regression; however, their approaches will probably perform poorly in lower precision, as we demonstrate is the case for pivoted Cholesky based solves in Section 4.3.

Finally, kernels with compact support have been previously studied from a kernel approximation point of view [Genton, 2001, Gneiting, 2002]. However, these works focused on developing new techniques to approximate an infinitely supported kernel with a kernel that has demonstrated compact support, rather than using floating point precision to develop an approximate kernel with compact support.

B EXTENDED EXPERIMENTS

Summation Approaches In Figure A.1a, we display the different times of block summation across precisions, as well as Kahan summation and floating point accumultion of float kernel matrix MVMs, finding that all half precision accumulation mechanisms behave similarly, with Kahan summation being slightly slower than the other two. This plot is inspired by the study performed by the authors of KeOps [Charlier et al., 2021], available at `https://www.kernel-operations.io/keops/_auto_benchmarks/plot_accuracy.html`. Due to these results, we use block summation, casting each block's summation up to float before down-casting to half, as is the default in KeOps.

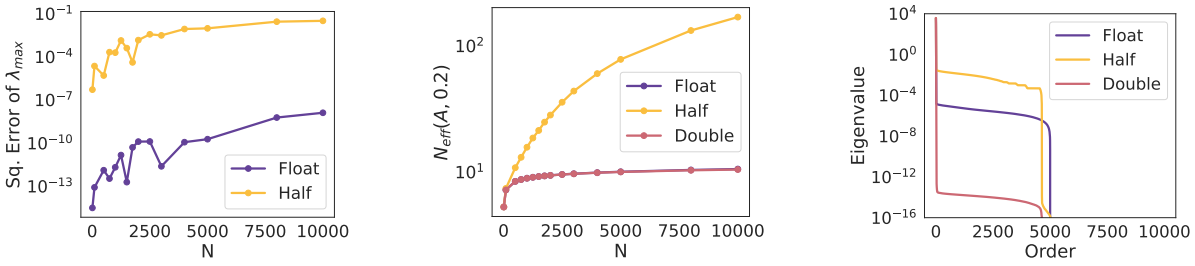


(a) Summations. (b) Matérn-$1/2$. (c) Error of sparsified MVMs.

Figure A.1: **(a)** Block summation in floating point adds negligible overhead compared with block summation in half precision, while being as accurate as Kahan summation. **(b)** Maximum distance representable for Matérn-$1/2$ kernel; note the similar trend to Figure 2a. **(c)** Error of the truncated MVM is zero as expected. To produce the sparsified MVM, we truncated any data points that had kernel entries that were un-representable in half precision.

Properties of Half Precision Kernel Matrices In Figure A.1b, we display the maximum distance representable for Matérn-$1/2$ kernels across varying lengthscales, as we show for RBF and rational quadratic kernels in Figures 2a and 2b. The trend for the Matérn family is similar to that of the RBF kernels, except that larger distances are representable.

Finally, in Figure A.1c, we show the error of sparsified MVMs (which is zero) across increasing dataset size for the data



(a) Difference of λ_{max}. (b) ED, RBF. (c) Spectrum, RBF.

Figure A.2: **(a)** Difference of $\lambda_{\max}$ across precisions for Matérn-$1/2$ kernel. Other kernels have similar eigenvalue diffeences. **(b)** Effective dimension (ED) for RBF kernels, the trend is similar to that of the Matérn-$1/2$ kernel because the eigenvalue spectrum **(c)** has a similar bunched up pattern in half.

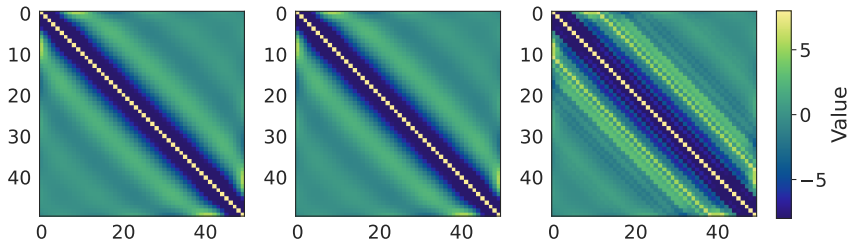


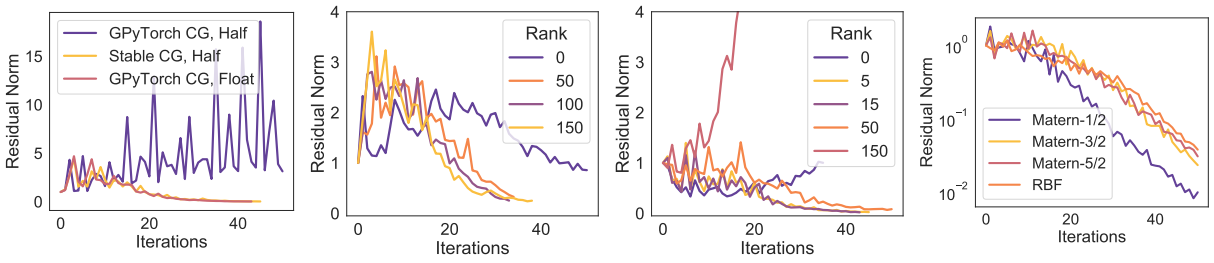
Figure A.3: Matrix inverses of an RBF kernel in double (left), float (middle), and half (right) precisions. The inverse is performed in double precision, while the evaluation itself is performed in half precision. The half precision inverse is qualitatively distinct from the other two indicating a slightly distinct spectrum.

reduction experiment in Section 4.1.

The difference of the largest eigenvalue of a Matérn-$1/2$ kernel is shown in Figure A.2a in float and half as compared to double precision (which we use as a proxy for infinite precision). Note that extremely small relative differences for these largest values.

In Figure A.2**(b)**, we show ED for RBF kernels with the associated spectrum in **(c)**.

In Figure A.3, we display $(\mathbf{K} + 0.01)^{-1}$ for RBF kernels with lengthscale 1 and 50 data points in $[-3, 3]$ across double (left), float (middle), and half (right) precisions. We first evaluate the kernel to a lower precision and then pass into double precision before using a Cholesky factorization to invert the kernel matrix, finding that the half precision kernel inverse has a distinct pattern (larger magnitude off-diagonal values) compared to the float and double inverse matrices.



(a) Buzz (b) Elevators (c) KeggD (d) PoleTele

Figure A.4: **(a)** Residual norms across solvers on buzz. **(b)** Residual norm for stable CG on elevators. Here no preconditioning also converges. **(c)** Residual norm for stable CG on Kegg Directed. **(d)** Residual norm for no ARD on Pol.

Benchmarking Half Precision CG In Figure A.4a, we display how CG in half diverges, but our stable CG converges as does CG in float. In Figure A.4b and A.4c, we display the effect of preconditioning on solves, finding again that larger preconditioners tend to converge very slightly faster.

In Figure A.5, we display the optimization trajectory on *3droad* finding that there are clearer divergences in terms of the outputscale; however, each parameter converges to similar values by the end of training.

Additional Benchmark Results In Table 2, we display NLLs across five seeds on UCI datasets for float, half, and SVGPs, analogous to our RMSE and timing results in Table 1. Compilation times for these results are shown in Table 1.

In Table 3, we display RMSEs, times, and NLLs for Matérn-$5/2$ ARD kernels for both single and half precisions.. In Table 5, we display RMSEs, times, and NLLs for Matérn-$1/2$ ARD kernels for both single and half precisions. In Table 4, we display RMSEs, times, and NLLs for Matérn-$1/2$ ARD kernels for both single and half precisions.

C EXPERIMENTAL DETAILS

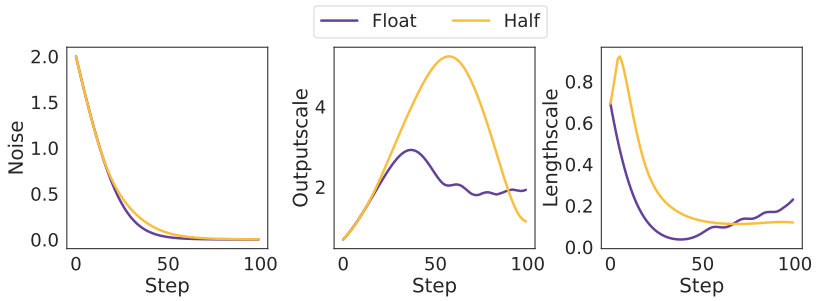


Figure A.5: Optimization trajectory on *3droad*.

Table A.1: Times with compilation breakdown across 3 seeds on a suite of UCI tasks.

		Half		Single	
Dataset	(N, d)	Optimization Time	Compilation Time	Optimization Time	Compilation Time
PoleTele	(13.5K, 26)	5 ± 0.024	73 ± 0.035	23.6 ± 0.5	34 ± 0.066
Protein	(41.1K, 9)	11 ± 2.795	73 ± 0.018	35.9 ± 5.7	35 ± 2.846
3droad	(391.4K, 3)	912.2 ± 0.96	91 ± 0.197	$1,210 \pm 35.0$	50 ± 0.220

Table A.2: Test time NLLs across 5 seeds on a suite of UCI tasks for float, half, and SVGPs with RBF ARD kernels.

Dataset	*(N, d)*	**Single**	**Half**	**SVGP**
PoleTele	(13.5K, 26)	-0.349 ± 0.004	-0.316 ± 0.004	-0.513 ± 0.011
Elevators	(14.9K, 18)	0.515 ± 0.0195	0.663 ± 0.024	0.437 ± 0.012
Bike	(15.6K, 17)	-0.3714 ± 0.0066	-0.413 ± 0.008	-1.020 ± 0.044
Kin40K	(36K, 8)	0.2352 ± 0.005	0.241 ± 0.005	-0.327 ± 0.007
Protein	(41.1K, 9)	0.9802 ± 0.0115	1.412 ± 0.001	0.964 ± 0.015
3droad	(391.4K, 3)	1.249 ± 0.0129	1.201 ± 0.005	0.537 ± 0.025
Song	(463.8K, 90)	1.146 ± 0.0043	1.765 ± 0.819	1.418 ± 0.002
Buzz	(524.9K, 77)	-0.424 ± 0.18	0.898 ± 0.714	-0.071 ± 0.010
Houseelectric	(1844.3K,9)	-0.72 ± 0.002	-0.439 ± 0.084	–

Table A.3: RMSEs, NLL, and training time across 3 seeds on a suite of UCI tasks. Here, we use Matérn-$5/2$ ARD kernels with 50 CG iterations and 50 optimization steps.

		RMSE		**NLL**		**Time**	
Dataset	(N, d)	**Single**	**Half**	**Single**	**Half**	**Single**	**Half**
PoleTele	(13.5K, 26)	0.098 ± 0.002	0.102	-0.454 ± 0.004	-0.447	49.1 ± 1.74	97
Protein	(41.1K, 9)	0.498 ± 0.007	0.509 ± 0.005	1.07 ± 0.01	1.04 ± 0.03	69.17 ± 1.7	110 ± 10
3droad	(391.4K, 3)	0.254 ± 0.006	0.231	0.812 ± 0.014	0.899	1666 ± 214	1501

Table A.4: RMSEs, NLL, and training time across 3 seeds on a suite of UCI tasks. Here, we use Matérn kernel $1/2$ with 50 CG iterations and 50 optimization steps.

		RMSE	**NLL**	**Time**
Dataset	(N, d)	**Half**	**Half**	**Half**
PoleTele	(13.5K, 26)	0.108 ± 0.003	0.0478 ± 0.006	85 ± 2.4
Elevators	(14.9K, 18)	0.365 ± 0.002	-0.432 ± 0.010	83 ± 1.9
Bike	(15.6K, 17)	0.096 ± 0.003	0.527 ± 0.004	84 ± 0.4
Kin40K	(36K, 8)	0.136 ± 0.002	0.194 ± 0.005	87 ± 0.1
Protein	(41.1K, 9)	0.481 ± 0.003	-0.762 ± 0.013	98 ± 0.1
3droad	(391.4K, 3)	0.083 ± 0.001	-0.045 ± 0.020	1285 ± 2.3

C.1 MAXIMUM DISTANCE REPRESENTABLE IN FINITE PRECISION

To compute these distances, we consider four separate stationary kernels [Rasmussen and Williams, 2008] with distance $d = |x - x'|$ and lengthscale l, focusing on determining what values they drop below a given ϵ. For Matérn-$1/2$ kernels, we need $\exp\{-d/l\} < \epsilon$ and solving gives $d > -\log \epsilon * l$. For other Matérn kernels (e.g. $3/2$ and $5/2$), there is no

Table A.5: RMSEs, NLL, and training time across 3 seeds on a suite of UCI tasks. Here, we use Matérn kernel $3/2$ with 50 CG iterations and 50 optimization steps.

		RMSE	NLL	Time
Dataset	(N, d)	**Half**	**Half**	**Half**
PoleTele	(13.5K, 26)	0.101 ± 0.003	0.475 ± 0.005	90 ± 2.2
Elevators	(14.9K, 18)	0.498 ± 0.112	-0.990 ± 0.423	86 ± 3.5
Bike	(15.6K, 17)	0.086 ± 0.003	0.567 ± 0.006	88 ± 0.1
Kin40K	(36K, 8)	0.097 ± 0.001	0.186 ± 0.004	96 ± 0.1
Protein	(41.1K, 9)	0.497 ± 0.003	-0.850 ± 0.008	107 ± 0.1
3droad	(391.4K, 3)	0.165 ± 0.003	-0.700 ± 0.024	1532 ± 3.4

Table A.6: RMSEs, NLL, and training time across 3 seeds on a suite of UCI tasks. Here, we use Matérn-$5/2$ kernels with 50 CG iterations and 50 optimization steps.

		RMSE	NLL	Time
Dataset	(N, d)	**Half**	**Half**	**Half**
PoleTele	(13.5K, 26)	0.102 ± 0.002	0.437 ± 0.011	96 ± 2.1
Elevators	(14.9K, 18)	0.520 ± 0.209	-0.967 ± 0.571	94 ± 5.9
Bike	(15.6K, 17)	0.082 ± 0.001	0.568 ± 0.008	93 ± 0.1
Kin40K	(36K, 8)	0.088 ± 0.001	0.078 ± 0.002	106 ± 0.1
Protein	(41.1K, 9)	0.512 ± 0.008	-1.048 ± 0.008	113 ± 11.3
3droad	(391.4K, 3)	0.226 ± 0.005	-0.905 ± 0.007	1486 ± 4.5

straightforward closed form solution, but empirical investigations showed that the maximum distance representable is somewhere between Matérn-$1/2$ and RBF.

For RBF Kernels, we have $\exp\{-1/2d^2/l^2\} < \epsilon$ and solving gets $d > (-2\log\epsilon)^{1/2}l$. For rational quadratic kernels, we have $(1 + \frac{d^2}{2\alpha l^2})^{-\alpha} < \epsilon$ and solving for d gets $d > (2\alpha(\epsilon^{1/\alpha} - 1))^{1/2} * l$. We found that for $\alpha = 2, 3$ the size of the support was much larger, and so showed only $\alpha = 5$. For periodic kernels, we have $-\frac{2}{\lambda}\sin^2\left(\frac{\pi}{p}|d|\right) < \log\epsilon$ and solving gets $|d| > \frac{p}{\pi}\arcsin\left(-\log\epsilon\frac{\lambda}{2}\right)$ which will only have solutions when $-\log\epsilon\frac{\lambda}{2} \leq 1$.

C.2 EXPERIMENTAL SETUP

All timing based experiments were performed using single NVIDIA V100 GPUs with 32GB of memory on a shared supercomputing cluster. Non-timing experiments also used NVIDIA RTX GPUs with either 24 or 48 GB of memory on either the same cluster or on a private internal server.

We used GPyTorch [Gardner et al., 2018] with the default parameter settings from Botorch's single task GP model[1] which are constraining the noise to be greater than 0.0001 and a Gamma$(1.1, 0.05)$ prior on the noise with initialization to 2, and Gamma$(2.0, 0.15)$ prior on the outpuscale and a Gamma$(3.0, 6.0)$ prior on the lengthscale(s). We fit using Adam for either 50 or 100 iterations unless otherwise documented and used a tolerance of 1.0 for the CG iterations unless otherwise stated. We used KeOps 1.5 for our experiments, noting that preliminary experiments with KeOps 2.0 produced significantly faster compilation times [Charlier et al., 2021].

For the datasets, we used the bayesian benchmarks package of https://github.com/hughsalimbeni/bayesian_benchmarks/, following their default training and testing splits.

At test time, we converted the models back to float precision; however, our experiments found that this actually had limited impact on the RMSEs.

D THEORETICAL ANALYSIS

D.1 EFFECT OF FINITE PRECISION ON SUPPORT

Following Rasmussen and Williams [Chapter 4.3 of 2008] we can express the eigenvalues of an RBF kernel as $\lambda_k = \sqrt{\frac{2a}{A}}B^k$ for some positive constants a, A and $B \in (0, 1)$ that depend on the hyperparameters of the RBF kernel. In infinite-precision

[1] https://botorch.org/api/models.html#module-botorch.models.gp_regression.

an RBF kernel has support over the whole space as $\lambda_k > 0$. However, if

$$k \geq \frac{\log \delta + \frac{1}{2}\log\left(\frac{A}{2a}\right)}{\log B} = \mathcal{O}(\log \delta)$$

then $\lambda_k = 0$ in finite-precision, where δ represents the round-off error. This means that the support of the kernel gets cut-off. This is similar to the support a Gaussian distribution $\mathcal{N}\left(\mu, \sigma^2\right)$ being the whole line $\mathbb{R}$, however, computing the probability of a sample being several standard deviations from the mean get cut-off to zero due to the sharp decay of the tails. Thus, our results focus on the empirical support of the kernel, not on the theoretical one.

D.2 EFFECT OF FINITE-PRECISION ON GENERALIZATION

Following Li and De Sa [2019], we assume that moving from infinite precision to finite precision and using stochastic rounding means that our finite precision number, $Q(a) \sim U(a-\delta, a+\delta)$ for some δ that depends on our quantization (e.g. our precision based scheme). Eqs. 11-15 of Opper and Vivarelli [1998] (see also Thm 1 of Dicker et al. [2017], Thm 4.1 of Zhang [2005] and Thm 4 of Caponnetto and Vito [2007]), generalization bounds often depend on the effective dimension of the training kernel matrix. Recall that the effective dimension is computed from the eignenvalues as $\sum_{i=1}^{N} \frac{\lambda_i}{\lambda_i + s}$ for some value s. We will compute our finite precision approximation by computing the expected value of the effective dimension under the stochastic rounding scheme.

Furthermore, we assume that each eigenvalue is quantized independently, to compute the expected effective dimension, we need to compute N integrals of the following form, where $p(x) = U(a-\delta, a+\delta)$:

$$\begin{aligned}
\mathbb{E}_{p(x)} \frac{x}{x+s} &= \int_{a-\delta}^{a+\delta} \frac{x}{x+s} \frac{1}{a+\delta-(a-\delta)} dx = \frac{1}{2\delta}(x - s\log(x+s))|_{a-\delta}^{a+\delta} \\
&= \frac{1}{2\delta}\left(a+\delta - s\log(a+\delta+s) - (a-\delta-s\log(a-\delta+s))\right) \\
&= 1 + \frac{s}{2\delta}\log\frac{a+s-\delta}{a+s+\delta} = 1 + \frac{s}{2\delta}\log\left(1 - \frac{2\delta}{a+s+\delta}\right) \\
&= 1 - \frac{s}{2\delta}\left(\frac{2\delta}{a+s+\delta} + \frac{4\delta^2}{2(a+s+\delta)^2} + \frac{8\delta^3}{3(a+s+\delta)^3} + \mathcal{O}(\delta^4)\right) \\
&= 1 - \frac{s}{a+s+\delta} - \frac{\delta s}{(a+s+\delta)^2} - \frac{4\delta^2 s}{3(a+s+\delta)^3} - \mathcal{O}(\delta^3) \qquad (1) \\
&\geq 1 - \frac{s}{a+s+\delta} - \frac{\delta s}{(a+s+\delta)^2} \qquad (2)
\end{aligned}$$

Now, putting this into our expectation over the quantized eigenvalues:

$$\begin{aligned}
\mathbb{E}\left(\sum_{i=1}^{N} \frac{Q(\lambda_i)}{Q(\lambda_i)+s}\right) &\geq \sum_{i=1}^{N} 1 - \frac{s}{\lambda_i+s+\delta} - \frac{\delta s}{(\lambda_i+s+\delta)^2} \qquad (3) \\
&= N - \sum_{i=1}^{N} \frac{s(\lambda_i+s+\delta)-\delta s}{(\lambda_i+s+\delta)^2} = N - \sum_{i=1}^{N} \frac{s(\lambda_i+s)}{(\lambda_i+s+\delta)^2} \\
&\geq N - \sum_{i=1}^{N} \frac{s}{\lambda_i+s} \\
&= N_{\text{eff}}(K, s)
\end{aligned}$$

Note that as $\delta \to 0$ all of these inequalities become tight as expected. What this shows is that in finite precision, the expected effective dimensionality can only be higher than the effective dimensionality in infinite precision.

In general, bounds such as Eqs. 11, 16 of Opper and Vivarelli [1998] (also Eq. 7.26 in Rasmussen and Williams [2008] tend to depend on the expected eigenvalues rather than those estimated empirically (e.g. in finite precision). However, they are related as $\frac{1}{N}\lambda_i^{\text{emp}} \to \lambda_i$ (see Rasmussen & Williams, 06 4.3.2) and so we estimate $N_{\text{eff}}(K, \sigma^2/n)$ with $N_{\text{eff}}(K^{\text{emp}}, \sigma^2)$. Plugging in our finite precision estimate to something like Thm. 4.1 of Zhang [2005] then suggests that the generalization error in (any)

finite precision will tend to be higher than for infinite precision. Roughly, these bounds state that the generalization error of a kernel ridge regressor is upper bounded by the sum of approximation error terms (relating to the fit of the kernel to the function) plus $N_{\text{eff}}(K, \lambda)/n$ for a regularization term λ (similar to the noise value in the GP setting).

Finally, our analysis shows that a larger δ (e.g. a lower precision estimate) will tend to further increase the generalization error. This tends to confirm our experimental study on the effective dimension in Figure 2d.

References

A. Caponnetto and E. D. Vito. Optimal rates for the regularized least-squares algorithm. *FOCM*, 2007.

B. Charlier, J. Feydy, J. Glaunès, F.-D. Collin, and G. Durif. Kernel operations on the gpu, with autodiff, without memory overflows. *JMLR*, 2021.

J. Chen, N. Cao, K. H. Low, R. Ouyang, C. K.-Y. Tan, and P. Jaillet. Parallel gaussian process regression with low-rank covariance matrix approximations. *arXiv:1305.5826*, 2013.

L. H. Dicker, D. P. Foster, and D. Hsu. Kernel ridge vs. principal component regression: Minimax bounds and the qualification of regularization operators. *Electronic Journal of Statistics*, 11(1):1022–1047, 2017.

J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson. GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration. *NeurIPS*, 2018.

M. G. Genton. Classes of kernels for machine learning: a statistics perspective. *JMLR*, 2001.

T. Gneiting. Compactly supported correlation functions. *Journal of Multivariate Analysis*, 2002.

G. H. Golub and C. F. V. Loan. *Matrix Computations*. The Johns Hopkins University Press, 2018. Fourth Edition.

J. Hensman, N. Fusi, and N. D. Lawrence. Gaussian Processes for Big Data. *UAI*, 2013.

Z. Li and C. M. De Sa. Dimension-free bounds for low-precision training. *Advances in Neural Information Processing Systems*, 32, 2019.

G. Meanti, L. Carratino, L. Rosasco, and A. Rudi. Kernel methods through the roof: handling billions of points efficiently. *NeurIPS*, 2020.

D.-T. Nguyen, M. Filippone, and P. Michiardi. Exact gaussian process regression with distributed computations. In *ACM/SIGAPP Symposium on Applied Computing*, 2019.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 2006. Second Edition.

M. Opper and F. Vivarelli. General bounds on bayes errors for regression with gaussian processes. *Advances in Neural Information Processing Systems*, 11, 1998.

G. Pleiss, M. Jankowiak, D. Eriksson, A. Damle, and J. R. Gardner. Fast Matrix Square Roots with Applications to Gaussian Processes and Bayesian Optimization. *NeurIPS*, 2020.

C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2008.

A. J. Smola and B. Schölkopf. Sparse Greedy Matrix Approximation for Machine Learning. *ICML*, 2000.

M. K. Titsias. Variational Learning of Inducing Variables in Sparse Gaussian Processes. *AISTATS*, 2009.

C. Williams and M. Seeger. Using the Nyström Method to Speed Up Kernel Machines. *NeurIPS*, 2000.

J. Zhang, A. May, T. Dao, and C. Re. Low-precision random fourier features for memory-constrained kernel approximation. In *AISTATS*, 2019.

T. Zhang. Learning bounds for kernel regression using effective data dimensionality. *Neural Computation*, 2005.