Low Precision Arithmetic for Approximate Gaussian Process Inference

Giulio Duregon - gjd9961 GJD9961@NYU.EDU
Jonah Poczobutt - jp6422 JP6422@NYU.EDU
Paul Koettering - pmk2054 PMK2054@NYU.EDU

Abstract

Using lower precision arthmetic is a popular technique in large-scale deep learning to improve computational requirements such as training time and memory resources. This technique has recently been adopted for Gaussian Processes Inference (Maddox et al., 2022) and it has been shown that mixed precision algorithms can provably accelerate computation. We propose investigating the effect of using low-precision arithmetic on approximate GP inference methods. We develop a new greedy method for performing low-precision approximate Gaussian Process inference and analyse the performance as well as failure modes of the method. We compare our method with existing approaches and find that when using low precision, we gain significant speedup in training time without the introduction of additional test error.

1. Introduction

The scale of data sets being used in modern machine learning is extremely large, often with hundreds of thousands or millions of data points being used by a single model. Whilst techniques such as neural networks are able to deal with large datasets through mini-batching, Gaussian processes must rely upon approximation techniques in order to perform inference and prediction.

Gaussian process inference is difficult because it requires solving a very large system of linear equations, either by inverting some data-matrix or using matrix decomposition techniques (Rasmussen & Williams, 2009). These methods typically scale by $\mathcal{O}(n^3)$ for their time-complexity and requires $\mathcal{O}(n^2)$ memory. When dealing with large data sets, the computational costs of inverting a data-matrix becomes intractable, thereby necessitating the need for approximation methods. Two methods which have been used previously to solve this problem are low-precision arithmetic and sparse Gaussian process methods (Candela & Rasmussen, 2005).

The low-precision arithmetic approach has been adopted from successful application in neural networks (Gupta et al., 2015), where novel summation techniques and alternative floating point representations have become commonplace (Sun et al., 2020). By using low-precision computations the memory and time requirements are reduced, however the linear algebra operations that are required to perform exact GP inference (such as Cholesky decomposition) are unstable in low-precision. A standard approach to reduce the computational requirements of Gaussian Processes is to select a smaller subset of inducing function inputs which create an approximation of the true Gaussian process model (Candela & Rasmussen, 2005) (Snelson & Ghahramani, 2005) (Titsias, 2009). This reduces the time complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(mn^2)$, where m is the size of the smaller subset of inducing points. One particular approach is to directly approximate the GP mean and covariance function using a variational inference technique to select a subset of points such that the KL divergence between the exact posterior GP and a variational approximation is minimized (Titsias, 2009) (de G. Matthews et al., 2015).

Despite the success of state-of-the-art sparse Gausssian Process approximation methods, they can still be practically unsuitable for extremely large data sets and in this paper we will aim to aleviate this by combining sparse Gaussian Process inference with low-precision techniques. Put simply, if the variational inference technique is approximating the true posterior anyway, thereby sacrificing some statistical accuracy, then slightly disturbing this already approximated posterior by using low-precision arithmetic should not have disastrous consequences for the result.

The main contributions of this project are :

- We provide an analysis of how low precision computations effect the performance of approximate gaussian process inference.
- We propose a new method for low-precision Gaussian process approximate inference based on our analysis.

Final Project for Bayesian Machine Learning course at NYU.

- We provide a practical implementation for this method and demonstrate the performance.
- We provide our code `https://github.com/giulio-duregon/LowPrecisionApproxGP`

2. Preliminaries

We will here review different floating point precision's that are used in the project, the fundamental approximate Gaussian Process inference technique and some of the numerical considerations we are required to make when developing our technique.

2.1. Numerical Precision

The three choices of floating point precision which will be used in this project are double, single and half precision (Salvucci, 1985). When using double precision each number is represented using 64 bits, whereas in single and half precision this is reduced to 32 and 16 bits respectively. This reduction in the number of bits used for each number limits the range of values which we are able to represent. Using the double precision format allows for representation of smallest numbers in absolute value as small as approximately 2.225×10^{-308} whereas if we only use single precision floating numbers this reduces the range of small numbers we are able to represent to approximately 1.175×10^{-38}. This leads to a round-off error of approximately 6×10^{-8} which can indeed be very significant if the scale of the numbers which we are using in our computations are very small, which can often the case in Gaussian Process inference. Furthermore, when we reduce the number of bits we choose to represent each data even further to 16-bits in half precision, the smallest numbers in absolute terms we are able to represent is approximately 6.104×10^{-5} with a round of error of approximately 1×10^{-4} which is four orders of magnitude greater than in single precision and can cause significant difficulty in numerical stability.

2.2. Approximate Gaussian Process Inference

A Gaussian Process is a set of random variables whereby each finite subset of the points follows a Gaussian distribution. In order to fully specify a Gaussian Process we must define the covariance kernel and mean function. We consider a regression task on observed data $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$ for $\boldsymbol{x}_i \in \mathbf{R}^P$ and $y_i \in \mathbf{R}$ where y_i is a noisy realization of the function we are interested in learning:

$$\begin{aligned} f(\cdot) &\sim \mathcal{GP}\left(m(\cdot), k(\cdot,\cdot)\right) \\ y_i &= f(\boldsymbol{x}_i) + \epsilon_i \\ \epsilon_i &\sim \mathcal{N}(0, \sigma^2) \end{aligned}$$

The posterior of this model, which will be used for making predictions has a closed-form expression (Rasmussen & Williams, 2009). The posterior depends upon the values of the Kernel hyper parameters and the noise level $(\boldsymbol{\theta}, \sigma^2)$ which we can estimated through marginal log likelihood maximisation. The marginal log likelihood is given in closed-form as:

$$\log p(y) = \log\left[\mathcal{N}(\boldsymbol{y}|0, \sigma^2 I + K_{nn})\right]$$

This calculation requires the large $\mathcal{O}(n^3)$ matrix inversion which we aim to avoid by using a set of inducing points. We must learn the best inducing points as well as the kernel hyper-parameters for a sparse GP method, the variational inference approach used in (Titsias, 2009) achieves this by minimizing the distance between an augmented true posterior $p(f, f_m|y)$ and an augmented variational posterior $q(f, f_m)$. Here f_m is a small set of auxiliary inducing variables evaluated at pseudo-inputs X_m. The augmented representation of the joint model has a set of parameters X_m which we vary to minimize the KL divergence $\mathcal{KL}(q(f, f_m), p(f, f_m|y))$. Minimization of this KL divergence is equivalent to maximisation of the evidence lower bound (ELBO). (Titsias, 2009) show that maximization of this bound is equivalent to maximisation of:

$$F_V(X_m) = \log[\mathcal{N}(\boldsymbol{y}|0, \sigma^2 I + Q_{nn})] - \frac{1}{2\sigma^2}\text{Tr}(\hat{K})$$

Where

$$Q_{nn} = K_{nm}K_{mm}^{-1} \quad \text{and} \quad \hat{K} = K_{nn} - K_{nm}K_{mm}^{-1}K_{mn}$$

which are both composed of variance covariance matrices. This term is the lower of bound of the true marginal log likelihood and the two terms represent the sum of the projected process approximation (Seeger et al., 2003) and an added regularization term. The trace term corresponds to the squared error of predicting the training latent values from the inducing variables and when it equals zero the inducing variables are able to perfectly reproduce full Gaussian Process prediction.

2.3. Numerical Properties

When investigating the effect of low-precision techniques on Gaussian Process regression we investigate the numerical properties of the kernel matrix.

The support of a kernel matrix provides information about the lengthscale across which the relationship between points is able to be encoded into the function approximation. If we are unable to represent very small numbers in our precision type then some of the smallest values in the kernel matrix, those that lie far from the diagonal, will be

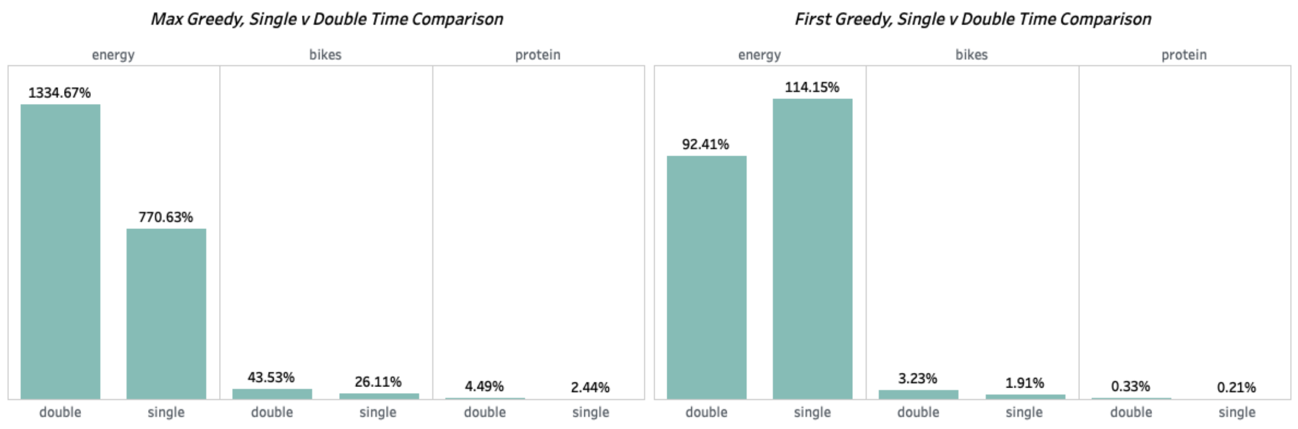

Figure 1. Train time comparison between precision types and greedy selection strategies as a percentage of train time for the baseline model. For *Max Greedy*, 20 total samples are averaged per dataset, 5 for each $|J| \in \{10, 20, 40, 60\}$.

zeroed out. This means that for lower precision the relationships between points with larger lengthscale may not be included. Furthermore, this effect may be compounded when using sparse Gaussian Process approximations since fewer points are included in the kernel matrix and therefore zeroing out of any of the relationships between points could lead to significant loss of expressivity. This relationship is something that we explore in this project.

The eigenspectrum of a kernel matrix also plays a significant role in the Gaussian Process regression procedure and (Maddox et al., 2022) showed that the difference in eigenspectrum of lower precision approximations can lead to lower generalization. They showed that the lower precision techniques tended to have slower eigenvalue decay. The effective dimension of a kernel matrix $N_{\text{eff}}(K, \sigma^2)$ provides upper generalization bounds and so when the decay is slower, the effective dimension is greater and the upper bound is weaker. This matches what is also observed in kernel regression and Gaussian Process methods (Zhang, 2005).

3. Related Work

3.1. Low Precision Arithmetic

Whilst the use of low-precision arithmetic is very prevalent in other machine learning domains and has been known to provide performance gains for a long time (Hammerstrom, 1990) (Chen et al., 2014), its use in Gaussian Processes is highly limited. (Maddox et al., 2022) proposed a novel numerically stable conjugate gradient algorithm that showed fast convergence in a variety of Gaussian Process regression tasks. The conjugate gradient method is used to solve the linear system in their method since it suffer less from round-off errors compared to other solvers and can be easily parallelized (Gardner et al., 2018b). Furthermore, this approach uses a collection of problem-specific strategies to improve the stability and convergence of the method such as preconditioning, re-orthogonalization, mixed precision and blocked arithmetic. Due to the zeroing out of small off-diagonal elements in lower precision, the support of a kernel matrix becomes increasingly truncated, resulting in more efficient matrix vector multiplication. In this project we also explore the effect on small off-diagonal elements in the sparse covariance matrix approximation.

3.2. Sparse Gaussian Process Regression

Much work has been done on trying to avoid the computational and memory requirements of large-scale Gaussian Processes. For example, (Hensman et al., 2013) proposed optimising the covariance and mean functions numerically allowing for the use of mini-batch optimization techniques. Other approaches include using general linear transformations to compress knowledge (Lázaro-Gredilla et al., 2010) and sparse pseudo-inputs Gaussian Processes methods (Snelson & Ghahramani, 2005). More recently, (Burt et al., 2019) investigated convergence rates of sparse Gaussian Process regression and showed that when using a variational inference approach, with high probability the KL divergence can be made arbitrarily small by increasing the number of inducing points slower than the size of the data. However, (Turner & Sahani, 2011) showed that maximizing the ELBO with respect to the variational and hyperparameters as is done in (Titsias, 2009) results in potentially introducing bias in hyperparameter estimation and (Bauer et al., 2016) showed that it comes particularly in the form of overestimation of the noise term. The effect of low precision arithmetic on this bias may be something for future work. Sparse Gaussian Processes have also been used for classification tasks (Hensman et al., 2014).

4. Methodology

4.1. Software Implementation & Baseline Model

To construct our mixed-precision approximate Gaussian Process models, we leveraged the GPyTorch open-source python library (Gardner et al., 2018a). GPyTorch provides an easy to use API for creating Gaussian Processes models. Each model inherits for either an *ExactGP* or *ApproximateGP* class, and includes a *Kernel* object and *Mean* object composed inside.

The *Kernel* object is saved to the *covar_module* attribute, while the *Mean* object is saved to the *mean_module* attribute. As the names imply, the mean and covariance module attributes define the resulting Gaussian Process, $\mathcal{GP}\left(m(\cdot), k(\cdot, \cdot)\right)$ used to model training data. Together, the *Kernel* and *Mean* objects dictate the forward and backward pass during training, as well as prediction during testing and inference.

Our baseline model is of type *ExactGP*, constructed with a *ConstantMean* mean module, with its covariance module defined by the composition of Scale and RBF Kernel functions. To implement our greedy selection algorithms (see sections 4.3 and 4.4) we copied the source code for the *InducingPointKernel* class and made changes to turn off backpropagation for the set of inducing points, as well as explicitly type casting the tensors to their desired precision. The *InducingPointKernel* acts as a wrapper around the covariance and mean modules, modifying their behavior and ensuring the Gaussian Process uses its set of inducing points for its covariance matrix.

We implemented our own inducing point selection strategies, writing source code in Python that can be found in the group's GitHub repository. All inducing point models referenced in the Experiments section used our custom *InducingPointKernel* class, instantiated with the composition of a Scale and RBF kernel for their covariance modules, as well as a *ConstantMean* object for their mean module.

4.2. Datasets

We leverage UCI's Machine Learning Repository (Dua & Graff, 2017), which contains many datasets of varying sizes fit for both regression and classification. Using a shell script and python script in tandem, we curl the datasets in and perform the necessary preprocessing transformations on the data to be ingested by our models for our experiments. Specifically, we consider three datasets of varying sizes to test our approach in a regression setting, "bikes" (Fanaee-T & Gama, 2013), "energy" (A. Tsanas, 2012), and "protein", as outlined in Table 1.

NAME	OBSERVATIONS	# DIMENSIONS
ENERGY	768	8
BIKE	17,379	16
PROTEIN	45,730	8

Table 1. Datasets & Dimension

By using a varying range of datasets we are able to explore how the performance of our low precision methods vary with respect to dimensionality of the data. Initially, we wanted to investigate even larger datasets (10^5 rows), however our current implementation does not lend itself to larger datasets (see Section 7).

4.3. Greedy Selection: *"Max Greedy"*

We present two general approaches to greedily selecting points as an alternative to the gradient based method. These greedy selection approaches are of particular use in "high dimensional spaces as the number of variables becomes very large." (Titsias, 2009) As one strategy for selecting inducing points, we use the approach outlined in (Titsias, 2009): and visualized in Algorithim 1.

We refer to the above approach as the *"Max Greedy"* selection strategy. In testing the *Max Greedy* approach, we also vary the number of gradient descent steps in the hyperparameter maximization (M) step, as well as the size of J, which determines how many candidate inducing points are tested at each expectation (E) step of our algorithm. We examine the total time and accuracy of this method under different precision regimes.

With respect to time complexity, this approach has a time complexity of $\mathcal{O}(|J|)$, where $|J|$ is the cardinality of set J. It has a fixed memory constraint of $\mathcal{O}(n \times d)$, where n is

Algorithm 1 *Max Greedy* Selection Algorithm

Input: m, the current set of inducing points
Input: J, randomly chosen candidate set of inducing points of cardinality s, $J \subset n - m : |J| = s \leq n - m$
Output: j, inducing point selected maximizing Δj

```
1:  procedure MAXGREEDY($m$, $J$);
2:      $max \leftarrow None$
3:      $j \leftarrow None$
4:      for $candidate$ in $J$ do
5:          $m' \leftarrow \{m, candidate\}$
6:          $current \leftarrow \Delta(m')$
7:          if $current > max$ then
8:              max $\leftarrow current$
9:              $j \leftarrow candidate$
10:         end if
11:     end for
12:     return $j$
13: end procedure
```

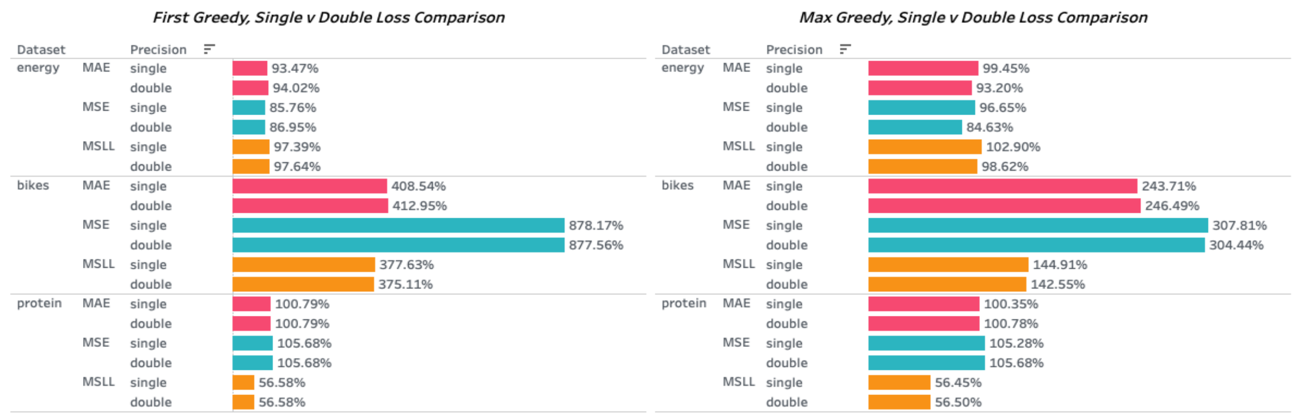

Figure 2. Test Error/Loss as percentage of baseline model, comparison between precision and greedy selection approaches. For *Max Greedy*, 20 total samples are averaged per dataset, 5 for each $|J| \in \{10, 20, 40, 60\}$.

the number of observations in the training set, and d is the dimensionality of said training data. This is because each training datapoint must be copied and saved to a variable representing the set of candidate inducing points to be selected from.

4.4. Greedy Selection: *"First Greedy"*

We additionally test an alternative greedy selection method that we refer to in this paper as *"First Greedy"*. In *First Greedy*, rather than find the candidate inducing point that best increases the marginal likelihood during the each expectation (E) step, we select the first candidate inducing point to increase the marginal likelihood. This approach is once again evaluated under different precision regimes.

With respect to time complexity, this approach has a best case time complexity of $\mathcal{O}(1)$, and a worst case of $\mathcal{O}(|J|)$. Like *Max Greedy*, *First Greedy* has a fixed memory constraint of $\mathcal{O}(n)$, as each training datapoint must be copied and saved to a variable representing the set of candidate inducing points to be selected from.

4.5. Model Evaluation

During our experiments, we evaluated model performance on training time for 100 gradient descent iterations for the baseline model, and time to accumulate 100 inducing points for our variable precision approximate greedy method models. After training, each model was evaluated on Mean Squared Error (MSE), Mean Absolute Error (MAE) and Mean Standardized Log Loss (MSLL). Interested in the effects on the covariance matrix when switching to lower precision, we also evaluated the composition of the covariance matrix using a heat map (see Section 6).

Algorithm 2 *First Greedy* Selection Algorithm

Input: m, the current set of inducing points
Input: J, candidate set of inducing points of cardinality $n - m$, $J = \{n \notin m\} : |J| = n - m$
Output: j, inducing point selected maximizing Δj

```
1:  procedure FIRSTGREEDY($m$, $J$);
2:      $current \leftarrow \Delta(m)$
3:      for $candidate$ in $J$ do
4:          $m' \leftarrow \{m, candidate\}$
5:          $new \leftarrow \Delta(m')$
6:          if $new > current$ then
7:              return $j$
8:          end if
9:      end for
10: end procedure
```

5. Experiments

To establish a reasonable benchmark of training time and performance, we ran our baseline model (as described in Section 4.1) 5 times for each dataset, initializing different random seeds each experiment. Baseline model training time, training loss, and test metrics can be seen in Table 2. While beginning experimentation, we tested running our baseline model in both single and double precision. We noticed virtually no impact on test performance, with significant reduction in training time duration. As such, all baseline model results reported in this paper are ran with single precision, making it harder for our approximate methods to perform well in terms on training time duration.

For our approximate inference investigation, we ran over 150 Gaussian Process models. Our experiments were primarily concerned with accomplishing the following three goals:

DATASET	TRAIN TIME (SECONDS)	TRAIN LOSS	TEST MSE	TEST MAE	TEST MSLL
ENERGY	0.76	4.86	8.72	2.31	2.16
BIKES	133.20	1,059.90	4,599.87	23.16	191.05
PROTEIN	2,784.29	14.94	42.10	5.21	7.94

Table 2. Baseline Model Performance, 100 Training Iterations, Single Precision

1. Investigate the trade-off between training time reduction and worsened test performance caused by switching from double (64-bit) precision to single (32-bit) precision

2. Comparing training time and test loss performance of the two greedy selection algorithims described in Section 4.4 (*First Greedy*) and Section 4.3 (*Max Greedy*)

3. Determine how varying the maximum size of the subset of candidate inducing points, J, affects training time and test performance for *Max Greedy* models.

All results reported for approximate methods are for models with 100 inducing points and 100 kernel hyperparameters optimization steps, experimenting on all three UCI datasets (Dua & Graff, 2017) mentioned in Table 1. All experiments were ran on the CPU of a Macbook Pro with 32GB of RAM and an Apple M1 Pro chip with 10 cores (8 performance, and tx2 efficiency).

6. Discussion

In this section we discuss the findings of our experiments as they pertain to the three research goals outlined in Section 5. Full data tables can be found in the appendix table 3.

6.1. Comparison Between Single and Double Precision

From our experiments, we noticed that switching our Gaussian Process models to train in single precision often resulted in significant time reductions, with minimal increased training and testing error. In Figure 1, we visualize the training times of double and single models under different greedy selection strategies, expressed as a percentage of the training time of the baseline model. In Figure 2 we investigate the repercussions from switching from double to single precision with respect to the loss metrics, again expressed as a percentage of the baseline model's loss for the respective dataset.

From Figure 1, we can see that when using the *Max Greedy* algorithm switching from double to single precision resulted in a 1.5-2x speed increase. Likewise, when using the *First Greedy* algorithm we note a 1.2-1.5x speed increase. For the cases of the bikes and protein datasets, the resulting training time was a much smaller fraction of the baseline models training time, ranging from a low of 0.21% for the protein dataset when using the *First Greedy* algorithm, to a high of 45.53% when using the *Max Greedy* algorithm on the bikes dataset. As visualized in Figure 2, in most cases the resulting impact from switching to lower precision on testing loss was 1-3% of the baseline model's accuracy. This finding reaffirms our hypothesis that when using approximate methods switching to low precision does not increase our testing error significantly. It is important to note that the overall performance on loss metrics as a percentage of the baseline model's performance varied widely between datasets.

Interestingly, no time savings were noticed when applying our approximate inference techniques to the energy dataset. We hypothesize that this time increase is due to the fact the energy dataset is relatively small (768 observations, 8 dimensions). The baseline model trained for 100 iterations remarkably fast fast at only .76 seconds (see Table 2). The *Max Greedy* approach took 7-13x longer, most likely due to the overhead associated with the greedy selection process. We note that approximate GP inference methods seem to be best used on medium-to-large data sets (10^3 to 10^6 rows).

6.2. Comparison between greedy selection algorithims

Model performance between *Max Greedy* and *First Greedy* displayed yet another nuanced version of the trade-off between training time and accuracy. As seen in Figure 1, the *Max Greedy* algorithm tended to take roughly 10x more time to train than its *First Greedy* counterpart. In Figure 2, the impacts of each technique are most visible in the bikes dataset, where the *Max Greedy* algorithm performed 3.1x worse than the baseline on MSE, while the *First Greedy* version performance 8x worse. However, we noticed little difference in performance between *First Greedy* and *Max Greedy* in the energy and protein datasets, with both models nearly matching the baseline performance for MAE and MSE, while halving the MSLL.

6.3. Comparison between size of subset J for *Max Greedy*

As we expected, training times for models using the *Max Greedy* inducing point selection algorithm increased linearly as the subsets of candidate points, (J), increased. This phenomenon is illustrated by Figure 3 in the protein

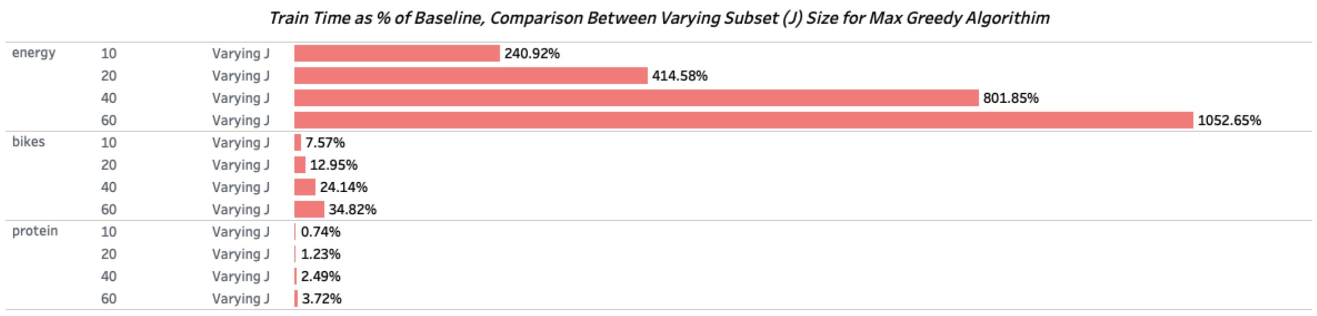

Figure 3. Average effects on Experiment Time for varying size of inducing point candidate subset, J, as % of baseline model. Includes experiment data ran for both single and double precision *Max Greedy* models.

and bikes dataset. As explained earlier, we did not see this time improvement displayed in the energy data set, as the baseline model completed its training in 0.76 seconds. The smallest subset size of candidate points we investigate when using the *Max Greedy* approach was a set of 10 points, which for the bikes and protein datasets resulted in training times 1-7.5% that of the baseline. The largest subset size of candidate points was 60, which took roughly 3.7-7.6% of training time that the baseline model required, roughly 6x more than its 10-subset-size counterpart. Contrary to what we expected, we did not notice substantial performance increases when increasing the size of the inducing point candidate subset. More discussion on this issue in Section 7.

6.4. Comparison of Covariance Matrices

we can see in Figure 5 that the number of elements which contain a nonzero value for the covariance between our chosen inducing points is significantly reduced. This is due to the fact that in lower precision, the range of the values which are able to be represented is lower, and therefore very small values are 'zeroed out'. This, however, does not necessarily mean that all of the elements which are zeroed out in the single precision covariance matrix but not zero in the double precision are below the single precision representation threshold in the double precision version. It is instead possible that some of the data was zeroed out in intermediate calculations, resulting in a zero value in the final single variance covariance matrix. It is perhaps most notable from this visualization that despite a significant proportional of the data having been 'zeroed out', the test accuracy remains comparable to the double precision version. We hypothesise that this is because the single precision retains only the most important information and is able to, in effect, discard the noise inherent in some of the GP inference procedure. Therefore we receive a 'free lunch' in terms of training time without having to pay heavily in terms of performance.

Figure 4 shows how the percentage of the inducing variable covariance matrix elements which are zeroed out varies as we increase the number of inducing points. It is precisely this difference in the two curves which provides an explanation as to why the single precision version has a lower training time. Considering that the number of elements in the inducing point covariance matrix scales quadratically with the number of inducing points then if the percentages between the two curves remains constant, the difference in the number of zero elements grows quadratically. This means that as we increase the number of inducing points, the speedup achieved through low precision methods will only grow. We see this affect in Figure 4 whereby the difference in the relative times taken to perform the GP inference for the different precision's is increasing as we scale the number of inducing points.

7. Project Limitations and Future Work

Our experiments, while proving useful for illustrating time related performance gains when switching to lower precision and leveraging approximate inference methods, suffer from a few drawbacks.

7.1. Normalization & Numerical Stability

For our experiments, we did not first normalize our training and testing data before performing our analysis. Data normalization is a best practice when using Gaussian Processes, showing significant gains in their predictive power when using a kernel function that computes distances between points. As such, when using the RBF kernel, unnormalized data makes tuning the length-scale parameter difficult, and distance between points cannot be represented effectively.

During our project, we briefly explored normalizing our data before training, but unfortunately the resulting covariance matrices would not be positive semi definite. We explored leveraging GPyTorch's settings module, increasing

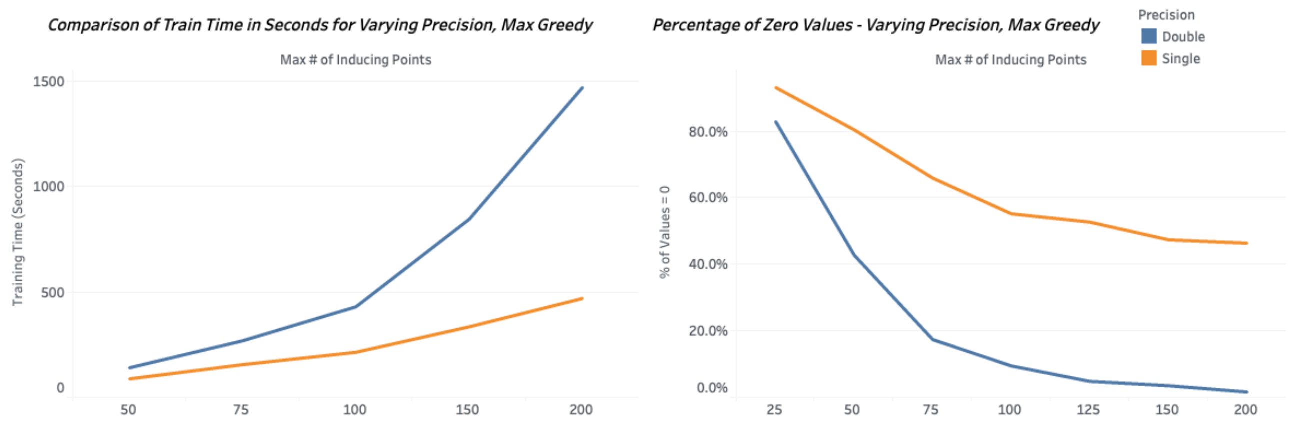

Figure 4. As we increase the number of inducing points in the single precision version, the percentage of elements in the inducing point covariance matrix which are zero remains significant relative to double precision. As a consequence, the speedup gained from using lower precision's increases with more inducing points.

the jitter added as well as increasing the amount of times that jitter would be added before throwing an exception.

For single precision we increased the jitter added to the diagonal of the covariance matrix from $10e^{-4}$ to $10e^{-3}$, while for double precision we increased the jitter from $10e^{-6}$ to $10e^{-4}$. We also changed the maximum amount of times GPyTorch would add jitter, from 3 to 10 before throwing an exception.

Despite these changes, we we're not able to run our experiments for a meaningful set of inducing points without generating a non-PSD covariance matrix. Ultimately, our experiments were intended to explore the *relative* time and accuracy differences of mixed-precision approximate GP methods, *absolute accuracy* was not of utmost importance. Based on our experiments, we believe that future work relating to numerical stability could greatly benefit the use of approximate and low-precision applications to

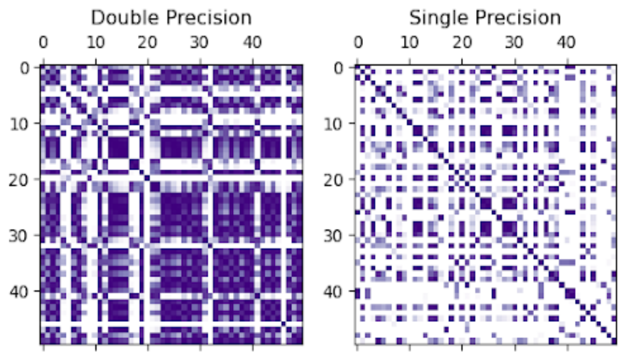

Figure 5. Covariance Matrix Comparison. A white element indicates that the value of the covariance between the two points is zero. This is for the MaxGreedy version with 50 inducing points.

Gaussian Process inference, making their inference powers much more accurate.

7.2. GPU & Greedy Selection Optimization

While GPyTorch offers an *InducingPointKernel* to support approximate inference models, the implementation only allows for gradient-descent based inducing point selection strategies. In order to implement our greedy selection algorithms, we manipulated GPyTorch source code, turning off gradient descent properties of the inducing points and enforcing strict tensor types to ensure the correct precision was being ran (see section 4.1).

Our resulting implementation was not optimized for GPU usage, and thus, batch-based learning was not an option. When trying to run on an author's GPU (a NVIDIA 2060 SUPER with 8GB of RAM), we frequently encountered memory related errors for our larger datasets (protein and road3d specifically). We suspect that these errors are a direct result of our un-optimized greedy selection code, as each approach we explored requires copying a full set of training data into RAM.

We are interested in exploring different approaches to enable GPU batch-based learning. For instance, if the greedy selection algorithms sample their candidate inducing points from a batch of training data loaded on the GPU, rather than a full copy of training data. We hypothesize that if correctly implemented, running the training on a GPU would be a viable strategy, leading to decreased training times and a viable path forward for analysis on larger datasets (100K+ rows).

7.3. Gradient-based Approaches & Alternative Kernels

For our baseline model, we elected to compare our experiment results against a vanilla *ExactGP* model, equipped with a *ConstantMean* mean module and a covariance module consisting of the composition of a Scale and RBF kernel.

For comparison between approximate methods, we are interested in continued exploration if = the accuracy and time trade-offs by comparing our greedy selection strategies against gradient-based inducing point selection methods. Furthermore, we are interested in understanding how the choice of covariance module effects the time complexity of each approach. Further experimentation could be done with Matern, Periodic, and Spectral kernels for instance, as well as exploring performance when operating with larger sets of inducing points.

7.4. Half Precision

We were interested in running experiments in half-precision (16-bit floating point numbers) however GPyTorch uses PyTorch's *linalg* API, and the implementation for the Cholenski decomposition linear-algebra routine does not currently support half precision data. Thus, we we're unable to explore model time and accuracy performance in half precision. If possible in the future, we'd like to conduct a thorough investigation into this.

8. Conclusion

We have shown preliminary results that using lower precision in approximate Gaussian Process techniques is indeed worthwhile and beneficial. Particularly in the setting where we are willing to sacrifice a small increase in test error for significant time savings. Therefore time-sensitive applications of Gaussian Process would be suited to using the greedy low-precision techniques which we describe in this paper. We have provided experimental evidence for these finding and discussed some potential future directions for exploration to take this idea further.

References

A. Tsanas, A. Xifara. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, 49: 560–657, 2012.

Bauer, M., van der Wilk, Mark, and Rasmussen, Carl Edward. Understanding probabilistic sparse gaussian process approximations. In *NIPS*, 2016.

Burt, David R., Rasmussen, Carl Edward, and van der Wilk, Mark. Rates of convergence for sparse variational gaussian process regression. In *International Conference on Machine Learning*, 2019.

Candela, Joaquin Quiñonero and Rasmussen, Carl Edward. A unifying view of sparse approximate gaussian process regression. *J. Mach. Learn. Res.*, 6:1939–1959, 2005.

Chen, Yunji, Luo, Tao, Liu, Shaoli, Zhang, Shijin, He, Liqiang, Wang, Jia, Li, Ling, Chen, Tianshi, Xu, Zhiwei, Sun, Ninghui, and Temam, Olivier. Dadiannao: A machine-learning supercomputer. *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622, 2014.

de G. Matthews, Alexander G., Hensman, James, Turner, Richard E., and Ghahramani, Zoubin. On sparse variational methods and the kullback-leibler divergence between stochastic processes. In *International Conference on Artificial Intelligence and Statistics*, 2015.

Dua, Dheeru and Graff, Casey. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Fanaee-T, Hadi and Gama, Joao. Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence*, pp. 1–15, 2013. ISSN 2192-6352. doi: 10.1007/s13748-013-0040-3. URL `[WebLink]`.

Gardner, Jacob R., Pleiss, Geoff, Bindel, David, Weinberger, Kilian Q., and Wilson, Andrew Gordon. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration, 2018a. URL `https://arxiv.org/abs/1809.11165`.

Gardner, Jacob R., Pleiss, Geoff, Bindel, David S., Weinberger, Kilian Q., and Wilson, Andrew Gordon. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *NeurIPS*, 2018b.

Gupta, Suyog, Agrawal, Ankur, Gopalakrishnan, K., and Narayanan, Pritish. Deep learning with limited numerical precision. In *ICML*, 2015.

Hammerstrom, Dan W. A vlsi architecture for high-performance, low-cost, on-chip learning. *1990 IJCNN International Joint Conference on Neural Networks*, pp. 537–544 vol.2, 1990.

Hensman, James, Fusi, Nicoló, and Lawrence, Neil D. Gaussian processes for big data. *ArXiv*, abs/1309.6835, 2013.

Hensman, James, de G. Matthews, Alexander G., and Ghahramani, Zoubin. Scalable variational gaussian process classification. In *International Conference on Artificial Intelligence and Statistics*, 2014.

Lázaro-Gredilla, Miguel, Candela, Joaquin Quiñonero, Rasmussen, Carl Edward, and Figueiras-Vidal, Aníbal R. Sparse spectrum gaussian process regression. *J. Mach. Learn. Res.*, 11:1865–1881, 2010.

Maddox, Wesley J., Potapczynski, Andres, and Wilson, Andrew Gordon. Low-precision arithmetic for fast gaussian processes. *ArXiv*, abs/2207.06856, 2022.

Rasmussen, Carl Edward and Williams, Christopher K. I. Gaussian processes for machine learning. In *Adaptive computation and machine learning*, 2009.

Salvucci, Guido D. Ieee standard for binary floating-point arithmetic. 1985.

Seeger, Matthias W., Williams, Christopher K. I., and Lawrence, Neil D. Fast forward selection to speed up sparse gaussian process regression. In *International Conference on Artificial Intelligence and Statistics*, 2003.

Snelson, Edward and Ghahramani, Zoubin. Sparse gaussian processes using pseudo-inputs. In *NIPS*, 2005.

Sun, Xiao, Wang, Naigang, Chen, Chia-Yu, Ni, Jiamin, Agrawal, Ankur, Cui, Xiaodong, Venkataramani, Swagath, Maghraoui, Kaoutar El, Srinivasan, Vijayalakshmi, and Gopalakrishnan, K. Ultra-low precision 4-bit training of deep neural networks. In *Neural Information Processing Systems*, 2020.

Titsias, Michalis K. Variational learning of inducing variables in sparse gaussian processes. In *AISTATS*, 2009.

Turner, Richard E. and Sahani, Maneesh. Two problems with variational expectation maximisation for time-series models. 2011.

Zhang, Tong. Learning bounds for kernel regression using effective data dimensionality. *Neural Computation*, 17: 2077–2098, 2005.

A. Appendix

Here we will list the items we have changed since the midterm report:

- We provided a complete set of experimentation results and analysed and provided explanation of the results which we saw.
- We adjusted our testing metrics to investigate testing error rather than the training loss as this provides a better, standardized comparison between our results.
- We provided a list of possible future directions and approaches which could be adopted to extend and improve the current project setup.
- We included alternative data sets to our experimentation to be able to better explore performance, dropping the 3droad dataset and including the protein dataset.
- We also looked at how the number of zero elements in the covariance matrix changes as we increase the number of inducing points.
- We did not investigate the affect of using different precision for the gradient based inducing point method. We gave an explanation of this in the discussion section and mentioned it in the future work section.

MODEL	DATASET	DTYPE	TRAIN TIME (SECONDS)	TRAIN LOSS	TEST MSE	TEST MAE	TEST MSLL
FIRST GREEDY	BIKES	DOUBLE	4.31	4,734.25	40,366.59	95.63	716.63
FIRST GREEDY	BIKES	SINGLE	2.54	4,750.86	40,394.84	94.61	721.45
FIRST GREEDY	ENERGY	DOUBLE	0.7	4.09	7.58	2.17	2.11
FIRST GREEDY	ENERGY	SINGLE	0.86	4.11	7.48	2.16	2.1
FIRST GREEDY	PROTEIN	DOUBLE	9.06	7.46	44.49	5.25	4.49
FIRST GREEDY	PROTEIN	SINGLE	5.96	7.46	44.49	5.25	4.49
MAX GREEDY (J=60)	BIKES	DOUBLE	57.98	1,709.49	14,004.06	57.08	272.34
MAX GREEDY (J=60)	BIKES	SINGLE	34.78	1,751.41	14,158.69	56.44	276.84
MAX GREEDY (J=60)	ENERGY	DOUBLE	10.09	4.85	7.38	2.15	2.13
MAX GREEDY (J=60)	ENERGY	SINGLE	5.82	5.42	8.43	2.3	2.22
MAX GREEDY (J=60)	PROTEIN	DOUBLE	125.06	7.46	44.49	5.25	4.48
MAX GREEDY (J=60)	PROTEIN	SINGLE	68.05	6.53	44.33	5.22	4.48
VARYING J	BIKES	0	3.42	4,742.55	40,380.72	95.12	719.04
VARYING J	BIKES	10	10.09	2,586.98	20,930.24	63.07	377.98
VARYING J	BIKES	20	17.24	2,111.87	17,023.50	58.76	319.98
VARYING J	BIKES	40	32.16	1,833.15	14,351.39	54.96	275.18
VARYING J	BIKES	60	46.38	1,730.45	14,081.37	56.76	274.59
VARYING J	ENERGY	0	0.78	4.10	7.53	2.16	2.1
VARYING J	ENERGY	10	1.82	4.87	7.68	2.2	2.15
VARYING J	ENERGY	20	3.13	4.92	7.61	2.19	2.15
VARYING J	ENERGY	40	6.06	4.90	7.60	2.19	2.15
VARYING J	ENERGY	60	7.95	5.14	7.91	2.22	2.17
VARYING J	PROTEIN	0	7.51	7.46	44.49	5.25	4.49
VARYING J	PROTEIN	10	20.63	7.46	44.49	5.25	4.49
VARYING J	PROTEIN	20	34.13	7.57	46.41	5.27	269.49
VARYING J	PROTEIN	40	69.35	7.58	44.81	5.24	4.53
VARYING J	PROTEIN	60	103.68	7.11	44.43	5.24	527.81

Table 3. Full Experiment Results, data for each row is the average of unique 5 experiments.