GPU-accelerated GP Regression with GPytorch

Alex Karonen

February 22, 2024

Abstract

This report studies the capabilities of GPU-accelerated Gaussian Process regressions using a PyTorch based Gaussian Process (GP) framework called GPytorch. GPs are known to be good statistical models for learning from data with uncertainty considered. However, with larger datasets, the GP regression suffers from computational inefficiency due to large matrix operations such as $N \times N$ matrix inversions, where N is the number of training samples. By utilizing GPUs for computation, it is possible to accelerate the computation down to near constant time with even tens of thousands of training samples, with only memory capacity being a limiting factor.

1 Introduction

Consider that we have obtained some observations from some process $X(t)$ and we want to model this process. Generalized regression would fit to the observations by optimizing a curve to go through all the observation points and nothing else. This means that we need to be certain of the mathematical model to predict new data correctly. However, in most cases this is not the situation due to complex system behavior which is hard to formulate in a closed form and noise in measurements which is why there is a need for implementing uncertainty in the model. This is exactly what a Gaussian Process (GP) regression brings to the table. The ability to give the model some prior information about the system through the choice of used covariance kernel and the uncertainty in the prediction are the key points why GP regression is the choice of method in modern data-analysis applications (Wang, 2023).

Although a good alternative for traditional generalized regression, GP regression can become computationally expensive when larger datasets are considered. This is where GPU-acceleration becomes handy. GPUs are for the most part faster when it comes to certain mathematical computations (such as matrix operations) than traditional central processor units, which helps when dealing with heavy computational tasks such as large dataset GP regressions. The only limiting factor with GPUs is the amount of memory as the matrices in GPs get exponentially larger when the number of training samples gets larger.

2 Model

In this report we consider model observations of the form

$$y(t) = f(t, \theta) + \epsilon, \tag{1}$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is gaussian noise in the observations and $f(t, \theta)$ is an unknown signal with $t \in \mathbb{R}$ and some parameters θ.

We can form the joint probability density for this model as

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{f}^* \end{pmatrix} \sim \mathcal{N}\Bigg(\begin{pmatrix} \mu \\ \mu^* \end{pmatrix}, \begin{pmatrix} C(t^*,t) + \sigma^2 \mathbf{I} & C(t,t^*) \\ C(t^*,t) & C(t^*,t^*) \end{pmatrix} \Bigg) \tag{2}$$

and the posterior predictive distributions as

$$p(\mathbf{f}^*|\mathbf{y}) = \mathcal{N}(\mathbf{f}^*_{est}, \mathrm{Cov}(\mathbf{f}^*)) \tag{3}$$

where

$$\begin{aligned} \mathbf{f}^*_{est} &= C(t^*,t)(C(t,t) + \sigma^2\mathbf{I})^{-1}(\mathbf{y} - \mu) + \mu^* \\ \mathrm{Cov}(\mathbf{f}^*) &= C(t^*,t^*) - C(t^*,t)(C(t,t) + \sigma^2\mathbf{I})^{-1}C(t,t^*). \end{aligned} \tag{4}$$

and $C(\cdot)$ is the covariance kernel of the model, $t \in \mathbb{R}$ are the training inputs (observations) and $t^* \in \mathbb{R}$ are the test inputs and their corresponding values f^*. Typical covariance kernels are an Radial basis function (squared exponential, RBF) kernel

$$C(t,t^*) = \exp(-\frac{||t - t^*||^2}{2\sigma^2}),\ t, t^* \in \mathbb{R} \tag{5}$$

or a Matèrn covariance kernel

$$C(t,t^*) = \frac{2^{1-v}}{\Gamma(v)} \left(\frac{||t - t^*||}{\ell} \right)^v K_v\left(\frac{||t - t^*||}{\ell} \right),\ t, t^* \in \mathbb{R} \tag{6}$$

where $v > 0$ is a smoothness parameter of the kernel, ℓ is a length parameter, Γ is the gamma-function and K_v is a modified Bessel function of second kind (order v).

In the case considered in this report, there is no prior on the model and the covariance kernel used is a RBF-kernel. The model likelihood is

$$p(\mathbf{y}|f^*) = \frac{1}{\sqrt{2\pi(C(t,t) + \sigma^2\mathbf{I})}} \exp\Bigg(-\frac{1}{2}(\mathbf{y} - \mu)^T (C(t,t) + \sigma^2\mathbf{I})(\mathbf{y} - \mu) \Bigg). \tag{7}$$

3 Methods

Gaussian process regression is one way of probabilistic regression to learn parameters from given data. Given some set of pairs of input-output values $(\mathbf{t}_i, y_i), i = 1, 2, 3, \ldots, N$, it is possible to compute a predictive distribution values f^* at some test input locations t^*.

Gaussian process regression assumes a GP prior for the functions values which is formulated as

$$p(\mathbf{f}|\mathbf{T}) = \mathcal{N}(\mu, \mathbf{C}), \tag{8}$$

where $\mathbf{f}$ is all the latent function values at inputs $\mathbf{T} = \mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}, \ldots, \mathbf{x_n}$, μ is the mean function (often assumed to be either zero or constant) and $\mathbf{C}$ is a covariance matrix formulated like in the Equation 2 where the entries are given by a kernel function $C(t, t^*)$. The chosen kernel function used in a specific case formulates assumptions such as similarity of samples and differentiability about the ground-truth function which we want to learn. The predictive posterior samples are obtained by putting a GP prior on the training set, formulating the likelihood of the training samples $\mathbf{y}$ given the latent function values $\mathbf{f}$ and combining these using the Bayes rule. Thus the joint posterior looks like

$$p(\mathbf{f}, \mathbf{f}^*|\mathbf{y}) \propto p(\mathbf{f}, \mathbf{f}^*)p(\mathbf{y}|\mathbf{f}) \tag{9}$$

By marginalizing out the training set latent variables we obtain the wanted predictive posterior form shown in Equation 3.

The package used in this report is GPytorch (Gardner et al., 2018) which is a scalable Gaussian Process focused toolbox built on the PyTorch-framework. GPytorch implements a novel method called Blackbox Matrix-Matrix multiplication (BBMM) for computing the relevant matrix operations for GPs. Compared to naive methods, BBMM reduces the $\mathcal{O}(n^3)$ complexity down to $\mathcal{O}(n^2)$ when it comes to computing exact GPs such as in GP regression. In addition the package being built on top of PyTorch gives the opportunity of using GPUs for even larger computational boost.

4 Numerical Examples

Here we have an example case where we have a model from which we have obtained 10 000 training samples from an interval between $[-1, 1]$. Running Gaussian regression on this data gives a pretty accurate representation of the ground-truth model with reasonable confidence on the interval as seen in Figure 1. However outside of the interval, we can observe that the model cannot anymore follow the same pattern, which is also displayed in the confidence of the model. The example here gives us a indication that the model works and thus can be tested with regards to the training time between different computational units.

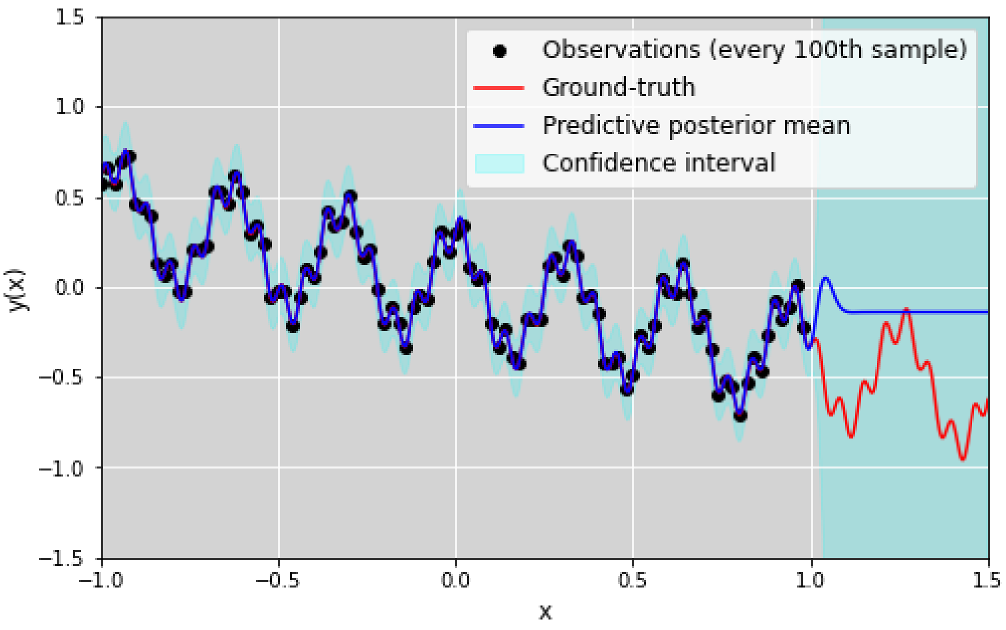
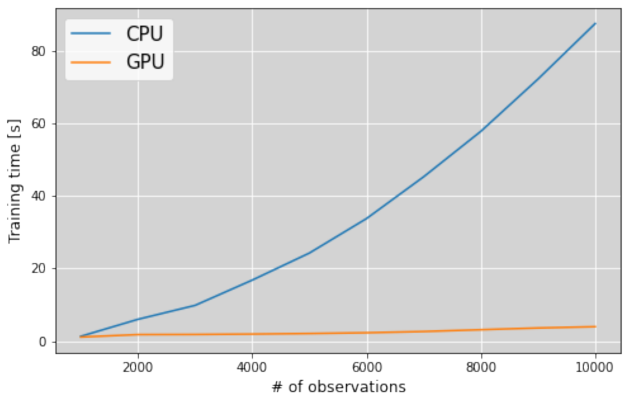
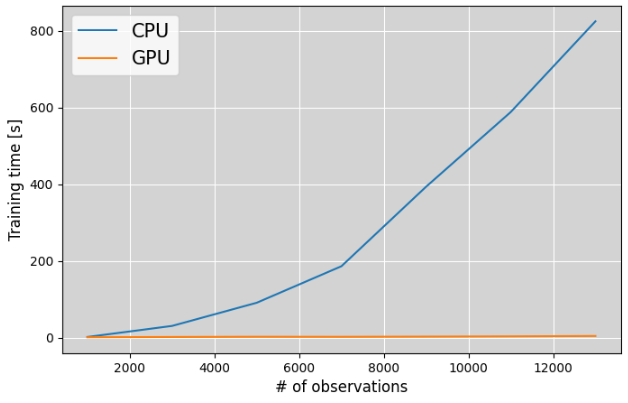


Figure 1: Example model posterior predictive mean over 10 000 training samples

The main focus of this report is to present the applicability of GPU-acceleration on GP regressions. The model presented was tested on two different machines with a varying computational capability. The first machine is my own home PC with a AMD Ryzen 5700X (CPU) and a NVIDIA RTX 4070 (GPU). The other tested machine is a Google Colaboratory -virtual machine with a Intel Xeon central processor unit and a NVIDIA V100 data center GPU. Training on the model was done for 50 iterations per run with each run having more training samples than the previous one. From Figures 2a and 2b we can see the comparison between the CPU and GPU training times on the machines.



(a) GPU vs. CPU training times (NVIDIA RTX 4070 vs AMD Ryzen 5700X)



(b) GPU vs. CPU training times (NVIDIA V100 vs Intel Xeon 2.3GHz)

5 Conclusions

Using GPytorch for this assignment was a very linear process. The package offers a wide range of modules all of which I could not get familiar with within the scope of this project. The modules that were used are thoroughly documented, and the implementation was easy an straight forward with the help of their example notebooks. The most complicated part about using GPytorch with GPUs would be the linking PyTorch with CUDA however I had already done that so I had no issues when it comes to installing and getting the package to operate.

From testing the computation speeds over large datasets we could determine that the GPU trains the GP regression model in almost constant time when compared to the CPU training time. However the GPUs computing capability is limited by its memory capacity as the covariance matrices needed to compute the GPs get exponentially larger. With thousands of training samples we are already talking about GBs of memory just to store a one $N \times N$ covariance matrix in *double* -precision.

While this report shows the computational boost that GPUs give when it comes to computing exact GPs, there is a need to reduce the heavy memory requirement with the full-covariance $N \times N$ matrices. With the possibility of implementing sparse covariance matrices, it would be possible to even further improve the capability of GPU-enhanced GPs.

References

Gardner, J. R., Pleiss, G., Bindel, D., Weinberger, K. Q., and Wilson, A. G. (2018). Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*.

Wang, J. (2023). An intuitive tutorial to Gaussian process regression. *Computing in Science & Engineering*, 25(4):4–11.