An exploration on the optimization routines of SVI for GPs

Zehui li *zl432@cantab.ac.uk*

1 Introduction

Gaussian process (GP) models with N instances require a computational complexity of $O(N^3)$ at the training stage because it involves the inverting of a covariance matrix. To extend the use of GPs to large datasets, (Titsias, 2009) proposes a variational inference method to reduce the computational complexity to $O(NM^2)$ with M inducing points. A further reduced computational bound of $O(M^2 min\{M, Bachsize\})$ can be obtained by the use of stochastic variational inference (SVI) (Hensman et al., 2013).

In SVI for GPs, there are three sets of variables to optimize at each iteration: **the inducing variables**, **the covariance parameters**, and **the inducing points locations**. When it comes to the implementation of the SVI for GPs, however, it is not clear what is the best strategy to optimize three sets of variables. One conclusion from the original paper (Hensman et al., 2013) is that to fix the covariance parameters for a few iterations can stable the convergence.

In this report, I expand on this topic and conduct an empirical study on different optimization routines of SVI for GPs: different orders of optimizing three sets of variables are compared against each other. The converging speed and marginal likelihood are used as the criteria to evaluate different optimization routines. In addition, the performances of various types of stochastic optimization algorithms are also compared.

The major contributions of this report are summarized below, and the code for data processing and model building can be found on Colab Notebook[1].

- Apply different optimization routines and algorithms to solve SVI for GP.
- Design experiments and compare the performance of various optimization routines and algorithms on a toy example.
- Test the performance of selected optimization routines and algorithms on a real data set.

[1] https://colab.research.google.com/drive/1mLrL9p9TICFUAN6eZT0rFk21GMFIbsj2

2 Background

In this section, I will review the concepts of SVI for GPs, which is then followed by an overview of different optimization algorithms and the tools I used in this report.

2.1 Review of SVI for GPs

Assume a dataset D of n training points, $D = (X, Y) = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$; each $y_i = f(x_i) + \epsilon$ is generated by a function f with Gaussian noises, where ϵ is the noise of Gaussian distribution with precision β. I will assume a GP prior for f. Then to reduce the computational complexity, I can introduce m inducing points: $\{z_1, z_2, \ldots, z_m\}$ with $m \ll n$ and the inducing variable u will equal to the following:

$$u = \begin{bmatrix} f(z_1) & f(z_2) & \ldots & f(z_m) \end{bmatrix}$$

To apply stochastic variational inference to the GP model, I will first approximate the distribution of u using a variational distribution $q(u) = \mathcal{N}(u|m, S)$, and then the lower bound $\mathcal{L}_3$ of the quantity $p(Y|X)$ can be written as equation-1.

$$\begin{aligned} &\log p(Y|X) \geq \mathcal{L}_3 \\ &= \sum_{i=1}^{n} \{\log \mathcal{N}(y_i|k_i^\top K_{mm}^{-1} m, \beta^{-1}) - \frac{1}{2}\beta \tilde{k}_{i,i} - \frac{1}{2}(S_i)\} \\ &\quad - KL(q(u)||p(u)) \end{aligned} \tag{1}$$

where K_{mm} is the covariance function between all the inducing points, k_i is the i^{th} column of the covariance function between all the inducing points

and training points; $\tilde{k}_{i,i}$ and $_i$ are also expressions involving the above covariance functions.

This lower bound $\mathcal{L}_3$ is used as the objective to minimize during the stochastic variational inference process. There are three sets of parameters to optimize for: the first set, **inducing variables**, includes m and S, which define the distribution of $q(u)$; the second set, **inducing points location**, includes $\{z_1, z_2, \ldots, z_m\}$; the third set of parameters are the **hyper-parameters for the covariance function**, and I denote them as θ. If an RBF kernel is used, for example, θ includes the length scale and variance of the RBF kernel.

2.2 Optimization Algorithms

Different stochastic gradient descent algorithms can be used to optimize three sets of parameters. In this report, I experiment with three distinct algorithms: **Rprop**, **Adadelta**, and **Adam**.

Rprop is the short for **resilient backpropagation** (Riedmiller and Braun, 1993). It is first developed for supervised learning in artificial neural networks, but it can be applied to all types of optimization problems. Different from vanilla sgd, Rprop maintains different learning rates for every parameter, and these learning rates are determined by the sign of changes of the partial derivative. One advantage of Rprop is that it does not require any free parameter values (for example, learning rate).

Adadelta (Zeiler, 2012) also maintains different learning rates for every parameter. It changes the learning rates in such way that parameters associated with infrequent features are updated with a larger learning rate while frequently updated parameters have lower learning rates. It achieves this by referring to a decaying average of past squared gradients. Similarly, **Adaptive Moment Estimation (Adam)** (Kingma and Ba, 2014) computes the adaptive learning rates in a different way. It not only relies on the past squared gradients to compute the learning rates, but it will also refer to the decaying averages of past gradients.

In section-3, three optimization algorithms are applied to SVI for GPs, and I will use them to compare different optimization routines.

2.3 GPy and climin

Two python libraries are used in this report: for the implementation of the GP model, I use **GPy**, which is a GP framework developed by the Sheffield machine learning group, including basic kernel functions and other building blocks for training the GP models. In this experiment, the **Stochastic Variational GP (SVGP)** class is used as the wrapper for the model. Noticeably, the natural gradients are not implemented within this class, and therefore, the parameters of the inducing variables, m, and S, will also be optimized as the other two sets of variables using their gradients. As a result, it takes more iterations for $q(u)$ to converge in our experiment than the original paper.

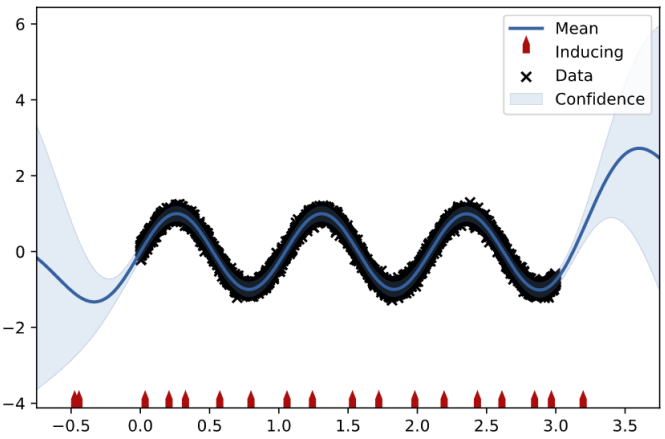

Figure 1: The result of Titsias' variational approach on the toy example. With 20 inducing points, the GP model can capture the trend of original data perfectly.

For the optimization algorithms, **climin** library is used, which provides multiple off-the-shelf optimization algorithms and is consistent with the **GPy** library. I use Rprop, Adadelta, and Adam classes from climin in the experiment.

3 Compare Optimization Routines

In this section, I create a toy example and compare different optimization routines in two situations. The inducing points locations are fixed, and I only need to optimize parameters for covariance function and inducing variables in the first case; secondly, inducing point locations are unfixed and optimized together with the other two sets of parameters.

3.1 Toy Examples and Sparse GP Model

The toy example consists of 5000 data points (x_i, y_i) with $y_i = sin(6 * x_i) + \epsilon$. To fit the data, I use an exponentiated quadratic covariance function and a white variance as the kernel function. Then, I randomly initialize 20 inducing points and use the Titsias' variational approach (Titsias,

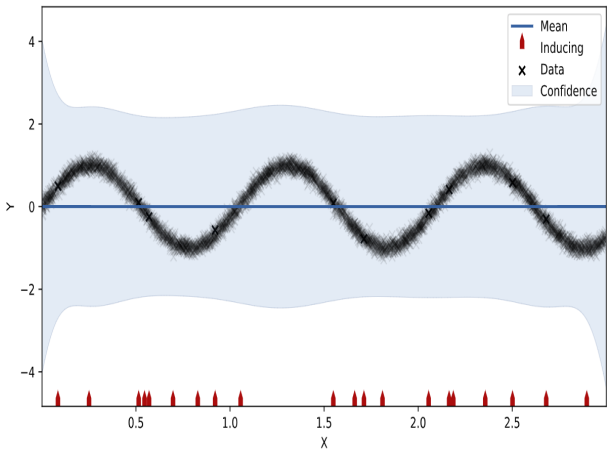

Figure 2: The GP model initialized with 20 inducing points before optimization. The model fits the data poorly.

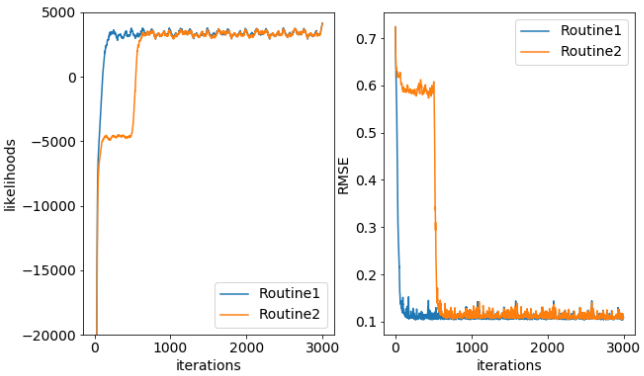

Figure 3: Use Adadelta optimizer to optimize m, S, and θ with two routines. Two figures show the change of **marginal likelihood** and **root mean square error** through the optimization process.

2009) to compute the sparse GP model.

As figure-1 shows, with only 20 inducing points, the model can fit the data very Ill. The full GP model has a log-likelihood of 4337, while the sparse GP model has a likelihood of 4308. The small difference indicates a tight bound on the marginal likelihood. The sparse GP model will be used as a standard to be compared to in the following subsections.

3.2 Optimize with fixed inducing point locations

The first experiment compares two optimization routines with fixed inducing point locations. The inducing point locations are sampled from a uniform distribution within the range of (x_{min}, x_{max}), and the locations of these inducing points are fixed throughout the optimization process. Figure-2 shows the GP model initialized with 20 inducing points. The model is consistent with the data at the location of the inducing points but fits poorly at other locations.

Two optimization routines (which are defined as the following) are used to optimize the inducing variables m, S, and the kernel parameters θ.

Routine 1 m, S, and θ are updated simultaneously throughout the optimization process for 3000 iterations.

Routine 2 Only m and S are updated with fixed θ in the first 500 iterations while m, S, and θ are updated simultaneously after 500 iterations.

I first test two routines with **Adadelta** optimizer, and the result is shown in figure-3, where the marginal likelihood and root mean square error (RMSE) is plotted. Both RMSE and marginal likelihood criteria indicate that the initial model converges to the sparse GP model. The lag of Routine 2 (the yellow line) in the first 500 iterations is because the parameters θ of the kernel function are fixed at the beginning of the iteration. However, the final RMSE and likelihood remain identical, and this shows that fixing the parameters for the kernel function does not improve the performance for **Adadelta** algorithm.

Secondly, I test two rountines with **Adam** and **Rprop** optimizer. **Adam** gives the same result as the **Adadelta** optimizer: routine 2 does not outperform routine 1. On the contrary, for **Rprop** optimizer, rounte 2 does result in a higher likelihood and lower RMSE. Figure-4 shows the resulting GP models generated by rountine 1 and rountine 2. The model optimized with routine 1 fits the data poorly while the GP model optimized with routine 2 does imitate the sparse GP in Figure-1.

In conclusion, when the inducing point locations are fixed, routine 2 is superior to routine 1 for some optimization algorithms (for example, Rprop); however, for other optimization algorithms that can maintain an adaptive learning rate, the difference is not significant. For example, Adam and Adadelta are able to adjust the learning rate by referring to the past gradient of parameters, and therefore they have the same performance under routine 1 and routine 2.

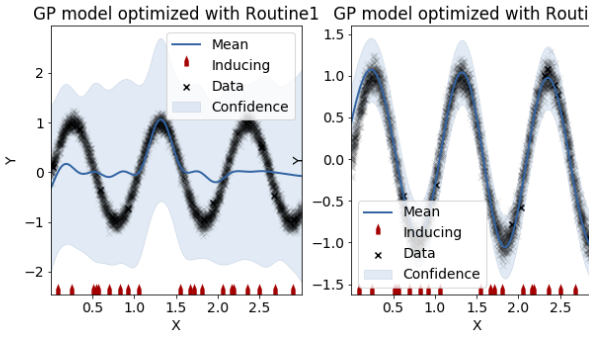

Figure 4: Rprop optimizer generates two completely different models with routine 1 and routine 2. The GP model from routine 1 fits the data poorly.

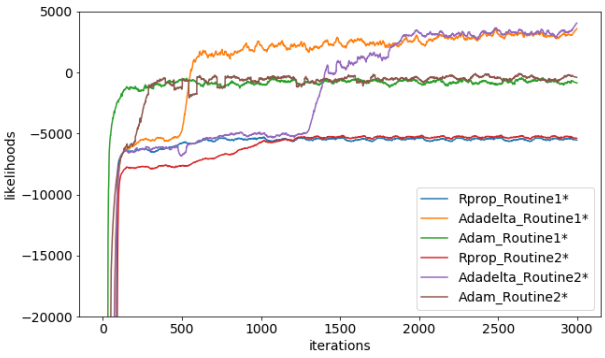

Figure 5: The marginal likelihood of the GP models using three algorithms and two routines. Routine 2* performs better than routine 1*, because the resulting likelihood of routine 2* is larger than routine 1* for all of the optimizer.

3.3 Optimize with unfixed inducing points location

In this subsection, I explore optimization routines with unfixed inducing point locations. In other words, the model will need to update inducing point locations $Z = \{z_1, z_2, \ldots, z_{10}\}$, inducing variable parameters m, S, and the kernel parameters θ at the same time. **Routine 1*** and **routine 2*** are applied to update these parameters.

Routine 1* m, S, θ, and $\mathbf{Z}$ are updated simultaneously throughout the optimization process for 3000 iterations.

Routine 2* Only m, S, and $\mathbf{Z}$ are updated with fixed θ in the first 500 iterations while all the parameters are updated simultaneously after 500 iterations.

Essentially, **Routine 1*** and **routine 2*** the same as **routine 1** and **routine 2**, and the only difference is that Z is optimized together with m and S. However, routine 2* does perform better than routine 1* for all three of the optimizers. Figure-5 shows the marginal likelihood of three of the optimizer with routine 1* and routine 2*, indicating that routine 2* is superior than routine 1* for Rprop, Adadelta, and Adam.

With the result from two experiments, we can empirically conclude that fixing the parameters of kernel function can help with the optimization, especially with unfixed inducing points. The intuition for why fixing θ helps is that θ defines the GP model, and before it is optimized, the inducing point Z and variational distribution $q(u) = \mathcal{N}(u|m, S)$ should have the correct value; otherwise, the optimization applied to θ is effortless.

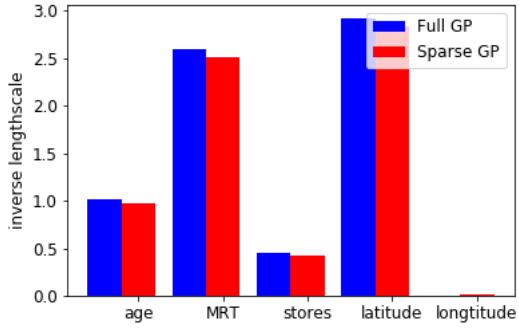

Figure 6: The inverse lengthscale of five features. MRT and latitude (location) are the determinate factors to decide the price of the house. The red bar is computed using sparse GP, which is almost the same as the parameters of the Full GP model.

When it comes to the performance of three optimization algorithms, as figure-5 shows, **Adadelta** turns out to be the best optimizer for solving the SVI for GPs. It has a much higher marginal likelihood than **Adam** and **Rprop**. In the next section, we will test the conclusion we get from the toy example on the real data set.

4 Working with the Real Data

The historical real estate data (Yeh and Hsu, 2018) in the Taiwan region is used in this report. The model will predict the house price according to five features: the house age, the distance to the nearest Metro Rail Transit System (MRT), the number of convenience stores in the living circle, the latitude, and the longitude. The kernel function is created by the multiplication of five RBF

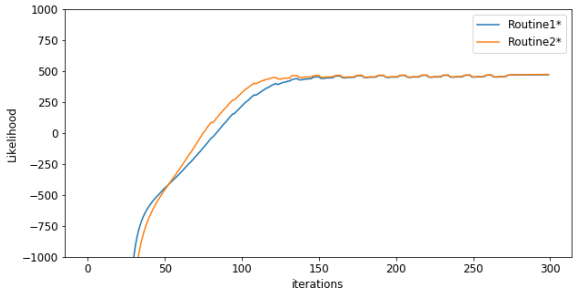

Figure 7: Adadelta is used to optimize the GP model with routine 1* and routine 2* separately. Routine 2* is superior because it helps the likelihood to converge faster than routine 1*.

kernels.

I first standardize every feature and then apply full GP and sparse GP to the problem. Figure-6 shows the inverse lengthscale of five features, where the red bar is computed by sparse GP while the blue bar is computed by full GP. The result is consistent with the common knowledge: the price of houses is mainly determined by the location of the house and the distance to the closest transportation station. Another observation is that sparse GP with 30 inducing points can approximate the full GP very well. Noticeably, there are only 414 data points in the training set, and 30 is not very much smaller than 414. The reason why we need relatively more inducing points is that the dimensions of the data are high, and we will expect more inducing points with the increase of dimensions of the data.

Now we will test the following hypotheses, which are obtained from the section-3.

Hypothesis 1 Routine 1* is superior to Routine 2* in solving SVI for GP.

Hypothesis 2 Adadelta optimization algorithm is superior than Adam and Rprop in solving SVI for GP.

To verify **hypothesis 1**, **Adadelta** is used to optimize SVI for GP with routine 1* and routine 2* separately. The batch size and the number of inducing points are both set to 30. The algorithm runs for 300 iterations in both routines, but for routine 2*, the kernel parameters θ are fixed for the first 50 iterations. The likelihood change is shown in Figure-7. It indicates that routine 2* is better than routine 1* because it helps the likelihood to converge faster: routine 2* converges around 50 iterations ahead of routine 1*.

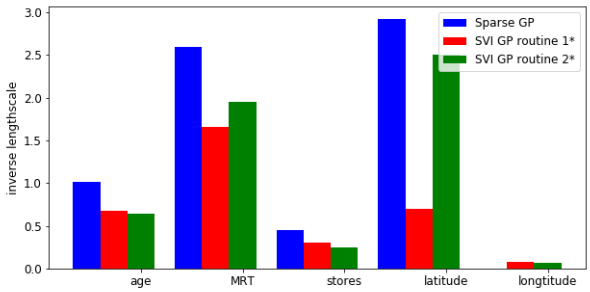

Figure 8: Two SVI GP models optimized with routine 1* and routine 2* are compared with the best approximate GP (sparse GP). With routine 2*, the length scale of two key features (latitude and MRT) are closer to sparse GP model than routine 2.

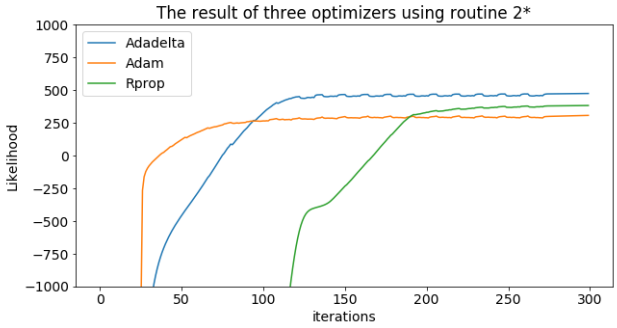

Figure 9: Three optimizers are applied to optimize SVI for GP. Adadelta has the best performance in terms of the marginal likelihood.

In addtion, the resulting values of kernel parameters reinforce **hypothesis 1**. As figure-8 indicates, the inverse lengthscale from the GP model with routine 2* is closer to the corresponding value in sparse GP than routine 1*. ,

To verify **hypothesis 2**, I run three optimization algorithms against each other with routine 2*. The result is shown in figure-9: Adadelta indeed has the best performance in terms of the marginal likelihood, which is consistent with the result from the toy example. Adam converges faster than the other two optimizers in the first 50 iterations, but it has the lowest performance. Rprop has a middle-level performance and outperformed Adam. The experiment suggests that **hypothesis 2** still holds in this data set.

Overall, the experiment with the real data set verifies two observations from the toy example, and in the next section, we will discuss the result in more detail.

5 Discussion and Related Work

In this report, I conduct multiple experiments to explore the effectiveness of different optimization routines and algorithms in solving SVI for GP. Following the empirical evidence, **Adedelta** with **routine 2*** is the most effective approach. Although we only focus on GP regression in the above experiments, it turns out that the structure of the optimization problem also applies to other situations, where probabilistic distribution needs to be approximated. For example, (Hensman et al., 2015) uses Gaussian approximation to fit a given posterior, which involves the optimizing for inducing variable $q(u) = \mathcal{N}(u|m, S)$, inducing points location Z, and the kernel parameters. In their work, they also use Adedelta as the optimizer, but the suggested optimization routine is different from routine 2*, in which they fix the inducing point locations for a few iterations and then optimize all the parameters simultaneously. We try this routine in the regression example, but the result is worse than both routine 1* and routine 2*.

In terms of future work, several aspects could be improved. In the experiment, the SVI class in GPy does not implement the natural gradient. With the natural gradient descent, we expect to see faster convergence; furthermore, natural gradient descent will automatically enforce the positive definite property of covariance matrix of $q(u)$, and the removal of constraint can help with the optimization process. Secondly, this reports analyze problem from an empirical perspective. To validate the hypotheses, mathematical proofs are needed. For example, Adadelta is widely used in Gaussian approximation, but it remained a question of why Adadelta is superior to other optimizers with adaptive learning rate. Understanding the underlying reasons for the superiority of Adadelta can help with designing better optimization algorithms for Gaussian approximation.

References

James Hensman, Nicolo Fusi, and Neil D Lawrence. 2013. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*.

James Hensman, Alexander G Matthews, Maurizio Filippone, and Zoubin Ghahramani. 2015. Mcmc for variationally sparse gaussian processes. In *Advances in Neural Information Processing Systems*, pages 1648–1656.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Martin Riedmiller and Heinrich Braun. 1993. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE international conference on neural networks*, pages 586–591. IEEE.

Michalis Titsias. 2009. Variational learning of inducing variables in sparse gaussian processes. In *Artificial Intelligence and Statistics*, pages 567–574.

I-Cheng Yeh and Tzu-Kuang Hsu. 2018. Building real estate valuation models with comparative approach through case-based reasoning. *Applied Soft Computing*, 65:260–271.

Matthew D Zeiler. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.