Technische Universität Dresden

Department of Artificial Intelligence
Chair of Machine Learning for Computer Vision

Gaussian Processes and Neural Networks

Project Paper by Â§8 of the
Exam Regulations (Computational Modeling and Simulation)

Susu Hu 4888947
Supervisor: Mark Schöne
Dresden, 26th March 2021

Contents

1 Introduction

Neural networks, as parametric models, have gained significant attention due to the ability to learn complex features of large datasets. Despite the training and testing accuracy of neural networks, their poor uncertainty estimation remains problematic in high risk machine learning tasks such as autonomous driving and medical diagnoses ([GAL(2016)Gal], S.). Unlike the point estimator in neural networks, Gaussian Processes (GPs) handle the uncertainty naturally as Bayesian non-parametric models through learning the distribution over functions.

A single fully connected neural network with infinite many neurons has been proven to converge to a Gaussian process ([NEAL(1996)Neal], S.). Neural networks with deep architecture seem to have better performance, leading to the study of deep Gaussian process. The equivalence between Gaussian processes and fully connected neural networks with more than one layer is derived by Matthews et al.([MATTHEWS ET AL.(2018)Matthews, Rowland, Hron, Turner, und Ghahramani], S.). Lee et al. introduced a Neural Network Gaussian Process by constructing a recursive computation of kernel function([LEE ET AL.(2018)Lee, Bahri, Novak, Schoenholz, Pennington, und Sohl-Dickstein], S.). Deep Gaussian processes are also modeled by stacking several GPs layers where the input to the next GP layer is governed by the previous GP ([HENSMAN ET AL.(2014)Hensman, Damianou, und Lawrence, SUN ET AL.(2018)Sun, Zhang, Wang, Zeng, Li, und Grosse], S.).

On the perspective of pattern-recognition, Convolutional Neural Networks (CNNs) demonstrate remarkable performance with their translation equivariant architecture design ([COHEN ET AL.(2016)Cohen, Cohen, und Nl], S.), which motivates the study of convolutional Gaussian processes. Convolution structure was introduced into Gaussian processes kernels to better suit inputs more than one dimension ([VAN DER WILK ET AL.(2017)van der Wilk, Rasmussen, und Hensman], S.). In this project we primarily utilize the convolution Gaussian processes and experiment with MNIST dataset.

2 Background

2.1 Neural Networks

Artificial Neural networks come from the idea of human brain, which can be seen as a collection of connected neurons, loosely resembling the biological neurons. The governing equation of a single layer neural network is given by,

$$\begin{aligned} f(x) &= \psi(\sum_{i=1}^{N} w_i x_i + b) \\ &= \psi(\mathbf{w}^{\mathbf{T}}\mathbf{x} + b) \end{aligned} \tag{1}$$

With $\boldsymbol{x} \in \mathbb{R}^N$ being the input vector, $\boldsymbol{w} \in \mathbb{R}^N$ the weight vector, $b \in \mathbb{R}$ the basis, ψ the activation function, and $f(x) \in \mathbb{R}^N$ the output. Figure 1 shows a single layer neural network.

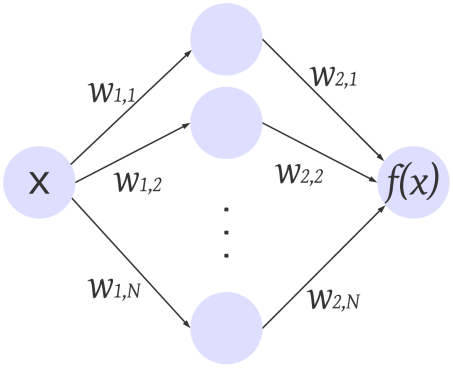

Figure 1: Singel Layer Neural Network

As argued by Hornik ([HORNIK(1993)Hornik], S.) a single layer neural network can approximate continuous functions uniformly under certain constrains. However, deep neural networks by addling more layers instead of expanding neurons in a single layer require less training parameters and demonstrate better testing accuracy on more practical tasks ([MHASKAR ET AL.(2017)Mhaskar, Liao, und Poggio], S.). Figure 2 shows a deep neural network with three hidden layers. The output can be computed recursively using the equation above

$$f^L(\boldsymbol{x}) = \boldsymbol{b}_{L-1} + \boldsymbol{W}_{L-1}\psi_{L-1}(f^{L-1}(\boldsymbol{x})) \tag{2}$$

where the subscript indicates the layer of weight $\boldsymbol{W}$, bias $\boldsymbol{b}$ and activation function ψ, and the superscript indicates the layer of input/output.

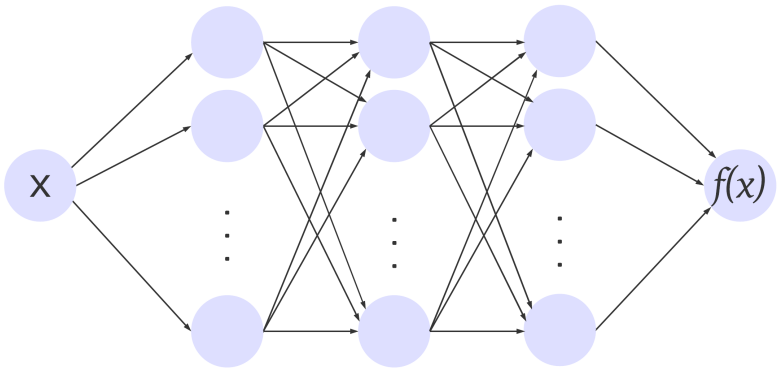

Figure 2: Deep Neural Network With three Hidden Layers

The neural networks mentioned above are all fully-connected, where every neuron is connected with every neuron in the adjacent layers. The structure of neural networks can be designed other ways. One of the most importance architecture is the Convolutional Neural Network. Under the assumption that inputs are images, which is the situation in most cases, a Convolutional Neural Network can be depicted as shown in figure 3, which normally contains convolution layers, pooling layers and fully-connected layers. Convolution layers can capture features despite of the location due to its translational equivariance.

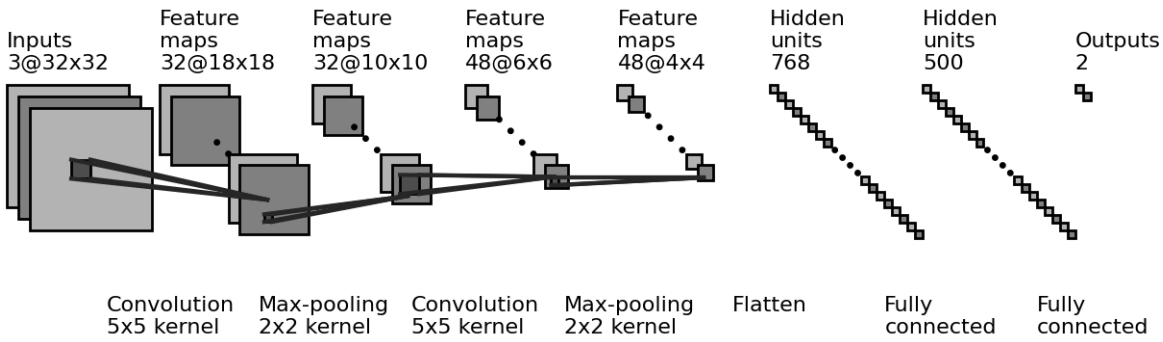

Figure 3: Convolutional Neural Network

Learning in neural networks is to learn a set of weights that minimize the loss function that is defined by the difference of the predicted values and the true labels. A commonly used optimization algorithm is gradient descent, which uses the partial derivative of the loss function w.r.t the neural network parameters.

2.2 Bayesian Neural Networks

From the statistical point of view, the neural network model is hard to interpretate and is prone to overfitting since it generates point estimates with a large number of parameters. Although

some regularization, such as Ridge and Lasso regularization, can be used to counteract overfitting, this kind of neural networks is nonetheless deterministic and lacks the ability to express prediction uncertainty. When the parameters of a neural network are sampled from a probability distribution, and the posterior distribution is estimated using Bayes rule, the neural network goes Bayesian. Here $\boldsymbol{D} = (x_i, y_i)_{i=1}^N$ stands for training data in supervised learning with paired data points and labels. The Bayes rule can be written as below,

$$P(\boldsymbol{\theta}|\boldsymbol{D}) = \frac{P(\boldsymbol{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\boldsymbol{D})} \tag{3}$$

The posterior distribution is simply the product of likelihood and the prior, and normalized by the marginal likelihood which is also known as evidence. The likelihood $P(\boldsymbol{D}|\boldsymbol{\theta})$ represents the likelihood of the data given the parameters. The prior $P(\boldsymbol{\theta})$ encodes the uncertainty of the variables to be estimated. With the Bayes rule, the posterior is calculated to update the uncertainty of variables to be estimated. The marginal likelihood is computed with the sum rule by integrating out the parameters $\boldsymbol{\theta}$, which in practice is often intractable.
When the parameters are leant through training, the prediction y* of new input x* can be computed by this equitation,

$$P(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{D}) = \int P(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{\theta})P(\boldsymbol{\theta}|\boldsymbol{D})d\boldsymbol{\theta} \tag{4}$$

which can be interpreted as the expectation of conditional distribution of the new data w.r.t the posterior, equivalent to computing a weighted average of predictions from an ensemble of neural network.

2.3 Gaussian Processes

Instead of defining distribution over parameters, Gaussian processes as a non-parametric model defines distribution over functions with kernel methods. These sampled functions have joint distribution which is completely specified by its mean and covariance functions. For simplicity, mean is often assumed to be zero.
Formally, Gaussian processes is a collection of random variables, any finite subsets of which also has a multivariate Gaussian distribution([RASMUSSEN(2003)Rasmussen], S.). A Gaussian process with zero-mean, noise variance σ_ϵ^2 and kernel function K can be written as

$$f \sim \mathcal{GP}(\mathbf{0}, \mathbf{K}) \tag{5}$$

$$y_i|x_i \sim \mathcal{N}(f(x_i), \sigma_\epsilon^2) \tag{6}$$

Covariance functions or kernel functions denoted by $\mathbf{K}(x, x')$, where $\mathbf{K} : X \times X \to \mathbb{R}$, basi-

cally measure the similarity between inputs. For example, a popular choice for kernel function is the squared exponential (SE) kernel which is given by,

$$\mathbf{K}_{SE}(x, x') = \sigma^2 exp(-\frac{(x - x')^2}{l^2}) \tag{7}$$

$(x - x')^2$ goes to zero when the two inputs are close to each other, thus $\mathbf{K}(x, x')$ goes to σ^2. And $(x - x')^2$ goes to infinity when the two inputs are far from each other, thus $\mathbf{K}(x, x')$ goes to 0. The lengthscale l controls the wiggle of the function and the output variance σ^2 determines the amplitude of the functions. These two parameters are to be optimized during training process.

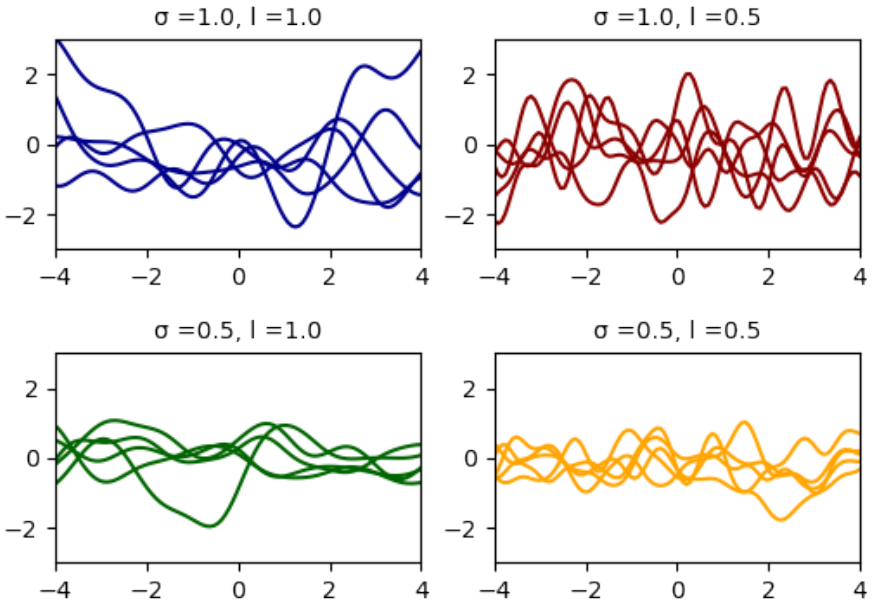

Figure 4: Squared exponential kernel with different parameters

Learning in Gaussian processes is to learn an ensemble of functions and their probability distributions. The better the function fits the training data, the higher probability it has. For a training dataset $\boldsymbol{D} = (x_i, y_i)_{i=1}^N$ and a novel input to $x*$, a Gaussian process forms a joint multivariate Gaussian distribution between joint outputs and inputs with a expanded covariance matrix, which can be written as,

$$\begin{bmatrix} y \\ y* \end{bmatrix} = \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{K}_{x,x} + \sigma_\epsilon^2 \boldsymbol{I} & \mathbf{K}_{x,x^*} \\ \mathbf{K}_{x^*,x} & \mathbf{K}_{x^*,x^*} \end{bmatrix} \right) \tag{8}$$

The distribution of predicted values y* for novel inputs x* is simply conditional distribution,

$$P(y^*|x^*, D) \sim N(y^*|\mathbf{K}_{x^*,x}(\mathbf{K}_{x,x} + \sigma_\epsilon^2 \boldsymbol{I})^{-1} y, \mathbf{K}_{x^*,x^*} - \mathbf{K}_{x^*,x}\mathbf{K}_{x,x}^{-1}\mathbf{K}_{x,x^*}) \tag{9}$$

Matrix inversion $(\mathbf{K}_{x,x}+\sigma_\epsilon^2 \boldsymbol{I})^{-1}$ is required in this computation, making the complexity $\mathcal{O}(N^3)$ given $N \times N$ covariance matrix, which remains an issue in the scalability of Gaussian processes. Besides the computational complexity, another challenge is that the true posterior distribution cannot be computed analytically when the prior and the likelihood are not conjugate.

2.4 Inference Methods

To deal with the challenges mentioned above, Markov chain Monte Carlo (MCMC), a classic stochastic approximation method which samples from a probability distribution, is often used. When the Markov chain converges to an equilibrium state, target distribution is obtained. The difficult part is to determine when the MCMC has converged and how many steps are needed. MCMC is easy to implement but it can take a long time to reach the equilibrium state. It provides unbiased approximation but the unbounded running time limits its application.
An alternative approach to sampling-based methods is variational inference (VI) which transfers inference problem to optimization problem. Given an intractable probability distribution p, variational inference methods try to approximate it with a tractable family of distributions $\mathcal{Q}$ by finding $q \in \mathcal{Q}$ which is most similar to p. The Kullback-Leibler divergence is defined as below to measure the similarity between true posterior distribution and the approximated one,

$$\mathcal{KL}[q(\theta)||p(\theta|x)] = \int q(\theta) log \frac{q(\theta)}{p(\theta|x)} d\theta \tag{10}$$

Optimizing KL divergence directly is not possible since we do not know $p(x)$. An alternative objective function can be obtained by taking the logarithm of marginal likelihood:

$$\begin{aligned} \log p(x) &= \int q(\theta) log \frac{q(\theta)}{p(\theta|x)} d\theta + \int q(\theta) \log \frac{p(x,\theta)}{q(\theta)} d\theta \\ &= \mathcal{KL}[q(\theta)||p(\theta|x)] + \mathcal{L}(\theta) \end{aligned} \tag{11}$$

The first term on the right hand side is the KL divergence and the second term is called evidence lower bound (ELBO). Since the left hand side does not depend on variable θ and KL divergence is non-negative, minimizing KL divergence is equivalent to maximizing this surrogate term ELBO. A major advantage of variational inference is that it is faster and more scalable. Thus we adopt this method in this report.

3 Methodology

3.1 Multiclass Classification Gaussian Processes

As supervised learning, multiclass classification provides paired input and label data, where cardinality of this discrete label set $C > 2$. Learning in this context is to learn the correct mapping from input space $\boldsymbol{X} = (\boldsymbol{x_1}, ..., \boldsymbol{x_N})$ to lable space $\boldsymbol{y} = (y_1, ..., y_N)$, where $y_i \in \{1, 2, ..., C\}$. In MNIST dataset, C is equal to 10.
Compared to binary classification problem, multiclass classification is more complexed because each class requires one latent GP $\boldsymbol{f} = (f^1, f^2, ..., f^C)$,where $f^c \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}'))$. The likelihood of labels are modeled categorically:

$$p(y_i = c|\boldsymbol{x_i}, \boldsymbol{f_i}) = h^k(\boldsymbol{f}(\boldsymbol{x_i})) \tag{12}$$

where $h^k(f)$ is a transfer function that maps from the vector of GP values to probability.
The cost of evaluating the likelihood increases when the class number is large. Despite the cost of inference, imposing GP priors on those latent processes provides a probabilistic framework which enables the modeling of complicated dependencies. Furthermore, there has been extensive studies on improving the scalability of GPs.

3.2 Convolutional Gaussian Processes

Convolutional structured kernel for Gaussian Processes has been proposed([VAN DER WILK ET AL.(2017)van der Wilk, Rasmussen, und Hensman], S.) analogous to convolutional layers in neural networks for learning on images. Learning in convolutional Gaussian processes is based on patch response, the sum of which builds up the image response. The patch response is defined as $g : \mathbb{R}^{w \times h} \to \mathbb{R}$ for a kernel with size $w \times h$. In total, $P = (W - w + 1) \times (H - h + 1)$ patches can be obtained when using stride of 1, where W, H is the image size. The image response can be written as

$$f(x) = \sum_{p=1}^{P} g\left(\mathbf{x}^{[p]}\right) \tag{13}$$

where p represents the patch number of the image $\mathbf{x}$. If the patch response function $g(\cdot)$ is given a GP prior,

$$g \sim \mathcal{GP}(0, k_g(\mathbf{z}, \mathbf{z}')) \tag{14}$$

the correlation of all patches responses can be written as

$$k_f(\{\mathbf{x}, p\}, \{\mathbf{x}', p'\}) = k_g\left(\mathbf{x}^{[p]}, \mathbf{x}'^{[p']}\right) \tag{15}$$

Thus a GP prior can also be induced on $f(\cdot)$:

$$f \sim \mathcal{GP}\left(0, \sum_{p=1}^{P}\sum_{p'=1}^{P'} k_g\left(\mathbf{x}^{[p]}, \mathbf{x}'^{[p']}\right)\right) \tag{16}$$

Convolutional GP is analogous to convolution neural networks in the way that both convolve the kernel function over the image. The difference is that the patch response in convolutional GP is non-linear and non-parametric while that in convnets is linear and followed by non-linear transformation. Increasing the depth of convnets will increase the flexibility by allowing enough non-linear transformation afterwards. However, more filters in convolutional GP will not demonstrate better learning ability since it will only result in summing all the kernel.
This convolution structure is supposed to capture non-local features among images by comparing one patch in image $\mathbf{x}_i$ with another patch in image $\mathbf{x}_j$, thus performs better under the image transformations.

3.3 Robust-max Multiclass Likelihood

One common categorical likelihood transformation is softmax which takes the normalized exponential function

$$p(y_i = c|\boldsymbol{f}(\boldsymbol{x_i})) = \frac{exp(f^c(x_i))}{\sum_{c=1}^{C} exp(f^c(x_i))} \tag{17}$$

Conditioning on x_i is omitted for the sake of simplicity. However the scalability of this method becomes a problem with large data inputs (vgl. [IZMAILOV ET AL.(2018)Izmailov, Novikov, und Kropotov], S.). Moreover, when using softmax function, GPs are coupled, resulting in intractable integration in direct variational inference.
Robust-max has been proposed(vgl. [HERNÃ¡NDEZ-LOBATO ET AL.(2011)HernÃ¡ndez-lobato, HernÃ¡ndez-lobato, und Dupont], S.) to tackle this issue. The likelihood of robust-max is:

$$p(y = k|\boldsymbol{f}) = (1-\epsilon)\prod_{c\neq y}^{C} \Theta(f^k(x) - f^c(x)) + \frac{\epsilon}{C} \tag{18}$$

where ϵ represents the fraction of errors in labeling, and Θ represents the Heaviside function. Robust-max considers only the number of labeling errors, neglecting their distance to the decision boundaries.
This model has better scalability and often yields good classification accuracy as it is robust against outliers. But it sacrifices one important character of softmax, gradual classification, for the all-or-nothing method. We adopt this likelihood function for the sake of computational efficiency.

3.4 Sparse Gaussian Process

The biggest challenge in Gaussian process is its scalability. The computation cost is $\mathcal{O}(N^3)$ since the inverse of matrices needs to be calculated for inference, where N is the number of data. This cubic complexity happens when approximation in Gaussian processes is required at every input data for exact posterior computation. A sparse Gaussian process method has been proposed to reduce the computation cost of approximating full inference([QUINONERO-CANDELA UND CARL EDWARD(2005)Quinonero-Candela und Carl Edward], S.). This methods mainly relies on utilizing a selection of inducing points, which is a non-trivial task.
Here we introduce M extra inducing inputs and inducing variables pairs $\mathbf{Z}$, $\mathbf{u}$. Since inducing inputs $\mathbf{Z}$ are a subset of full input space, the corresponding random variables $\mathbf{u}$ share the same property as latent GP function $\boldsymbol{f}$, so the joint distribution can be written as

$$p(\boldsymbol{f}, \mathbf{u}) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{f}\\ \mathbf{u}\end{bmatrix} | \mathbf{0}, \begin{bmatrix}\mathbf{K}_{nn} & \mathbf{K}_{nm}\\ \mathbf{K}_{mn} & \mathbf{K}_{mm}\end{bmatrix}\right) \tag{19}$$

where $\mathbf{K}_{mm}$ denotes the the evaluation of covariance function between all pairs of inducing points $\mathbf{z}_m$ and $\mathbf{z}_{m'}$, and $\mathbf{K}_{mn}$ the evaluation of covariance function across the inducing points and all data inputs similarly. With this form of multivariate Gaussian distribution, the joint probability can be written as

$$\begin{aligned} p(\boldsymbol{f}, \mathbf{u}) &= p(\boldsymbol{f}|\mathbf{u})p(\mathbf{u}) \\ &= \mathcal{N}(\boldsymbol{f}|\mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{u}, \mathbf{K}_{nn} - \mathbf{Q}_{nn})\mathcal{N}(\mathbf{u}|\mathbf{0}, \mathbf{K}_{mm}) \end{aligned} \tag{20}$$

where $\mathbf{Q}_{nn} = \mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{K}_{nm}^{T}$. Thereby, the joint distribution of observed data and latent variables takes the form

$$p(\boldsymbol{y}, \boldsymbol{f}, \mathbf{u}) = p(\boldsymbol{y}|\boldsymbol{f})p(\boldsymbol{f}|\mathbf{u})p(\mathbf{u}) \tag{21}$$

To obtain the joint distribution of data labels $\boldsymbol{y}$ and laten functions $\boldsymbol{f}$, summation over inducing variables $\mathbf{u}$ is approximated

$$p(\boldsymbol{y}, \boldsymbol{f}) = \sum_{\mathbf{u}} p(\boldsymbol{y}, \boldsymbol{f}|\mathbf{u})p(\mathbf{u}) \tag{22}$$

To achieve computational efficiency, further assumption is made that the function value at all input data points $\boldsymbol{f}$ and the observed data labels $\boldsymbol{y}$ are independent when conditioned on inducing variables $\mathbf{u}$([QUINONERO-CANDELA UND CARL EDWARD(2005)Quinonero-Candela und

Carl Edward], S.), thus, the joint probability can be re-written as

$$p(\boldsymbol{y}, \boldsymbol{f}) = \sum_{\mathbf{u}} p(\boldsymbol{y}|\mathbf{u})p(\boldsymbol{f}|\mathbf{u})p(\mathbf{u}) \tag{23}$$

Approximations differ with difference choices of $p(\boldsymbol{y}|\mathbf{u})$ and $p(\boldsymbol{f}|\mathbf{u})$, making $\mathbf{u}$ an important variable to be optimized during training.
Undert the assumption above, this sparse Gaussian process method reduces the complexity of inference from $\mathcal{O}(N^3)$ to $\mathcal{O}(NM^2)$ where M is the number of inducing points.

3.5 Inference and Optimization

When the likelihood and the prior are conjugate, the true posterior distribution can be calculated analytically, but even with closed-form solutions, the actual computing is of high cost due to the need for covariance matrix inversion and determinant. Normally the priors and likelihoods are tractable in Bayesian models, but fact that it is difficult to normalize the posterior (the marginal likelihood) results in the intractability. And in Gaussian processes, the inversion and determinant of the covariance matrix are expensive to compute to obtain the density of GP prior, making it intractable. So approximation is required mainly for two reasons 1) non-conjugacy of prior and likelihood, 2) high computation cost of matrix inversion and determinant.
As mentioned in 2.4, there are two types of methods for approximation, one is sampling-based MCMC and the other is variational inference method. The sampling-based MCMC method is unbiased, but it converges slow and also requires high computational cost, here we adopt the variational inference method.
To obtain the posterior distribution $p(\boldsymbol{f}|\mathbf{X}, \boldsymbol{y})$, we choose a tractable distribution $q(\boldsymbol{f}) \in \mathcal{Q}$ and minimize the KL divergence:

$$\mathcal{KL}[q(\boldsymbol{f})||p(\boldsymbol{f}|\mathbf{X}, \boldsymbol{y})] = \sum_{\boldsymbol{f}} q(\boldsymbol{f}) \log \frac{q(\boldsymbol{f})}{p(\boldsymbol{f}|\mathbf{X}, \boldsymbol{y})} \tag{24}$$

Since KL-divergnece is non-negative and

$$\log p(\boldsymbol{y}|\mathbf{X}) = \mathcal{L}(q(\boldsymbol{f})) + \mathcal{KL}[q(\boldsymbol{f})||p(\boldsymbol{f}|\mathbf{X}, \boldsymbol{y})] \tag{25}$$

our final object is to maximize the surrogate ELBO

$$\mathcal{L}(q(\boldsymbol{f})) = \sum_{\boldsymbol{f}} q(\boldsymbol{f}) \log \frac{p(\mathbf{X}, \boldsymbol{y}, \boldsymbol{f})}{q(\boldsymbol{f})} \tag{26}$$

Here we choose Gaussian distribution as $\mathcal{Q}$ since it is tractable and large enough to contain

the target distribution.

There are plenty of algorithms for optimization problem which are based on stochastic gradient descend. Here we optimized with L-BFGS-B and Adam.

L-BFGS-B, developed based on limited-memory algorithm([ZHU ET AL.(1997)Zhu, Byrd, Lu, und Nocedal], S.), is a quasi-Newton method which approximates the Hessian matrix with low-rank update, thus more computationally efficient than Newton methods. Since it is a second derivative optimization method and the loss is guaranteed to decrease every time step, it might converge faster than first derivative optimization method theoretically. Adam on the other hand is a first derivate based optimization method which takes momentum into account when updating the gradient descent([KINGMA UND BA(2017)Kingma und Ba], S.). It is a popular optimizer in neural networks training because of it is computational efficient and often yields good results.

4 Results

4.1 Gaussian Process Models

We implement Gaussian process on MNIST dataset, which is a handwritten 0-9 digits dataset with 10 classes. The image size is 28×28.
The first model we use is Gaussian process without convolutional structure. The chosen kernel is squared exponential kernel. It was tested on full MMIST dataset with 60000 training samples and 10000 testing samples. We randomly initialize 50 inducing points. The model reached approximately 92.1% accuracy for both L-BFGS-B and Adam optimizers after 20 iterations. With L-BFGS-B optimizer, the model reached 94.2% accuracy on training and 93.3% on testing after 100 iterations, while with Adam 96.6% and 95.4% respectively.

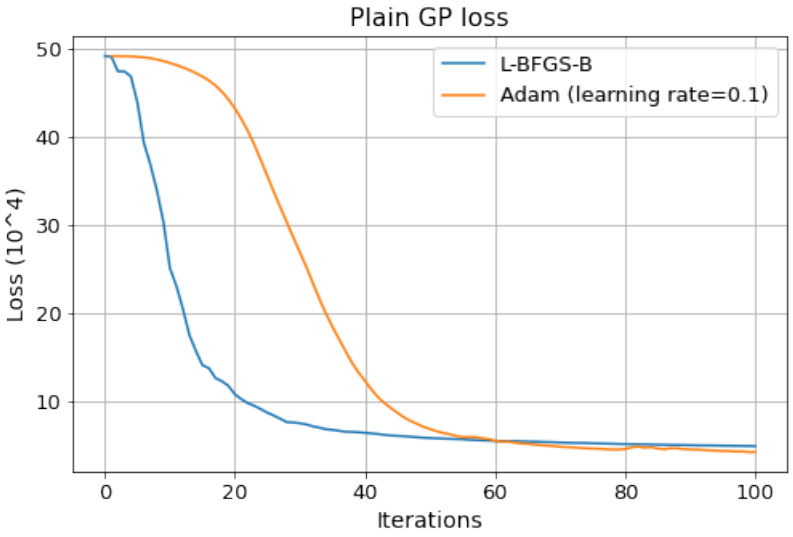

Figure 5: Plain GP loss change

Figure 5 shows the loss change with iterations for both optimization methods. L-BFGS-B did converge faster than Adam but Adam performed slightly better after 60 iterations despite of some oscillation. And per iteration, L-BFGS-B took 2.10 seconds while Adam took only 0.02 second.

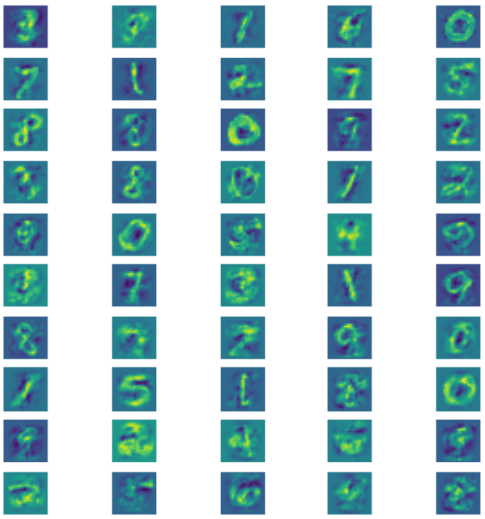

Figure 6: 50 optimized inducing patches of plain GP model

Figure 6 shows the 50 optimized inducing points, which contains all 10 classes.
The second model is convolutional Gaussian process. Due to the high computation cost and our limited hardware capability, we only managed to test it on 5000 training samples and 500 testing samples. We chose squared exponential with size 9×9 as base kernel. This kernel size is relatively large because smaller number of patches is desired to reduce covariance computation for patches while it might reduce the ability to capture features. We randomly initialized 100 inducing patches and after 100 iterations, this model reached 81.5% training accuracy and 78.6% testing accuracy with L-BFGS-B optimizer and 96.4% and 94.6% respectively with Adam. It took around 21.74 seconds per iteration for L-BFGS-B and 0.27 second for Adam.

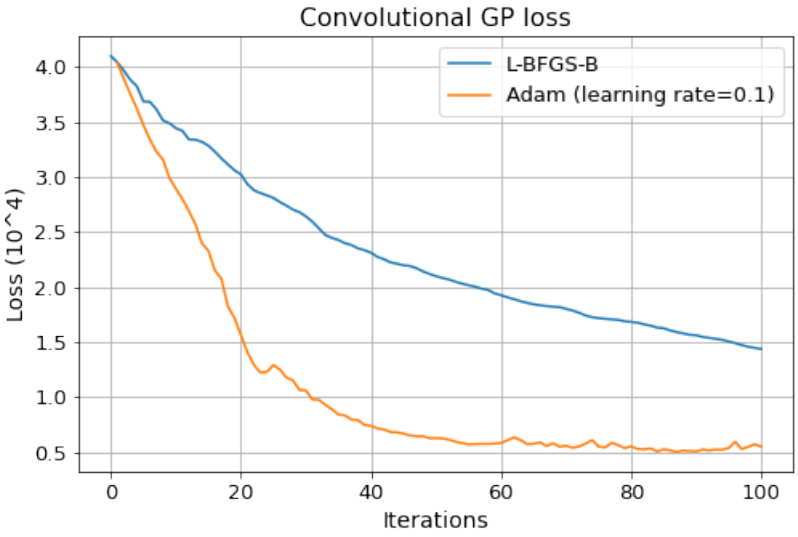

Figure 7: Convolutional GP loss change

As figure 7 shows, with the convolutional kernel, L-BFGS-B converges much slower than Adam despite that L-BFGS-B is guaranteed to decrease loss ever time step and Adam experiences some oscillation. The reason might be that L-BFGS-B performs line-search for approximating the optimal step-size, which is not a free parameter. Under this more complicated loss computing process, more iterations are required for L-BFGS-B to reach a better results.

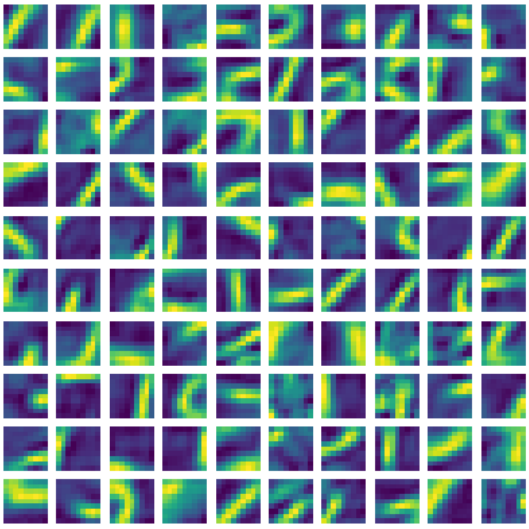

Figure 8: 100 optimized inducing patches of convolutional GP model

Figure 8 shows the 100 optimized inducing patches of convolutional GP model.
Table 1 shows the summary of the GP models mentioned above. In plain GP, one inducing point is one full image, so 50 is assumed to be a good number. In convolutional GP, one inducing point is one patch, and we have $(28 - 9 + 1) \times (28 - 9 + 1) = 400$ patches per image with our chosen kernel size and stride, so we increased the number of inducing points to 100.

Model	**Data**	**Image size**	**Kernel size**	**Optimizer**	**Iterations**	**#inducing points**	**Training acc.**	**Testing acc.**	**Speed (second/iter)**
Plain GP	Full MNIST	[28, 28]	-	L-BFGS	100	50	94.2%	93.3%	2.10
Plain GP	Full MNIST	[28, 28]	-	Adam	100	50	96.6%	95.4%	0.02
Conv GP	Partial MNIST	[28, 28]	[9, 9]	L-BFGS	100	100	81.5%	78.6%	21.74
Conv GP	Partial MNIST	[28, 28]	[9, 9]	Adam	100	100	96.4%	94.6%	0.27

Table 1: GP model comparison

In general, MNIST data set is considered large for convolutional Gaussian process, making the scalability issue of Gaussian process more obvious.

4.2 Neural Network Models

Next, we experimented with two neural network models. The first model we use contains one hidden fully connected layer with 512 neurons. The model is optimized with Adam learning rate 0.1 and dropout rate 0.2. After 5 epochs, training accuracy reached 98.6% and testing accuracy 97.8%, and each epoch took only 4 seconds on average.

Layer(type)	**Output Shape**	**Param#**	**Activation**
flatten(Flatten)	(None, 784)	0	
dense(Dense)	(None, 512)	401920	Relu
dropout(Dropout)	(None, 512)	0	
dense_1(Dense)	(None, 10)	5130	Softmax
Total params: 407050 Trainable params: 407050 Non-trainable params: 0			

Table 2: Fully-connected neural network model summary

To compare with convolutional Gaussian process, we also experimented with a convolutional neural network, which contains 10 filters of kernel size 9×9. Again, this relatively large kernel size is selected for easy comparison with convolutional Gaussian process we present above. After 5 epochs, convolutional model reached 98.9% training accuracy and 98.4% testing accuracy, slightly higher than the fully connected neural model. But to reach this accuracy, it requires enough filter numbers to capture possible features, so each epoch took longer, 18 seconds on average.

Layer(type)	**Output Shape**	**Param#**	**Activation**
Conv2d(Conv2D)	(None, 28, 28, 10)	820	Relu
max_pooling2d	(None, 14, 14, 10)	0	
flatten_1(Flatten)	(None, 1960)	0	
dense_2(Dense)	(None, 10)	19610	Softmax
Total params: 20430 Trainable params: 20430 Non-trainable params: 0			

Table 3: Convolutional neural network model summary

In conclusion, for this simple dataset convolution structure is not necessary in either Gaussian process or neural network, but a proper kernel size and enough number of kernel can increase the accuracy by capturing enough translation invariant features. Either with convolutional structure or not, neural network all performed better on the perspective of both accuracy and computation efficiency.

5 Discussion

In this research project, we have studied mathematical theories of Gaussian processes and implemented it on MINST dataset and compared the results with that of neural networks. The neural network models we chose are not carefully designed or fine-tuned, but they demonstrated better performance than Gaussian processes nevertheless. With the benchmark neural network model, the accuracy on MNIST has reached 99.84% ([BYERLY ET AL.(2020)Byerly, Kalganova, und Dear], S.). There is always a trade-off between reducing computation cost and capturing the full data features during the sparse Gaussian process. Even though the second derivative optimization method L-BFGS-B is guaranteed to reduce loss every time step, it does not necessarily perform better than first derivative optimization method with complicated loss function due to its self-adjusted step-size.

6 Outlook

The scalability of Gaussian process remains the biggest challenge despite of its uncertainty quantification ability and full mathematical interpretability. Researches have been working on this for years but the results are not satisfying. The big data era has imposed an even more difficult condition on Gaussian process. Due to this exact issue of Gaussian process, experiments were only conducted on simple MNIST dataset. If the computation issue is to be efficiently tackled in the future, further experiments of Gaussian processes on more practical data shall be studied.

7 Acknowledgement

We thank Mark Schöne for his invaluable comments and helpful suggestions.

References

[Byerly et al.(2020)Byerly, Kalganova, und Dear] Byerly et al. 2020 Byerly, Adam ; Kalganova, Tatiana ; Dear, Ian: A branching and merging convolutional network with homogeneous filter capsules. In: *arXiv preprint arXiv:2001.09136* (2020)

[Cohen et al.(2016)Cohen, Cohen, und Nl] Cohen et al. 2016 Cohen, Taco S. ; Cohen, T S. ; Nl, Uva: Group Equivariant Convolutional Networks. (2016), S. 10

[Gal(2016)Gal] Gal 2016 Gal, Yarin: Uncertainty in Deep Learning. (2016), S. 174

[Hensman et al.(2014)Hensman, Damianou, und Lawrence] Hensman et al. 2014 Hensman, James ; Damianou, Andreas ; Lawrence, Neil: Deep Gaussian Processes for Large Datasets. (2014), S. 1

[HernÃ¡ndez-lobato et al.(2011)HernÃ¡ndez-lobato, HernÃ¡ndez-lobato, und Dupont] HernÃ¡ndez-lobato et al. 2011 HernÃ¡ndez-lobato, Daniel ; HernÃ¡ndez-lobato, Jose M. ; Dupont, Pierre: Robust Multi-Class Gaussian Process Classification. (2011), S. 9

[Hornik(1993)Hornik] Hornik 1993 Hornik, K.: Some new results on neural network approximation. 6 (1993), Nr. 8, 1069–1072. http://dx.doi.org/10.1016/S0893-6080(09)80018-X. – DOI 10.1016/S0893–6080(09)80018–X. – ISSN 0893–6080

[Izmailov et al.(2018)Izmailov, Novikov, und Kropotov] Izmailov et al. 2018 Izmailov, Pavel ; Novikov, Alexander ; Kropotov, Dmitry: Scalable Gaussian Processes with Billions of Inducing Inputs via Tensor Train Decomposition. (2018). http://arxiv.org/abs/1710.07324

[Kingma und Ba(2017)Kingma und Ba] Kingma und Ba 2017 Kingma, Diederik P. ; Ba, Jimmy: Adam: A Method for Stochastic Optimization. (2017). http://arxiv.org/abs/1412.6980

[Lee et al.(2018)Lee, Bahri, Novak, Schoenholz, Pennington, und Sohl-Dickstein] Lee et al. 2018 Lee, Jaehoon ; Bahri, Yasaman ; Novak, Roman ; Schoenholz, Samuel S. ; Pennington, Jeffrey ; Sohl-Dickstein, Jascha: Deep Neural Networks as Gaussian Processes. (2018). http://arxiv.org/abs/1711.00165

[Matthews et al.(2018)Matthews, Rowland, Hron, Turner, und Ghahramani] Matthews et al. 2018 Matthews, Alexander G. de G. ; Rowland, Mark ; Hron, Jiri ;

TURNER, Richard E. ; GHAHRAMANI, Zoubin: Gaussian Process Behaviour in Wide Deep Neural Networks. (2018). http://arxiv.org/abs/1804.11271

[MHASKAR ET AL.(2017)Mhaskar, Liao, und Poggio] MHASKAR ET AL. 2017 MHASKAR, Hrushikesh ; LIAO, Qianli ; POGGIO, Tomaso: When and Why Are Deep Networks Better Than Shallow Ones? 31 (2017), Nr. 1. https://ojs.aaai.org/index.php/AAAI/article/view/10913. – ISSN 2374–3468. – Number: 1

[NEAL(1996)Neal] NEAL 1996 NEAL, P Radford M. Radford M: *Bayesian Learning for Neural Networks*. Springer New York, 1996 http://public.ebookcentral.proquest.com/choice/publicfullrecord.aspx?p=3074890. – ISBN 978–1–4612–0745–0. – OCLC: 958525906

[QUINONERO-CANDELA UND CARL EDWARD(2005)Quinonero-Candela und Carl Edward] QUINONERO-CANDELA UND CARL EDWARD 2005 QUINONERO-CANDELA, Joaquin ; CARL EDWARD, Rasmussen: A unifying view of sparse approximate Gaussian process regression. 6 (2005), S. 1939–1959

[RASMUSSEN(2003)Rasmussen] RASMUSSEN 2003 RASMUSSEN, Carl E.: Gaussian processes in machine learning. In: *Summer school on machine learning* Springer, 2003, S. 63–71

[SUN ET AL.(2018)Sun, Zhang, Wang, Zeng, Li, und Grosse] SUN ET AL. 2018 SUN, Shengyang ; ZHANG, Guodong ; WANG, Chaoqi ; ZENG, Wenyuan ; LI, Jiaman ; GROSSE, Roger: Differentiable Compositional Kernel Learning for Gaussian Processes. (2018). http://arxiv.org/abs/1806.04326

[VAN DER WILK ET AL.(2017)van der Wilk, Rasmussen, und Hensman] VAN DER WILK ET AL. 2017 WILK, Mark van d. ; RASMUSSEN, Carl E. ; HENSMAN, James: Convolutional Gaussian Processes. 30 (2017), 2849–2858. https://proceedings.neurips.cc/paper/2017/hash/1c54985e4f95b7819ca0357c0cb9a09f-Abstract.html

[ZHU ET AL.(1997)Zhu, Byrd, Lu, und Nocedal] ZHU ET AL. 1997 ZHU, Ciyou ; BYRD, Richard H. ; LU, Peihuang ; NOCEDAL, Jorge: Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. 23 (1997), Nr. 4, 550–560. http://dx.doi.org/10.1145/279232.279236. – DOI 10.1145/279232.279236. – ISSN 0098–3500