Learning Input-conditional Invariances via the Marginal Likelihood

Rui Li[1]

MSc Machine Learning

Supervisors:
Marc Peter Deisenroth, So Takao, Mark van der Wilk

Submission date: September 2021

[1]**Disclaimer:** This report is submitted as part requirement for the MSc Degree in Machine Learning at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

Good generalization ability is fundamental for any machine learning algorithm to be useful. In supervised learning, we can improve the model's generalization ability by forcing the model to be invariant to transformations that are known to not influence the output, i.e., force the model to give the same predictions for input under the aforementioned transformations. Commonly, this is done through data augmentation, where an augmented dataset is created by applying hand-crafted transformations to the inputs. Due to the tediousness of manually determining the maximal amount of transformation that the model should be invariant to, recent works propose to learn that maximal amount of transformation instead. However, these models are still limited as they can only learn the maximal amount of transformation which is uniform to the dataset, while different subsets of dataset usually have different properties and therefore have different levels of tolerance to the aforementioned transformation. In this thesis, we aim to learn the maximal amount of transformation that is conditioned on the inputs such that the proposed model is expected to generalize better compared with the model with uniform invariance when given a limited amount of data. To do so, we propose the idea of hidden invariance class which allows the model to learn different amount of transformation for each data point with a limited amount of parameters. Experimental results show that when different subsets of data have distinct levels of tolerance to a certain type of transformation, the proposed models are capable of learning meaningful input-conditional invariance and therefore generalize better than the model with uniform invariance.

Acknowledgements

Firstly, I would like to thank my supervisors Marc Deisenroth, So Takao and Mark van der Wilk for their constructive remarks and invaluable guidance. Without their crystal clear thoughts and well-oriented questions, this project would never have gone very far. I am profoundly grateful to Marc and So for giving me the opportunity to explore this project in the first place, being incredibly generous with their time, and being supportive and patient when I slowly studied the background knowledge at the beginning and stuck with bugs in the middle. Also, thank So and Marc for their detailed comments on the draft, which significantly improves the quality of this thesis.

Secondly, I would like to thank James Wilson and Alexander Terenin for their insightful comments during weekly meetings. Thank Eiki Shimizu for the useful discussions throughout the project. Thank Pola Schwöbel and Martin Jørgensen for helping track down tricky bugs.

At last, I would like to thank my parents for their constant love and support which keep me motivated and confident. Thank Yi Song for being an awesome friend and giving me encouragement and support whenever I need them. Thank Fei La for being a lovely furry friend and comforting me. Thank Furudate Haruichi, Scarlett Johansson, Mayday, and Taylor Swift for their wonderful works in manga, movies and music, which provide the happy distractions to rest my mind outside of my study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Good generalization ability is fundamental for any machine learning algorithm to be useful. In supervised learning, we would like the trained model to make correct predictions in as much of the input space as possible when given a limited amount of training examples, i.e., we would like the model to be able to generalize what it learnt into unseen data. To generalize well, the model must be in some extent invariant[1] to transformations that are known to not influence the output. Otherwise, the model will not be able to recognize inputs under slight modifications and fail miserably in even the simplest task.

To encourage the model's invariance to transformations that are known to not influence the output, we can constrain the model to make the same predictions for data points under such transformations. Take image classification for example, to improve the model's invariance to rotation and translation, we can rotate and translate given images and force the model to make the same predictions for these transformed images. In modern machine learning pipelines, this is often done through data augmentation, where we first create an enlarged dataset by adding slightly modified copies of already existing data or newly creating synthetic data from existing data, then train the model on the enlarged dataset.

Despite the success of data augmentation in incorporating invariance into the model and therefore improving the model's generalization ability [1, 41, 16], meaningful augmentation heavily relies on expert knowledge, trial and error, and cross-validation, which makes the process of generating augmentation dataset as tedious as manually crafting features. Also, from a Bayesian perspective, data augmentation is undesirable because by incorporating pseudo data which are not true observations (augmentation dataset), the posterior may become overconfident. We can also hard-encode invariance into the model through delicate parameterization, where a typical example would be Convolutional Neural Network (CNN) [23]. Recent works also propose to incorporate more sophisticated invariance such as scale and rotation invariance [6, 26, 40, 33] into CNN.

A common drawback about traditional ways of incorporating invariance into the model is that it is unclear which constraints are suitable for the given problem. In addition, it is also unclear what is

[1] We will give a detailed introduction of invariance in Section 3.1. For now the reader can just consider "being invariant to a transformation t" as "giving the same prediction for inputs under transformation t", or just consider "being invariant" as "being robust".

the maximal amount of constraints we should impose on the model. To improve the aforementioned drawback, recently the following question has been proposed: Can we let the model *learn* the suitable invariance from data? Although the general question is hard to tackle, a special case has been successfully solved: given a type of transformation we would like the model to be invariant to, it is possible to learn *the maximal amount of transformation* the model should be invariant to from training data alone. [35] achieves this through training with marginal likelihood in the Gaussian process (GP) setting and [2] achieves this through training with regularized loss in the neural network setting.

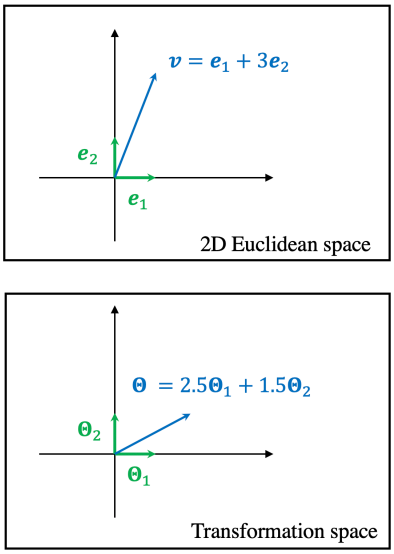
Given a transformation, it is certainly desirable to learn the extent to which the model should be invariant to from training data alone, instead of pre-assigning the maximal amount of transformation by trial and error. However, existing methods [35, 2] are still limited as they can only learn the maximal amount of transformation that is *uniform* to all data points, while different subsets of dataset usually correspond to different amount of transformation the model should be invariant to. Take digit classification for example and suppose we would like the model to be invariant to rotation. As shown in Fig. 1.1, even within the same class of digit, different data point still corresponds to different amount of maximal rotation angle. By learning the extent of the transformation the model should be invariant to that is *conditioned on the input*, we expect to have a model that will generalize even better when given a limited amount of data.

Figure 1.1: Even for digits in the same class in the MNIST dataset, the maximal amount of rotation we can perform on these digits before they become indistinguishable from digit 9 are different.

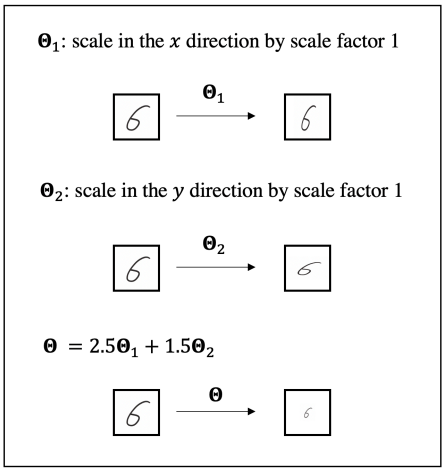
In this thesis, we aim to design invariant models which are capable of learning the maximal amount of transformation that is *conditioned on the input*. As invariance is essentially a function's property and GPs are well suited to incorporate high-level property into the function, here we adopt the framework proposed in [35] which is based on GPs. Specifically, the learnt invariance should reflect the different data points' different levels of tolerance to the given transformation. For example, if successful, in digit classification the designed model should be invariant to large amount of rotation to digits like 0, and invariant to small amount of rotation to digits like 6 and 9. To learn input-conditional invariance, we propose the idea of hidden invariance class[2], which allows the model to assign suitable amount of transformation for each data point based on its property. To briefly illustrate how to learn input-conditional invariance, consider the following metaphor: to obtain any vector in a vector space, we can use the linear combination of a set of basis vectors. As different amount of transformations can be parameterized as different real vectors in the same vector space[3], we can obtain any amount of transformation through a set of "basis transformations" (we use the quotation mark to emphasize that rigorously they are not basis vectors in the vector space). Therefore, input-conditional invariance can be learnt by learning the coefficients in the

[2]Hidden invariance class will be introduced in Section 4.1.1.
[3]We will introduce how to do this in Section 3.3.

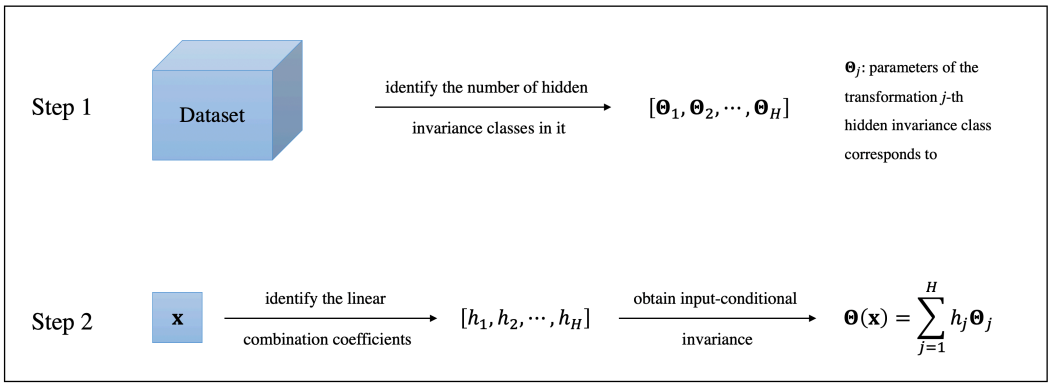
linear combination of "basis transformations" for each input. We illustrate how to learn input-conditional invariance in Fig 1.2.



(a) Similarly to how basis vectors in the vector space can form any vector, we can use a set of "basis transformations" to form any transformation.



(b) An illustration of how new transformation is generated when we are interested in scale transformation.



(c) General procedure of learning input-conditional invariance. We first identify the number of hidden invariance classes in the given dataset, then we learn the transformations along with the coefficients in the linear combination for each data point.

Figure 1.2: Illustration of hidden invariance class and how to learn input-conditional invariance.

The main contribution of this thesis is extending [35] and proposing invariant GPs which are capable of learning input-conditional invariance. By allowing the model to be invariant to different amount of transformations based on input data's properties, we hypothesize that it will generalize better when given a limited amount of data. A second, more minor contribution is to compare two variational bounds proposed

in [35] and [30] experimentally and show which one is more suitable for learning input-conditional invariance. Experimental results show that when different subsets of data have distinct levels of tolerance to a certain type of transformation, the proposed models are capable of learning meaningful input-conditional invariance and generalize better than the invariant model with uniform invariance. As a spoiler, we give the illustration of learnt amount of transformation in our proposed model in Fig. 1.3.

(a) Training Examples.

(b) Samples describing the learned invariance for training examples (digits in the blue squares are training examples). The model becomes insensitive to different amount of rotation based on different properties of training examples.

Figure 1.3: Learnt input-conditional invariance for synthetic dataset where some digits are manually rotated by a large amount of angle. The model successfully learns different invariance for different digits.

This thesis is structured as follows. Chapter 2 gives the necessary background knowledge and a short review of common methods of incorporating invariance into the model. Chapter 3 gives the introduction to invariant Gaussian process, which is the model this thesis is built on. Chapter 4 introduces the idea of hidden invariance class in detail and presents the models proposed to learn input-conditional invariance. Chapter 5 discusses the setup of the experiment. Chapter 6 gives the experiment results. Chapter 7 concludes this thesis and discusses the future work.

Chapter 2

Background

This chapter gives a brief introduction to the background knowledge required for this thesis. We begin with a short introduction to general Bayesian inference in Section 2.1. Then, we introduce Gaussian process, which is the model used to construct the invariant model, in Section 2.2. At last, we review common methods to encourage invariance in Section 2.3.

2.1 Bayesian Inference

2.1.1 Bayesian Modeling and Inference

In machine learning, we aim to infer the underlying patterns in the given data. To do so, we express our assumptions about the underlying patterns through a set of possible models and then learn the parameters of the models from the data.

In Bayesian machine learning, we treat the parameters of the models as random variables and infer their probability distribution. Specifically, by denoting the unknown parameters as $\mathbf{w}$, the set of training inputs as $\mathbf{X}$ and the collection of corresponding training outputs as $\mathbf{Y}$, the process of inferring the probability distribution of unknown parameters $\mathbf{w}$ can be summarized as:

1. Specify a prior probability distribution $p(\mathbf{w})$ for the unknown parameters $\mathbf{w}$, which expresses our belief of all possible values of $\mathbf{w}$.

2. Specify the likelihood $p\left(\mathbf{Y} \mid \mathbf{w}, \mathbf{X}\right)$, which represents our assumption about the process of how the observed data are generated.

3. Use Bayes' theorem $p\left(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}\right) = \frac{p(\mathbf{Y}|\mathbf{w},\mathbf{X})p(\mathbf{w})}{\int p(\mathbf{Y}|\mathbf{w},\mathbf{X})p(\mathbf{w})\mathrm{d}\mathbf{w}}$ to infer the posterior probability distribution $p\left(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}\right)$ for the unknown parameters, which represents the updated belief of all possible values of $\mathbf{w}$ when we observe the data.

2.1.2 Predictions and Decisions

Once we obtain the posterior probability distribution $p(\mathbf{w} \mid \mathbf{X}, \mathbf{Y})$, for any unseen input $\mathbf{x}^*$, we can infer the probability distribution of its corresponding output $\mathbf{y}^*$ through averaging over all possible parameter settings:

$$p(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{w} \mid \mathbf{X}, \mathbf{Y})\, p(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{w})\, \mathrm{d}\mathbf{w}. \tag{2.1}$$

Then, the mean of $p(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{X}, \mathbf{Y})$ is usually used as the prediction and the variance of $p(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{X}, \mathbf{Y})$ is used as uncertainty qualification of our prediction.

In practice, we might also want to compare different models to see which one is the most suitable model for the given data. In Bayesian machine learning, this can be done through comparing the marginal likelihood $p(\mathbf{Y} \mid \mathbf{X})$, which is the expected likelihood under the prior distribution of the parameters:

$$p(\mathbf{Y} \mid \mathbf{X}) = \int p(\mathbf{Y} \mid \mathbf{w}, \mathbf{X})\, p(\mathbf{w}) \mathrm{d}\mathbf{w} \tag{2.2}$$

2.2 Gaussian Process

2.2.1 Definition

Gaussian process is the gold standard for many real-world modeling problems as it has high flexibility and well-calibrated uncertainty qualification. Gaussian process provides a way to put probability distribution on random *functions* $f : \mathcal{X} \to \mathbb{R}$, such that for any finite set of locations $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N] \subseteq \mathcal{X}$, the function values $\mathbf{f} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)]$ evaluated at $\mathbf{X}$ are jointly Gaussian distributed.

The formal definition for Gaussian process is

Definition 2.2.1 (Gaussian Process) *[29] A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.*

Similarly to mean and covariance are all it is needed to define a Gaussian distribution, a Gaussian process is fully specified by defining its mean function and covariance function. Denoting the mean function as $m(\mathbf{x})$ and the covariance function as $k(\mathbf{x}, \mathbf{x}')$, we have

$$\begin{aligned} &\mathbb{E}[f(\mathbf{x})] \stackrel{\mathrm{d}}{=} m(\mathbf{x}), \\ &\mathbb{E}\left[(f(\mathbf{x}) - m(\mathbf{x}))\left(f(\mathbf{x}') - m(\mathbf{x}')\right)\right] \stackrel{\mathrm{d}}{=} k(\mathbf{x}, \mathbf{x}'), \end{aligned} \tag{2.3}$$

and the Gaussian process $f(\cdot)$ can be written as

$$f(\mathbf{x}) \sim \mathcal{GP}\left(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\right). \tag{2.4}$$

Then, the prior on the function values $\mathbf{f} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)]$ evaluated at the training set $\mathbf{X}$ is

$$p(\mathbf{f} \mid \mathbf{X}) = \mathcal{N}\left(m(\mathbf{X}), \mathbf{K}_{\mathbf{ff}}\right), \tag{2.5}$$

where $\mathbf{K_{ff}}$ is the shorthand for $k(\mathbf{X}, \mathbf{X})$, where $[k(\mathbf{X}, \mathbf{X})]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.

2.2.2 Exact Inference

As in practice $m(\mathbf{x})$ is usually set to be 0 for symmetry reasons, in the following sections we will use this convention.

Given the training inputs $\mathbf{X}$ and test inputs $\mathbf{X}^*$, the joint prior distribution of the training outputs $\mathbf{f}$ and the test outputs $\mathbf{f}^*$ are

$$p\left(\mathbf{f}, \mathbf{f}^* \mid \mathbf{X}, \mathbf{X}^*\right) = \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \mathbf{K_{ff}}, & \mathbf{K_{ff^*}} \\ \mathbf{K_{f^*f}}, & \mathbf{K_{f^*f^*}} \end{bmatrix}\right). \tag{2.6}$$

When assuming $y_i = f(\mathbf{x}_i) + \varepsilon_i$ where $\varepsilon_i \sim \mathcal{N}\left(0, \sigma^2\right)$, we have

$$p\left(\mathbf{y}, \mathbf{f}^* \mid \mathbf{X}, \mathbf{X}^*\right) = \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \mathbf{K_{ff}} + \sigma^2\mathbf{I}, & \mathbf{K_{ff^*}} \\ \mathbf{K_{f^*f}}, & \mathbf{K_{f^*f^*}} \end{bmatrix}\right). \tag{2.7}$$

Then, using the following well-known property of the Gaussian distribution

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_x \\ \boldsymbol{\mu}_y \end{bmatrix}, \begin{bmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C}^\top & \mathbf{B} \end{bmatrix}\right), \tag{2.8}$$

$$\mathbf{x} \mid \mathbf{y} \sim \mathcal{N}\left(\boldsymbol{\mu}_x + \mathbf{C}\mathbf{B}^{-1}\left(\mathbf{y} - \boldsymbol{\mu}_y\right), \mathbf{A} - \mathbf{C}\mathbf{B}^{-1}\mathbf{C}^\top\right), \tag{2.9}$$

we have

$$\begin{aligned} p\left(\mathbf{f}^* \mid \mathbf{X}, \mathbf{X}^*, \mathbf{y}\right) = \mathcal{N}(&\mathbf{K_{f^*f}}\left(\mathbf{K_{ff}} + \sigma_n^2\mathbf{I}\right)^{-1}\mathbf{y}, \\ &\mathbf{K_{f^*f^*}} - \mathbf{K_{f^*f}}\left(\mathbf{K_{ff}} + \sigma_n^2\mathbf{I}\right)^{-1}\mathbf{K_{ff^*}}\Big), \end{aligned} \tag{2.10}$$

which is the posterior predictive distribution.

2.2.3 Approximate Inference

Note that when doing inference with Eq. (2.10), we need to invert $\mathbf{K_{ff}} + \sigma_n^2\mathbf{I} \in \mathbb{R}^{N \times N}$ (N is the number of the total training data), which results in $O\left(N^3\right)$ time complexity and $O\left(N^2\right)$ storage complexity. When N is large, it would be computational expensive, if not possible, to use GPs. In this section we introduce sparse variational approximation which makes GPs applicable to a large amount of data efficiently.

Sparse Approximation via Inducing Points

The basic idea behind sparse approximation is to summarize the information contained in the original dataset into a set of inducing points $\mathbf{Z} = \{\mathbf{z}_i\}_{i=1}^M, (M \ll N)$, such that the matrix needs to be inverted becomes $\mathbb{R}^{M \times M}$ [32]. One common way of doing this, as first introduced in [34], is collecting the

function values $f(\mathbf{Z})$ in the variable $\mathbf{u} \in \mathbb{R}^M$, and specify free mean $\mathbf{m}$ and variance $\mathbf{S}$ as variational parameters:

$$q(\mathbf{u}) = \mathcal{N}(\mathbf{u}; \mathbf{m}, \mathbf{S}). \tag{2.11}$$

Then, we can specify the distribution for all other points $f^{\setminus \mathbf{Z}}(\cdot)$ using the prior conditioned on $\mathbf{u}$, which results in a family of GP posteriors on $f(\cdot)$:

$$\begin{aligned} q(f(\cdot)) &= p(f^{\setminus \mathbf{Z}}(\cdot) \mid \mathbf{u}) q(\mathbf{u}) \\ &= \mathcal{GP}\left(\mathbf{k}_{\mathbf{u}}^\top(\cdot)\mathbf{K}_{\mathbf{uu}}^{-1}\mathbf{m}, k(\cdot,\cdot) - \mathbf{k}_{\mathbf{u}}^\top(\cdot)\mathbf{K}_{\mathbf{uu}}^{-1}\left[\mathbf{K}_{\mathbf{uu}} - \mathbf{S}\right]\mathbf{K}_{\mathbf{uu}}^{-1}\mathbf{k}_{\mathbf{u}}(\cdot)\right), \end{aligned} \tag{2.12}$$

where $\mathbf{k}_{\mathbf{u}}(\cdot) = \left[k\left(\mathbf{z}_m, \cdot\right)\right]_{m=1}^M$ is the covariance between the inducing outputs and the rest of the process, and $\left[\mathbf{K}_{\mathbf{uu}}\right]_{mm'} = k\left(\mathbf{z}_m, \mathbf{z}_{m'}\right)$

Stochastic Variational Inference

Armed with the family of approximate posteriors defined in Eq. (2.12), we can then minimize the KL divergence between the approximate posterior $q(f(\cdot))$ and the true posterior $p(f(\cdot) \mid \mathbf{y}, \mathbf{X})$. Although technically the KL divergence can not deal with densities over functions, [8] shows that this particular KL divergence can be computed from the KL between a finite marginal distribution on function values:

$$\mathbb{KL}[q(f(\cdot)) \| p(f(\cdot) \mid \mathbf{y}, \mathbf{X})] = \mathbb{KL}[q(f(\mathbf{X}), \mathbf{u}) \| p(f(\mathbf{X}), \mathbf{u} \mid \mathbf{y}, \mathbf{X})]. \tag{2.13}$$

To simplify our notations, we denote $f(\mathbf{X})$ as $\mathbf{f}$ and omit the dependence on $\mathbf{X}$. Then, using Bayes' theorem we have

$$p(\mathbf{f}, \mathbf{u} \mid \mathbf{y}) = \frac{p(\mathbf{f} \mid \mathbf{u}) p(\mathbf{u}) p(\mathbf{y} \mid \mathbf{f})}{p(\mathbf{y})}, \tag{2.14}$$

where

$$p(\mathbf{f} \mid \mathbf{u}) = \mathcal{N}\left(\mathbf{K}_{\mathbf{fu}}\mathbf{K}_{\mathbf{uu}}^{-1}\mathbf{u}, \mathbf{K}_{\mathbf{ff}} - \mathbf{K}_{\mathbf{fu}}\mathbf{K}_{\mathbf{uu}}^{-1}\mathbf{K}_{\mathbf{uf}}\right), \tag{2.15}$$

and

$$p(\mathbf{u}) = \mathcal{N}\left(\mathbf{0}, \mathbf{K}_{\mathbf{uu}}\right). \tag{2.16}$$

From the previous section, we have

$$q(\mathbf{f}, \mathbf{u}) = p(\mathbf{f} \mid \mathbf{u}) q(\mathbf{u}), \tag{2.17}$$

then,

$$\begin{aligned}
\mathbb{KL}[q(\mathbf{f},\mathbf{u})\|p(\mathbf{f},\mathbf{u}\mid\mathbf{y})] &= \int q(\mathbf{f},\mathbf{u})\log\frac{q(\mathbf{f},\mathbf{u})}{p(\mathbf{f},\mathbf{u}\mid\mathbf{y})}\mathbf{dudf} \\
&= \log p(\mathbf{y}) + \int q(\mathbf{f},\mathbf{u})\log\frac{q(\mathbf{f},\mathbf{u})}{p(\mathbf{f},\mathbf{u},\mathbf{y})}\mathrm{d}\mathbf{u}\mathrm{d}\mathbf{f} \\
&= \log p(\mathbf{y}) + \int q(\mathbf{f},\mathbf{u})\log\frac{p(\mathbf{f}\mid\mathbf{u})q(\mathbf{u})}{p(\mathbf{y}\mid\mathbf{f})p(\mathbf{f}\mid\mathbf{u})p(\mathbf{u})}\mathrm{d}\mathbf{u}\mathrm{d}\mathbf{f} \\
&= \log p(\mathbf{y}) + \int p(\mathbf{f}\mid\mathbf{u})q(\mathbf{u})\log\frac{q(\mathbf{u})}{p(\mathbf{u})}\mathrm{d}\mathbf{u}\mathrm{d}\mathbf{f} - \int p(\mathbf{f}\mid\mathbf{u})q(\mathbf{u})\log p(\mathbf{y}\mid\mathbf{f})\mathrm{d}\mathbf{u}\mathrm{d}\mathbf{f} \\
&= \log p(\mathbf{y}) + \int q(\mathbf{u})\log\frac{q(\mathbf{u})}{p(\mathbf{u})}\mathrm{d}\mathbf{u} - \int p(\mathbf{f}\mid\mathbf{u})q(\mathbf{u})\log p(\mathbf{y}\mid\mathbf{f})\mathrm{d}\mathbf{u}\mathrm{d}\mathbf{f} \\
&= \log p(\mathbf{y}) + \mathbb{KL}[q(\mathbf{u})\|p(\mathbf{u})] - \int q(\mathbf{f})\log p(\mathbf{y}\mid\mathbf{f})\mathrm{d}\mathbf{f},
\end{aligned} \tag{2.18}$$

where

$$\begin{aligned}
q(\mathbf{f}) &\overset{\mathrm{d}}{=} \int p(\mathbf{f}\mid\mathbf{u})q(\mathbf{u})\mathrm{d}\mathbf{u} \\
&= \mathcal{N}\left(\mathbf{f}\mid\mathbf{K_{fu}}\mathbf{K_{uu}^{-1}}\mathbf{m},\mathbf{K_{ff}}+\mathbf{K_{fu}}\mathbf{K_{uu}^{-1}}\left(\mathbf{S}-\mathbf{K_{uu}}\right)\mathbf{K_{uu}^{-1}}\mathbf{K_{uf}}\right).
\end{aligned} \tag{2.19}$$

As a result, we have

$$\min\mathbb{KL}[q(\mathbf{f},\mathbf{u})\|p(\mathbf{f},\mathbf{u}\mid\mathbf{y})] \iff \max\sum_{n=1}^{N}\mathbb{E}_{q(f(\mathbf{x}_n))}\left[\log p\left(y_n\mid f\left(\mathbf{x}_n\right)\right)\right] - \mathrm{KL}[q(\mathbf{u})\|p(\mathbf{u})], \tag{2.20}$$

which implies we can minimize the KL divergence and find the approximate posterior by maximizing the following lower bound of the marginal likelihood:

$$\mathcal{L} = \sum_{n=1}^{N}\mathbb{E}_{q(f(\mathbf{x}_n))}\left[\log p\left(y_n\mid f\left(\mathbf{x}_n\right)\right)\right] - \mathbb{KL}[q(\mathbf{u})\|p(\mathbf{u})]. \tag{2.21}$$

Note that the above bound factorizes over data points and therefore widely used stochastic optimization methods like Adam [19] can be used.

To summarize, given M inducing points $\mathbf{Z} = \{\mathbf{z}_m\}_{m=1}^{M}$ and the family of variational posterior defined in Eq. (2.12), the approximate posterior is $q(f) = \mathcal{GP}(\mu(\cdot),\nu(\cdot,\cdot))$ with

$$\mu(\cdot) = \mathbf{k}_{\mathbf{u}}^{\top}(\cdot)\mathbf{K_{uu}^{-1}}\mathbf{m}, \quad \nu(\cdot,\cdot) = k(\cdot,\cdot) - \mathbf{k}_{\mathbf{u}}^{\top}(\cdot)\mathbf{K_{uu}^{-1}}\left[\mathbf{K_{uu}}-\mathbf{S}\right]\mathbf{K_{uu}^{-1}}\mathbf{k}_{\mathbf{u}}(\cdot), \tag{2.22}$$

where $\mathbf{k}_{\mathbf{u}}(\cdot) = \left[k\left(\mathbf{z}_m,\cdot\right)\right]_{m=1}^{M}$ and $\left[\mathbf{K_{uu}}\right]_{mm'} = k\left(\mathbf{z}_m,\mathbf{z}_{m'}\right)$.

The ELBO is given by

$$\log p(\mathbf{y}) \geq \mathcal{L} = \sum_{n=1}^{N}\mathbb{E}_{q(f(\mathbf{x}_n))}\left[\log p\left(y_n\mid f\left(\mathbf{x}_n\right)\right] - \mathrm{KL}[q(\mathbf{u})\|p(\mathbf{u})\right]. \tag{2.23}$$

2.2.4 Inter-Domain Inducing Variables

In this section, we introduce inter-domain methods which will be used later to simplify certain expressions that come up in invariant GPs.

Recall that the core idea of inducing points is to summarize the information in the original data into a small amount of data. Inter-domain methods [22] extends the summary ability of inducing variables by allowing inducing variables belonging to different domains, such that we can better capture the information in the original dataset.

In standard sparse approximations, inducing variables $\mathbf{u}$ are set to be the function values evaluated at inducing points $\mathbf{Z}$, i.e., $\mathbf{u} = f(\mathbf{Z})$. In inter-domain method, we use the following transformation instead:

$$u(\mathbf{z}) \stackrel{\mathrm{d}}{=} \int f(\mathbf{x}) h(\mathbf{x}, \mathbf{z}) \mathrm{d}\mathbf{x}, \tag{2.24}$$

where $h(\mathbf{x}, \mathbf{z})$ is the pre-defined function which encodes prior knowledge into inducing variables.

Then, the transformed-domain instance of the mean is

$$m(\mathbf{z}) = \mathbb{E}[u(\mathbf{z})] = \int \mathbb{E}[f(\mathbf{x})] h(\mathbf{x}, \mathbf{z}) \mathrm{d}\mathbf{x} = \int m(\mathbf{x}) h(\mathbf{x}, \mathbf{z}) \mathrm{d}\mathbf{x}, \tag{2.25}$$

and the inter-domain and transformed-domain instances of the covariance function are:

$$\begin{aligned} k(\mathbf{x}, \mathbf{z}') &= \mathbb{E}\left[f(\mathbf{x}) u(\mathbf{z}')\right] = \mathbb{E}\left[f(\mathbf{x}) \int f(\mathbf{x}') h(\mathbf{x}', \mathbf{z}') \mathrm{d}\mathbf{x}'\right] = \int k(\mathbf{x}, \mathbf{x}') h(\mathbf{x}', \mathbf{z}') \mathrm{d}\mathbf{x}', \\ k(\mathbf{z}, \mathbf{z}') &= \mathbb{E}\left[u(\mathbf{z}) u(\mathbf{z}')\right] = \mathbb{E}\left[\int f(\mathbf{x}) h(\mathbf{x}, \mathbf{z}) \mathrm{d}\mathbf{x} \int f(\mathbf{x}') h(\mathbf{x}', \mathbf{z}') \mathrm{d}\mathbf{x}'\right] \\ &= \iint k(\mathbf{x}, \mathbf{x}') h(\mathbf{x}, \mathbf{z}) h(\mathbf{x}', \mathbf{z}') \mathrm{d}\mathbf{x} \mathrm{d}\mathbf{x}'. \end{aligned} \tag{2.26}$$

In practice inter-domain inducing methods can be simply applied by defining $h(\mathbf{x}, \mathbf{z})$ and then using Eq. (2.25) and Eq. (2.26) as drop-in replacement.

2.2.5 Efficient Sampling from Posterior

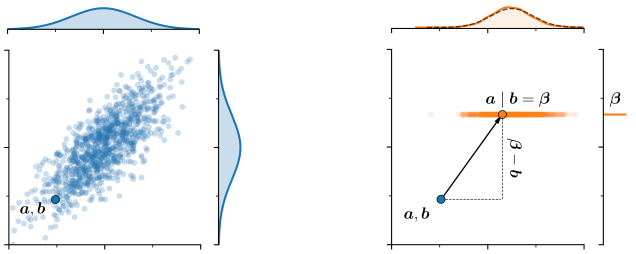
For the last part of GP's background, we introduce a way to sample from GP posteriors efficiently, which will be used later to evaluate the ELBO of invariant GPs efficiently.

As GPs are usually used as a building block in large applications and the problem is usually intractable which results in Monte Carlo estimation, it is desirable to sample from GP posteriors more efficiently. [37] proposes a way to efficiently sample from GP posteriors in linear time complexity. Their discovery is based on Matheron's rule presented as below:

Theorem 1 (Matheron's Rule) *Let* $\mathbf{a}$ *and* $\mathbf{b}$ *be jointly Gaussian random variables. Then the random variable* $\mathbf{a}$ *conditional on* $\mathbf{b} = \boldsymbol{\beta}$ *is equal in distribution to*

$$(\mathbf{a} \mid \mathbf{b} = \boldsymbol{\beta}) \stackrel{\mathrm{d}}{=} \mathbf{a} + \operatorname{Cov}(\mathbf{a}, \mathbf{b}) \operatorname{Cov}(\mathbf{b}, \mathbf{b})^{-1} (\boldsymbol{\beta} - \mathbf{b})$$

In words, Matheron's rule provides a way to decompose Gaussian conditional distribution into a prior term and a data-dependent update term. Therefore, we can sample from $p\left(\mathbf{a} \mid \mathbf{b}=\boldsymbol{\beta}\right)$ by first sampling from the joint prior $p(\mathbf{a}, \mathbf{b})$, then update the joint draw $\mathbf{a}, \mathbf{b}$ to account for the residual $\boldsymbol{\beta}-\mathbf{b}$. Fig. 2.1 illustrates the procedure for the simple case of bivariate normal random variables.



(a) Joint sample $\mathbf{a}, \mathbf{b}$. (b) Transform $\mathbf{a}, \mathbf{b}$ into $\mathbf{a} \mid \mathbf{b}$.

Figure 2.1: Sampling from $p\left(\mathbf{a} \mid \mathbf{b}=\boldsymbol{\beta}\right)$ through Matheron's rule (picture is taken from [38]).

As the GP posteriors are exactly Gaussian conditional distribution, we can use Matheron's rule to decompose the GP posteriors as follow:

$$\underbrace{(f \mid \mathbf{u})(\cdot)}_{\text{posterior}} \overset{\mathrm{d}}{=} \underbrace{f(\cdot)}_{\text{prior}} + \underbrace{k(\cdot, \mathbf{Z}) K_{\mathbf{uu}}^{-1}\left(\mathbf{u}-\mathbf{f}_m\right)}_{\text{data-dependent update}} \tag{2.27}$$

where $\mathbf{f}_m = f(\mathbf{Z})$. Then, sampling from posterior can be done by

$$\underbrace{f^{(s)}(\cdot) \mid \mathbf{X}, \mathbf{y}}_{\text{sample from posterior}} = \underbrace{f^{(s)}(\cdot)}_{\text{sample from prior}} + \underbrace{k(\cdot, \mathbf{X}) K^{-1}\left(\boldsymbol{y}-f^{(s)}(\mathbf{X})\right)}_{\text{data-dependent update}} \tag{2.28}$$

and the procedure is illustrated in Fig. 2.2

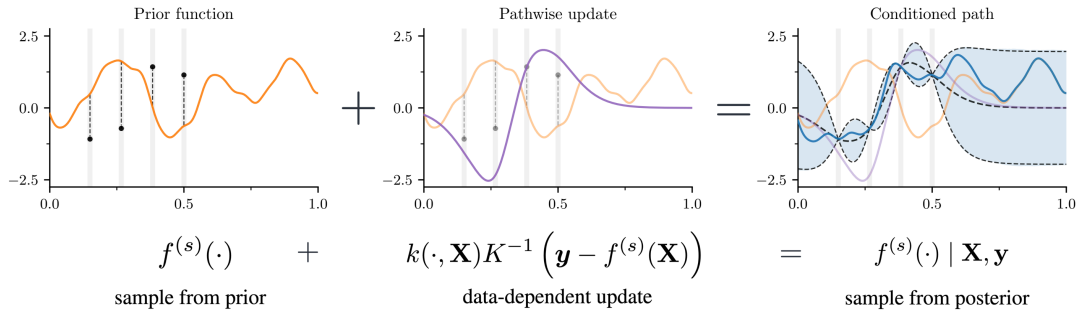


Figure 2.2: Sampling from GP posteriors through Matheron's rule (modified from [9]).

2.3 Common Ways of Incorporating Invariance

As incorporating invariance can let the model become more data-efficient and generalize better, it has been investigated extensively in the literature. In this section, we provide a brief review of existing methods by grouping them into three rough categories.

2.3.1 Data Augmentation

Data augmentation first extends the training set by adding transformed data points that are known to not influence the output, then train the model on the augmented dataset [3, 27]. Common techniques include augmenting with small rotations, scaling, thickening and deformations [4]. In deep learning tasks like ImageNet [10], it is standard to apply flips, crops, and color alterations [21, 17].

Due to the tedious labour in manually specifying the optimal amount of applied transformations, recent works usually adopt unsupervised model to generate augmentation data. For example, [1, 15] use the GAN framework to match augmentations to the data distribution, while [16] align images in the same class in a pairwise fashion to learn image deformations appear within different classes. Reinforcement learning is also used to augment dataset by finding an optimal augmentation policy [7, 24].

2.3.2 Model Constraint

We can also *hard-coded* invariance through adding constraints to the model. As a typical example, CNN [23] obtains translation invariance by applying the same filters across different image locations and pooling afterwards. Recent works also hard-encode rotation invariance [6, 26, 39] and scale invariance [40, 33] into CNN.

Invariances have also been incorporated into kernel methods. [20, 12, 13] investigate kernel with translation invariance, and [36] designs convolutional kernel which allow Gaussian process to possess similar property as CNN.

2.3.3 Regularization

Apart from adding constraints on the model's parameters, invariance can also be incorporated through adding extra penalties to the objective function. For example, [5, 14] encourage local perturbations invariance in SVM through regularization, while [2] bias training towards solutions that incorporate invariances in neural network.

Chapter 3

Learning Invariances via Invariant Gaussian Process

In this chapter, we start with the introduction of invariance in Section 3.1 and proceed to the construction of invariant functions in Section 3.2, and parameterization of transformations in Section 3.3. Then, we discuss the objective function of learning invariance in Section 3.4. At last, we introduce how to construct invariant GPs in Section 3.5 and how to do inference in invariant GPs in Section 3.5.

3.1 What is Invariance

Before giving the definition of invariance, we first introduce the notion of orbit, which will be used frequently when discussing invariance.

Definition 3.1.1 (Orbit) *Given a set of transformations $\mathcal{T}$ where $t : \mathcal{X} \to \mathcal{X}$, $\forall t \in \mathcal{T}$, a point $\mathbf{x}$'s orbit $\mathcal{O}_{\mathbf{x}}$ is the set of points obtained by applying transformations in $\mathcal{T}$ to $\mathbf{x}$, i.e., we have*

$$\mathcal{O}_{\mathbf{x}} = \{t(\mathbf{x}) \mid t \in \mathcal{T}\}$$

Fig. 3.1 gives an example of a point's orbit when $\mathcal{T} = \{t_+(\cdot), t_-(\cdot)\}$ where $t_+(\cdot)$ will rotate a given image by 90° and $t_-(\cdot)$ will rotate a given image by -90° (minus means counterclockwise).

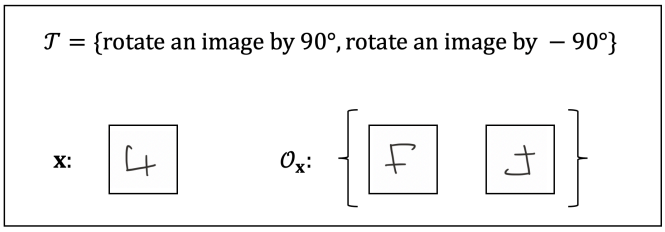


Figure 3.1: Orbit of a point when $\mathcal{T}$ contains two transformations which will rotate an image by $90°$ and $-90°$ respectively (minus means counterclockwise).

Now we give the definition of invariance, here we follow the convention in [20, 12]:

Definition 3.1.2 (Strict Invariance) *[20, 12] For any transformation $t : \mathcal{X} \to \mathcal{X}$ in a set $\mathcal{T}$, if a function $f(\cdot)$ satisfies*

$$f(\mathbf{x}) = f\left(t(\mathbf{x})\right), \quad \forall \mathbf{x} \in \mathcal{X}, \quad \forall t \in \mathcal{T}, \tag{3.1}$$

we say function $f(\cdot)$ is strictly invariant to transformations in $\mathcal{T}$.

In words, for a function $f(\cdot)$ to be strictly invariant to a set of transformations $\mathcal{T}$, $f(\cdot)$ needs to give the same output for the input $\mathbf{x}$ and its orbit $\mathcal{O}_{\mathbf{x}}$. For example, suppose $\mathcal{T} = \{t_+(\cdot), t_-(\cdot)\}$, $\mathcal{X}$ is the set of handwritten digits, and $f(\cdot)$ is the handwritten digits classifier. Then, classifier $f(\cdot)$ is said to be strictly invariant to $\mathcal{T}$ if it assigns the same label for any $\mathbf{x} \in \mathcal{X}$ and its orbit $\mathcal{O}_{\mathbf{x}}$ (the images where $\mathbf{x}$ is rotated by $-90°$ and $90°$). Fig. 3.2 gives an illustration of a function which is strictly invariant to $\mathcal{T}$, i.e., strictly invariant to $90°$ and $-90°$ rotation.

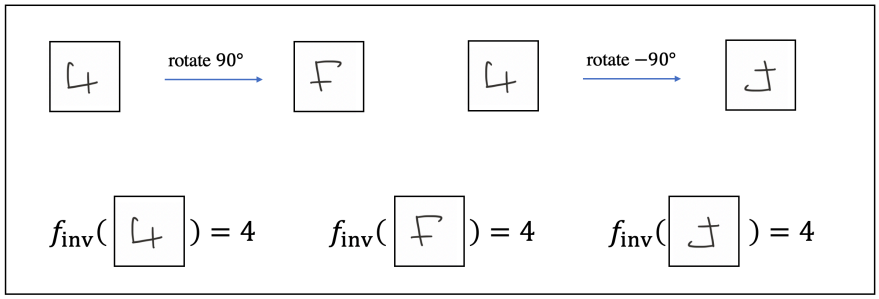


Figure 3.2: When a function is strictly invariant to $90°$ and $-90°$ rotation, it will give the same output for the original image and the images that are rotated by $90°$ and $-90°$.

A point worth emphasizing separately is that invariance is always associated with a set of transformations. We can consider invariance as a measurement of how a function's output will be influenced by a set of transformations. When a function's output isn't influenced by a set of transformations and gives the same output for any input and its corresponding orbit, we say this function is strictly invariant to those transformations.

For many machine learning tasks, strict invariance could potentially jeopardize the model's performance as it requires the model to predict the *same* value for data points under transformations. When the transformations doesn't capture the property of the data extremely accurately (which is often the case), by constraining the model to predict the *same* value for data points under *inaccurate* transformations, the model's performance will inevitably be jeopardized. As shown in Fig. 3.2, by constraining the function to be strict invariant to -90° rotation, it will classify the digit which looks more like 5 into 4. Therefore, in this thesis we focus on learning approximate invariance as defined in [35]:

Definition 3.1.3 (Approximate Invariance) *[35] For transformation t in a set $\mathcal{T}$, given the transformation probability $p(t)$, we say $f(\cdot)$ is approximately invariant to $\mathcal{T}$ if*

$$P\left([f(\mathbf{x}) - f(t(\mathbf{x}))]^2 > L\right) < \epsilon, \quad \forall \mathbf{x} \in \mathcal{X}, \quad t \sim p(t).$$

In words, strict invariance requires $f(\cdot)$ to have exactly the same value for $t(\mathbf{x}), \forall x \in \mathcal{X}, \forall t \in \mathcal{T}$, while approximate invariance requires $f(\cdot)$ to have similar values for $t(\mathbf{x}), \forall x \in \mathcal{X}, \forall t \in \mathcal{T}$. For the remainder of this thesis, we will refer approximate invariance as simply "invariance".

3.2 How to Construct Invariant Functions

For a function $f(\cdot)$ to be invariant to a set of transformations $\mathcal{T}$, it needs to give similar values for $\mathbf{x}$ and its orbit $\mathcal{O}_{\mathbf{x}}$. Therefore, we can construct an invariant function by first choosing an arbitrary function $g(\cdot)$, then averaging $g(\cdot)$ over the input's orbit. Specifically, we can construct an invariant function $f(\cdot)$ as

$$f(\mathbf{x}) \stackrel{\mathrm{d}}{=} \frac{1}{|\mathcal{O}_{\mathbf{x}}|} \sum_{\mathbf{x}_a \in \mathcal{O}_{\mathbf{x}}} g(\mathbf{x}_a). \tag{3.2}$$

Fig. 3.3 illustrates how to construct an invariant function.

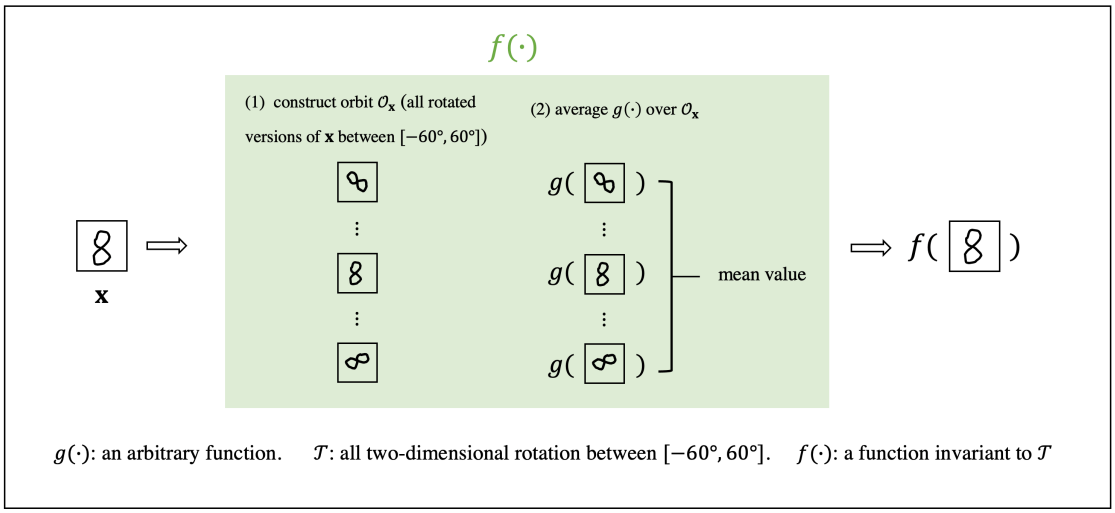


Figure 3.3: To construct an invariant function $f(\cdot)$ for transformations in set $\mathcal{T}$, we first choose an arbitrary function $g(\cdot)$, then for any input $\mathbf{x}$, we obtain $f(\mathbf{x})$ by averaging $g(\cdot)$ over its orbit $\mathcal{O}_{\mathbf{x}}$.

By constructing invariant function in this way, the extent to which the function is invariant to a given transformation is fully determined by the orbit we are averaging over. For example, by averaging over all rotated versions of input image between $[-10°, 30°]$, we obtain a function which is invariant to rotation between $[-10°, 30°]$. Then, conceptually we can learn the amount of transformation the model should be invariant to by adjusting the orbit we are averaging over. Unfortunately, it is not clear how to parameterize orbit. Therefore, here we introduce the following alternative definition of invariance, which is easier to work with:

Definition 3.2.1 (Approximate Invariance) *[35] For transformation $t \in \mathcal{T}$, given the transformation probability $p(t)$, denote the orbit distribution as $p(\mathbf{x}_a \mid \mathbf{x})$ where points $\mathbf{x}_a$ are obtained by applying the transformations $t \sim p(t)$ to $\mathbf{x}$, we say $f(\cdot)$ is invariant to $\mathcal{T}$ if*

$$P\left(\left[f(\mathbf{x}) - f\left(\mathbf{x}_a\right)\right]^2 > L\right) < \epsilon, \quad \forall \mathbf{x} \in \mathcal{X}, \quad \mathbf{x}_a \sim p\left(\mathbf{x}_a \mid \mathbf{x}\right).$$

When working with Definition 3.2.1 instead, we can still construct the invariant function $f(\cdot)$ as we did before. To obtain the function value $f(\mathbf{x})$, we average over input $\mathbf{x}$'s orbit $\mathcal{O}_{\mathbf{x}}$:

$$f(\mathbf{x}) \stackrel{\mathrm{d}}{=} \int_{\mathcal{O}_{\mathbf{x}}} g\left(\mathbf{x}_a\right) p\left(\mathbf{x}_a \mid \mathbf{x}\right) \mathrm{d}\mathbf{x}_a. \tag{3.3}$$

However, now the item controls the extent of transformation $f(\cdot)$ should be invariant to becomes orbit distribution $p\left(\mathbf{x}_a \mid \mathbf{x}\right)$, which is much easier to parameterize compared with orbit $\mathcal{O}_{\mathbf{x}}$.

3.3 How to Parameterize Orbit Distribution

Recall orbit distribution $p(\mathbf{x}_a \mid \mathbf{x})$ is obtained by applying the transformations $t \sim p(t)$ to $\mathbf{x}$. Then, once we find a way to parameterize transformation distribution $p(t)$, we find a way to parameterize orbit distribution as they are essentially referring to the same thing.

To parameterize transformation distribution $p(t)$, we first introduce how to represent transformation $t(\cdot)$. A useful conclusion from group theory is that for any type of transformation, there exists a set of generators which can be used to generate any transformation of that type. Consider the following example: once we know how to rotate an image by 1°, we can rotate an image by any amount of degree through rotating that image 1° many times. Here the operation of "rotate an image by 1°" can be considered as the generator of rotation transformation. Specifically, if we constrain ourselves to transformations in the matrix group, then by denoting the generators as $\mathbf{G}_k$ (for complex transformations there are usually more than one generator) and supposing there are K generators in total, any transformation $\mathbf{T}$ can be formed by

$$\mathbf{T} = \exp\left(\sum_{k=1}^{K} \theta_k \mathbf{G}_k\right) \tag{3.4}$$

where $\exp$ is the matrix exponential function: $\exp(\mathbf{A}) = \sum_{n=0}^{\infty} \frac{1}{n!}\mathbf{A}^n$.

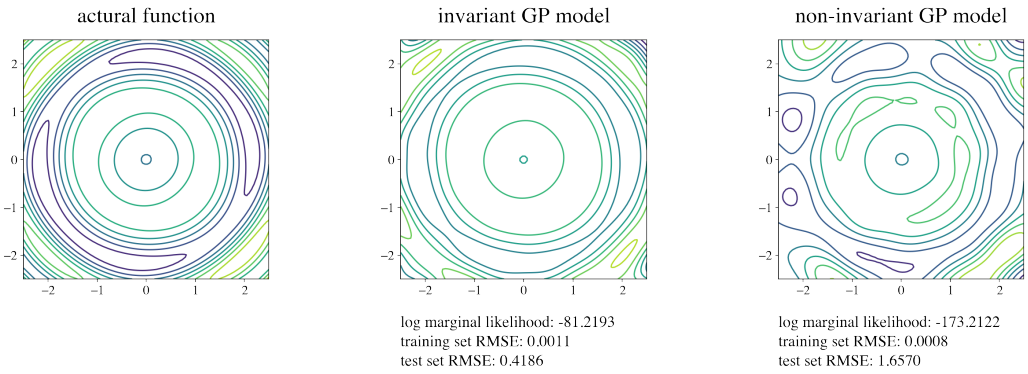
Take 2D affine transformation for example, here the transformation can be expressed in matrix form and it has six generators $\mathbf{G}_1, \ldots, \mathbf{G}_6$ which correspond to translation in x direction, translation in y direction, rotation, scaling in x direction, scaling in y direction, and shearing. These six generators can be written as

$$\begin{aligned}
\mathbf{G}_1 &= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_3 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\
\mathbf{G}_4 &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_5 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_6 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.
\end{aligned} \tag{3.5}$$

By combining each generator through the exponential map in Eq. (3.4), we can obtain any form of affine transformation. Then, we can apply that transformation through transforming the coordinate grid points of the input[1]. For general transformations, we can find the corresponding generators in group theory.

As the generators are fixed when given the type of the transformation, we can use the combination coefficients $[\theta_1, \ldots, \theta_K]$ in the the exponential map (Eq. (3.4)) to represent different amount of transformation. Then, the transformation distribution $p(t)$ in Definition 3.1.3 can be expressed equivalently through the distribution over $[\theta_1, \ldots, \theta_K]$. As a result, we can parameterize the orbit distribution through distribution over $[\theta_1, \ldots, \theta_K]$. Specifically, because we are interested in learning the maximal amount of transformation the model should be invariant to, we can model orbit distribution as uniform distribution and consider θ_k as the maximal amount of the transformation generator $\mathbf{G}_k$ corresponds to. Then, to

[1] In this thesis we use the spatial transformation network proposed in [18].



(a) Ground truth function.

(b) Learnt functions in the invariant model.

(c) Learnt functions in the non-invariant model.

Figure 3.4: While the non-invariant model has better fits on the training data, the invariant model generalizes better. The marginal likelihood correctly identifies the invariant model as the one that generalizes better.

learn input-conditional invariance, we only need to make orbit parameters condition on the input.

3.4 Objective Function to Learn Invariance

Now that we know how to parameterize the extent of transformation the model should be invariant to, we introduce the objective function should be used to learn that extent.

As invariance requires the model $f(\cdot)$ to assign similar values for inputs under certain transformation, it imposes a strong constraint to the model and will often limit how good the model can *fit* the training data. Therefore, widely used losses like negative log-likelihood are not suitable for learning invariances as they only encourage how well the model fits the data.

Marginal likelihood introduced in Section 2.1 is a loss widely used in Bayesian machine learning. Recall it is found by integrating the likelihood $p(\mathbf{y} \mid f)$ over the prior on $f(\cdot)$:

$$p(\mathbf{y} \mid \boldsymbol{\theta}) = \int p(\mathbf{y} \mid f) p(f \mid \boldsymbol{\theta}) \mathrm{d}f. \tag{3.6}$$

Apart from effectively capturing the model complexity as well as the data fit [28, 25, 29], marginal likelihood is also closely related to bounds on the generalisation error [31, 11], which makes marginal likelihood an ideal choice for learning invariance. The example given in Fig. 3.4 shows that the marginal likelihood does correctly identify the invariant model as the one that generalizes better.

In practice marginal likelihood is usually intractable because Eq. (3.6) involves a high-dimensional integral. However, good approximation for marginal likelihood exists in Gaussian process model and [35] proposes a way of incorporating invariance into Gaussian process.

3.5 How to Construct Invariant Gaussian Process

Now with all the necessary tools being introduced, we can construct invariant GP and learn the maximal amount of transformation the GP should be invariant to. We start with the introduction of how to construct invariant GP.

Due to the fact that Gaussians are closed under summation, we can easily obtain an invariant GP by putting a GP prior on the function $g(\cdot)$ in Eq. (3.3). Specifically, when $g(\cdot) \sim \mathcal{GP}\left(0, k_g(\cdot,\cdot)\right)$, we have

$$f(\cdot) \sim \mathcal{GP}\left(m_f(\cdot), k_f(\cdot,\cdot)\right), \tag{3.7}$$

where

$$\begin{aligned} m_f(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})] \\ &= \mathbb{E}\left[\int p\left(\mathbf{x}_a \mid \mathbf{x}\right) g\left(\mathbf{x}_a\right) \mathrm{d}\mathbf{x}_a\right] \\ &= \int p\left(\mathbf{x}_a \mid \mathbf{x}\right) \mathbb{E}\left[g\left(\mathbf{x}_a\right)\right] \mathrm{d}\mathbf{x}_a \\ &= 0, \end{aligned} \tag{3.8}$$

and

$$\begin{aligned} k_f(\mathbf{x}, \mathbf{x}') &= \mathbb{E}\left[\left(f(\mathbf{x}) - m_f(\mathbf{x})\right)\left(f(\mathbf{x}') - m_f(\mathbf{x}')\right)\right] \\ &= \mathbb{E}\left[f(\mathbf{x}) f(\mathbf{x}')\right] \\ &= \mathbb{E}\left[\left(\int p(\mathbf{x}_a \mid \mathbf{x}) g(\mathbf{x}_a) \mathrm{d}\mathbf{x}_a\right)\left(\int p(\mathbf{x}'_a \mid \mathbf{x}') g(\mathbf{x}'_a) \mathrm{d}\mathbf{x}'_a\right)\right] \\ &= \iint p(\mathbf{x}_a \mid \mathbf{x}) p(\mathbf{x}'_a \mid \mathbf{x}') \mathbb{E}\left[g(\mathbf{x}_a) g(\mathbf{x}'_a)\right] \mathrm{d}\mathbf{x}_a \mathrm{d}\mathbf{x}'_a \\ &= \iint p(\mathbf{x}_a \mid \mathbf{x}) p(\mathbf{x}'_a \mid \mathbf{x}') k_g(\mathbf{x}_a, \mathbf{x}'_a) \mathrm{d}\mathbf{x}_a \mathrm{d}\mathbf{x}'_a. \end{aligned} \tag{3.9}$$

Then, the orbit parameters which control the maximal amount of rotation the GP should be invariant to become part of the kernel hyperparameters and can be learnt through regular GP training.

3.6 Inference in Invariant Gaussian Process

3.6.1 Variational Inference Using Inducing Points

To make invariant GPs applicable to large amount of data, here we use sparse variational approximation. As introduced in Section 2.2.3, given M inducing points $\mathbf{Z} = \{\mathbf{z}_m\}_{m=1}^M$ and the variational posterior $q(\mathbf{Z}) = \mathcal{N}\left(\mathbf{m}, \mathbf{S}\right)$, we have an approximate posterior $q(f) = \mathcal{GP}(\mu(\cdot), \nu(\cdot,\cdot))$ with

$$\mu(\cdot) = \mathbf{k}_{\mathbf{u}}^\top(\cdot)\mathbf{K}_{\mathbf{uu}}^{-1}\mathbf{m}, \quad \nu(\cdot,\cdot) = k(\cdot,\cdot) - \mathbf{k}_{\mathbf{u}}^\top(\cdot)\mathbf{K}_{\mathbf{uu}}^{-1}\left[\mathbf{K}_{\mathbf{uu}} - \mathbf{S}\right]\mathbf{K}_{\mathbf{uu}}^{-1}\mathbf{k}_{\mathbf{u}}(\cdot), \tag{3.10}$$

where $\mathbf{k}_{\mathbf{u}}(\cdot) = \left[k_f\left(\mathbf{z}_m, \cdot\right)\right]_{m=1}^M$ and $\left[\mathbf{K}_{\mathbf{uu}}\right]_{mm'} = k_f\left(\mathbf{z}_m, \mathbf{z}_{m'}\right)$.

The ELBO is

$$\mathcal{L} = \sum_{n=1}^{N} \mathbb{E}_{q(f(\mathbf{x}_n))} \left[\log p\left(y_n \mid f\left(\mathbf{x}_n\right)\right] - \mathbb{KL}[q(\mathbf{u}) \| p(\mathbf{u})\right] . \tag{3.11}$$

Note that the kernel $k_f\left(\cdot,\cdot\right)$ defined in Eq. (3.9) involves a double integral and therefore evaluating it exactly will be very computationally expensive, if not possible. We introduce two ways of evaluating Eq. (3.11) in Section 3.6.2 (for Gaussian likelihood only) [35] and Section 3.6.3 (for log-concave likelihoods) [30] respectively.

3.6.2 Variational Bound 1: Gaussian likelihood

Simplifying Double-sum Kernel by Inter-Domain Inducing Variables

We start by simplifying the calculation of double-sum kernel through inter-domain inducing variables.

By defining the values of inducing variables to be the same as the values evaluated through base function $g(\cdot)$, i.e., by defining

$$\mathbf{u}(\mathbf{z}) = \int f(\mathbf{x}) h(\mathbf{x}, \mathbf{z}) \mathrm{d}\mathbf{x} \stackrel{\mathrm{d}}{=} g(\mathbf{z}), \tag{3.12}$$

where $g(\cdot)$ is the base function used to construct invariant function $f(\cdot)$, we have

$$\begin{aligned} k_f(\mathbf{x}, \mathbf{z}) &\stackrel{\mathrm{d}}{=} \mathbb{E}[f(\mathbf{x}) u(\mathbf{z})] \\ &= \mathbb{E}[f(\mathbf{x}) g(\mathbf{z})] \\ &= \mathbb{E}\left[\int g\left(\mathbf{x}_a\right) p\left(\mathbf{x}_a \mid \mathbf{x}\right) \mathrm{d}\mathbf{x}_a g(\mathbf{z})\right] \\ &= \int p\left(\mathbf{x}_a \mid \mathbf{x}\right) \mathbb{E}\left[g\left(\mathbf{x}_a\right) g(\mathbf{z})\right] \mathrm{d}\mathbf{x}_a \\ &= \int p\left(\mathbf{x}_a \mid \mathbf{x}\right) k_g\left(\mathbf{x}_a, \mathbf{z}\right) \mathrm{d}\mathbf{x}_a, \end{aligned} \tag{3.13}$$

and

$$\begin{aligned} k_f(\mathbf{z}, \mathbf{z}') &\stackrel{\mathrm{d}}{=} \mathbb{E}\left[u\left(\mathbf{z}\right) u\left(\mathbf{z}'\right)\right] \\ &= \mathbb{E}\left[g\left(\mathbf{z}\right) g\left(\mathbf{z}'\right)\right] \\ &= k_g(\mathbf{z}, \mathbf{z}'). \end{aligned} \tag{3.14}$$

Then, the new covariances $k_f(\mathbf{x}, \mathbf{z})$ requires only a single integral and $k_f(\mathbf{z}, \mathbf{z}')$ requires no integral at all. Although $k_f\left(\mathbf{x}, \mathbf{x}'\right)$ still requires a double integral, as we will see in the next section, it can also be evaluated by only sampling once from the orbit distribution.

An Estimator Using Samples Describing Invariances

For Gaussian likelihood, the expectation in Eq. (3.11) can be evaluated analytically, giving the bound

$$\mathcal{L} = \sum_{n=1}^{N} \left[-\frac{1}{2} \log 2\pi\sigma^2 - \frac{1}{2\sigma^2} \left(y_n^2 - 2y_n\mu_n + \mu_n^2 + \sigma_n^2 \right) \right] - \mathbb{KL}[q(\mathbf{u}) \| p(\mathbf{u})], \tag{3.15}$$

where

$$\mu_n = \mu\left(\mathbf{x}_n\right) = \mathbf{k}_{\mathbf{f}_n\mathbf{u}} \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{m}, \tag{3.16}$$

$$\mu_n^2 = \mathbf{k}_{\mathbf{f}_n\mathbf{u}} \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{m}\mathbf{m}^\top \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{k}_{\mathbf{f}_n\mathbf{u}}^\top, \tag{3.17}$$

$$\sigma_n^2 = k_f\left(\mathbf{x}_n, \mathbf{x}_n\right) - \mathbf{k}_{\mathbf{f}_n\mathbf{u}} \mathbf{K}_{\mathbf{uu}}^{-1} \left(\mathbf{K}_{\mathbf{uu}} - \mathbf{S}\right) \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{k}_{\mathbf{u}\mathbf{f}_n}. \tag{3.18}$$

Note that $\mathrm{KL}[q(\mathbf{u}) \| p(\mathbf{u})]$ is tractable because by using inter-domain inducing variables, $\mathbf{K}_{\mathbf{uu}}$ can be evaluated directly (Eq. (3.14)). The intractable items are μ_n, μ_n^2 and σ_n^2.

For μ_n, the only intractable term is $\mathbf{k}_{\mathbf{f}_n\mathbf{u}}$, which can be evaluated by simple Monte Carlo estimation:

$$k_{\mathbf{fu}}(\mathbf{x}, \mathbf{z}) = \int p\left(\mathbf{x}_a \mid \mathbf{x}\right) \mathbf{k}_g\left(\mathbf{x}_a, \mathbf{z}\right) \mathrm{d}\mathbf{x}_a \Longrightarrow \hat{k}_{\mathbf{fu}}(\mathbf{x}, \mathbf{z}) = \frac{1}{S} \sum_{s=1}^{S} k_g\left(\mathbf{x}^{(s)}, \mathbf{z}\right), \quad \mathbf{x}^{(s)} \sim p\left(\mathbf{x}_a \mid \mathbf{x}\right). \tag{3.19}$$

This gives us the following estimation for μ_n:

$$\widehat{\mu_n} = \widehat{\mathbf{k}}_{\mathbf{f}_n\mathbf{u}} \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{m}. \tag{3.20}$$

For μ_n^2 and σ_n^2, we first rewrite them as

$$\mu_n^2 = \mathbf{k}_{\mathbf{f}_n\mathbf{u}} \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{m}\mathbf{m}^\top \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{k}_{\mathbf{f}_n\mathbf{u}}^\top = \mathrm{Tr}\left[\mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{m}\mathbf{m}^\top \mathbf{K}_{\mathbf{uu}}^{-1} \left(\mathbf{k}_{\mathbf{f}_n\mathbf{u}}^\top \mathbf{k}_{\mathbf{f}_n\mathbf{u}} \right) \right], \tag{3.21}$$

$$\sigma_n^2 = k_f\left(\mathbf{x}_n, \mathbf{x}_n\right) - \mathrm{Tr}\left[\mathbf{K}_{\mathbf{uu}}^{-1} \left(\mathbf{K}_{\mathbf{uu}} - \mathbf{S}\right) \mathbf{K}_{\mathbf{uu}}^{-1} \left(\mathbf{k}_{\mathbf{f}_n\mathbf{u}}^\top \mathbf{k}_{\mathbf{f}_n\mathbf{u}} \right) \right], \tag{3.22}$$

which allows us to focus on estimators of $k_f\left(\mathbf{x}_n, \mathbf{x}_n\right)$ and $\mathbf{k}_{\mathbf{f}_n\mathbf{u}}^\top \mathbf{k}_{\mathbf{f}_n\mathbf{u}}$. Note that $k_f\left(\mathbf{x}_n, \mathbf{x}_n\right)$ and an entry of $\mathbf{k}_{\mathbf{f}_n\mathbf{u}}^\top \mathbf{k}_{\mathbf{f}_n\mathbf{u}}$ can be treated identically as they can both be written as the integral

$$I = \iint p\left(\mathbf{x}_a \mid \mathbf{x}_n\right) p\left(\mathbf{x}'_a \mid \mathbf{x}_n\right) r\left(\mathbf{x}_a, \mathbf{x}'_a\right) \mathrm{d}\mathbf{x}_a \, \mathrm{d}\mathbf{x}'_a, \tag{3.23}$$

with $r = k_g\left(\mathbf{x}_a, \mathbf{x}'_a\right)$ and $r = k_g\left(\mathbf{x}_a, \mathbf{z}_m\right) k_g\left(\mathbf{x}'_a, \mathbf{z}_m\right)$ respectively. [35] proposes the following unbiased estimator which allows us to evaluate Eq. (3.23) by only sampling once from the orbit distribution:

$$\hat{I} = \frac{1}{S(S-1)} \sum_{s=1}^{S} \sum_{s'=1}^{S} r\left(\mathbf{x}^{(s)}, \mathbf{x}^{(s')}\right) \left(1 - \delta_{ss'}\right), \quad \mathbf{x}^{(s)} \sim p\left(\mathbf{x}_a \mid \mathbf{x}\right). \tag{3.24}$$

3.6.3 Variational Bound 2: Log-concave Likelihoods

Alternatively, we can bypass the calculation of the double-sum kernel $k_f(\cdot,\cdot)$ defined in Eq. (3.9) by using a lower bound of the ELBO in Eq. (3.11) [30]. At the expense of having a less tight bound of the marginal likelihood, we will be able to do easy inference in a wide class of likelihood.

Recall that invariant function $f(\cdot)$ is defined as

$$f(\mathbf{x}) \stackrel{\mathrm{d}}{=} \int_{\mathcal{O}_{\mathbf{x}}} g(\mathbf{x}_a)\, p(\mathbf{x}_a \mid \mathbf{x})\, \mathrm{d}\mathbf{x}_a. \tag{3.25}$$

Although $f(\cdot)$ might be intractable, it can be evaluated through Monte Carlo sampling:

$$\hat{f}(\mathbf{x}) = \frac{1}{S_o}\sum_{i=1}^{S_o} g\left(\mathbf{x}_a^i\right), \quad \mathbf{x}_a^i \sim p(\mathbf{x}_a \mid \mathbf{x}), \tag{3.26}$$

and we have:

$$\begin{aligned}
\mathbb{E}_{\prod_{j=1}^{S_o} p\left(\mathbf{x}_a^j \mid \mathbf{x}\right)}[\hat{f}(\mathbf{x})] &= \int \cdots \int \left[\frac{1}{S_o}\sum_{i=1}^{S_o} g\left(\mathbf{x}_a^i\right)\right] \prod_{j=1}^{S_o} p\left(\mathbf{x}_a^j \mid \mathbf{x}\right) \mathrm{d}\mathbf{x}_a^1 \ldots \mathrm{d}\mathbf{x}_a^{S_o} \\
&= \frac{1}{S_o}\int \cdots \int \sum_{i=1}^{S_o} \left[g\left(\mathbf{x}_a^i\right) \prod_{j=1}^{S_o} p\left(\mathbf{x}_a^j \mid \mathbf{x}\right)\right] \mathrm{d}\mathbf{x}_a^1 \ldots \mathrm{d}\mathbf{x}_a^{S_o} \\
&= \frac{1}{S_o}\int \sum_{i=1}^{S_o} g\left(\mathbf{x}_a^i\right) p\left(\mathbf{x}_a^i \mid \mathbf{x}\right) \mathrm{d}\mathbf{x}_a^i \\
&= f(\mathbf{x}).
\end{aligned} \tag{3.27}$$

Note that $f(\cdot)$ is deterministic in $\mathbf{x}$ and the randomness comes from the GP $g(\cdot)$. Therefore, for the likelihood $p(y \mid f)$ which is log-concave in f, we have

$$\begin{aligned}
\mathbb{E}_{q(f(\mathbf{x}))}\left[\log p\left(y \mid f(\mathbf{x})\right)\right] &= \mathbb{E}_{q(g)}\left[\log p\left(y \mid f(\mathbf{x})\right)\right] \\
&= \mathbb{E}_{q(g)}\left[\log p\left(y \mid \mathbb{E}_{\prod_{j=1}^{S_o} p\left(\mathbf{x}_a^j \mid \mathbf{x}\right)}\left[\hat{f}(\mathbf{x})\right]\right)\right] \\
&\underset{\text{Jensen's inequality}}{\geq} \mathbb{E}_{q(g)}\left[\mathbb{E}_{\prod_{j=1}^{S_o} p\left(\mathbf{x}_a^j \mid \mathbf{x}\right)}\left[\log p\left(y \mid \hat{f}(\mathbf{x})\right)\right]\right] \\
&= \mathbb{E}_{q(g)}\left[\mathbb{E}_{\prod_{j=1}^{S_o} p\left(\mathbf{x}_a^j \mid \mathbf{x}\right)}\left[\log p\left(y \mid \frac{1}{S_o}\sum_{i=1}^{S_o} g\left(\mathbf{x}_a^i\right)\right)\right]\right].
\end{aligned} \tag{3.28}$$

As a result, we have the following bound which allows for likelihoods that are log-concave in f:

$$\begin{aligned}
\log p(\mathbf{y}) &\geq \sum_{n=1}^{N} \mathbb{E}_{q(f(\mathbf{x}_n))} \left[\log p\left(y_n \mid f\left(\mathbf{x}_n\right)\right] - \mathrm{KL}[q(\mathbf{u}) \| p(\mathbf{u})\right] \\
&\geq \mathbb{E}_{q(g)} \left[\mathbb{E}_{\prod_{j=1}^{S_o} p\left(\mathbf{x}_a^j \mid \mathbf{x}\right)} \left[\log p\left(y_n \mid \frac{1}{S_o} \sum_{i=1}^{S_o} g\left(\mathbf{x}_a^i\right)\right)\right]\right] - \mathbb{KL}[q(\mathbf{u}) \| p(\mathbf{u})] \\
&\approx \frac{1}{S_g} \sum_{k=1}^{S_g} \frac{1}{S_{\mathcal{A}}} \sum_{j=1}^{S_{\mathcal{A}}} \log p\left(y_n \mid \frac{1}{S_o} \sum_{i=1}^{S_o} g_k\left(\mathbf{x}_a^{ji}\right)\right) - \mathbb{KL}[q(\mathbf{u}) \| p(\mathbf{u})]
\end{aligned} \tag{3.29}$$

To keep the above bound tight, we need to sample extensively from the approximate posterior $q(g)$ (the blue part in the above equation) and the orbit distribution $p\left(\mathbf{x}_a \mid \mathbf{x}\right)$ (the green part in the above equation). As shown in Section 2.2.5, we can efficiently sample from the approximate posterior $q(g)$ by using Matheron's rule [37]. Therefore, the main computational cost is sampling from the orbit distribution. Following [30], we use a large S_o and fix $S_{\mathcal{A}}$ to be 1.

Chapter 4

Learning Input-conditional Invariances via Invariant Gaussian Process

4.1 Setup

4.1.1 Hidden Invariance Class

To learn input-conditional invariance which truly captures the property of the given dataset, ideally we would like the model to assign each data point with distinct invariance. However, directly doing so (associate each data point with a corresponding orbit distribution) will inevitably result in a model that have too many parameters to be useful. Below we use an example to introduce the idea of hidden invariance class, which allows the model to assign each data point with distinct invariance with a limited amount of parameters.

Suppose the task we are interested in is digit classification and the invariance we would like the model to have is rotation invariance. Then, it is clear that there are at least three classes of invariance here: (i) digits like 0, which are invariant to large amount of rotation; (ii) digits like 6 and 9, which are invariant to small amount of rotation; (iii) the rest of the digits, which are invariant to medium amount of rotation. Based on this prior knowledge, we might divide the dataset into three subsets and refer to these subsets as hidden invariance classes. As each hidden invariance class has different levels of tolerance to rotations, we can model each hidden invariance class with an orbit distribution. Say we use $\mathbf{\Theta}_{\text{small}}$, $\mathbf{\Theta}_{\text{middle}}$ and $\mathbf{\Theta}_{\text{large}}$ to denote the parameters of each hidden invariance class's orbit distribution. Then, for each data point $\mathbf{x}_i$, we can obtain the parameters of its corresponding orbit distribution through a linear combination of $\mathbf{\Theta}_{\text{small}}$, $\mathbf{\Theta}_{\text{middle}}$ and $\mathbf{\Theta}_{\text{large}}$. Specifically, for each data point $\mathbf{x}_i$, we *learn* a continuous hidden representation $h(\mathbf{x}_i) = [h(\mathbf{x}_i)_{\text{small}}, h(\mathbf{x}_i)_{\text{middle}}, h(\mathbf{x}_i)_{\text{large}}]$ for it, which serves the purpose of the coefficients in the aforementioned linear combination. The parameters of its corresponding orbit distribution is then

computed as

$$\boldsymbol{\Theta}(\mathbf{x}_i) = h(\mathbf{x}_i)_{\text{small}}\boldsymbol{\Theta}_{\text{small}} + h(\mathbf{x}_i)_{\text{middle}}\boldsymbol{\Theta}_{\text{middle}} + h(\mathbf{x}_i)_{\text{large}}\boldsymbol{\Theta}_{\text{large}} \tag{4.1}$$

In general, we can pre-define a fixed number of hidden invariance classes based on the prior knowledge of the property of the dataset and the task of interest. Suppose there are H hidden invariance classes, the total parameters of orbit distributions would then be $\boldsymbol{\Theta} = [\boldsymbol{\Theta}_1, \ldots, \boldsymbol{\Theta}_H]^\top \in \mathbb{R}^{H\times V}$ where $\boldsymbol{\Theta}_j \in \mathbb{R}^{V\times 1}$ is the orbit distribution's parameter of the j-th hidden invariance class. Then, when given a hidden representation $\mathbf{h}_i = [h_{i1}, \ldots, h_{iH}]^\top \in \mathbb{R}^{H\times 1}$ for the input $\mathbf{x}_i$, the input-conditional orbit distribution's parameter can be computed as

$$\boldsymbol{\Theta}(\mathbf{x}_i) \overset{\text{d}}{=} \mathbf{h}_i^\top \boldsymbol{\Theta} = \sum_{j=1}^{H} \boldsymbol{\Theta}_j h_{ij}. \tag{4.2}$$

4.1.2 Two Types of Input-conditional Invariance

When we have strong prior knowledge of the dataset, we might already know which hidden invariance class each data point belongs to and it might be unnecessary to learn the hidden representation. Therefore, here we define two types of input-conditional invariance:

1. Type 1 input-conditional invariance
   Fix the hidden invariance class assignment and only learn the orbit parameters of each hidden invariance class

2. Type 2 input-conditional invariance
   Learn the hidden invariance class assignment along with the orbit parameters of each hidden invariance class

Because type 1 input-conditional invariance is a special case of type 2 input-conditional invariance (the learned hidden invariance class assignment happens to be the same as the pre-defined hidden invariance class assignment from prior knowledge), in the following sections we aim to design models that are capable of learning type 2 input-conditional invariance, i.e., learning the hidden class assignments and orbit parameters from data automatically.

4.2 Base Model

We first introduce the base model, which is capable of both learning type 1 input-conditional invariance and type 2 input-conditional invariance.

As the hidden representation $\mathbf{h}_i$ serves the purpose of coefficients in the linear combination of hidden invariance classes' orbit parameters, each elements of hidden representation $\mathbf{h}_i$ can in a sense be considered as the probability of data point $\mathbf{x}_i$ belonging to each hidden invariance class. Therefore, we can learn hidden representations $\mathbf{h}_i$ through a neural network with SoftMax output layer, such that each

element of $\mathbf{h}_i$ will sum up to 1. As this neural network encodes the inputs into their corresponding hidden representations, we refer to it as invariance encoder in the following context. The procedure of how input-conditional orbit parameters is computed is given in Fig. 4.1.

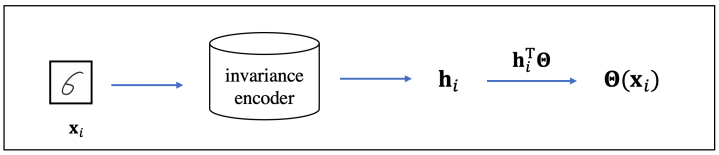


Figure 4.1: Procedure of computing input-conditional orbit parameters in the base model.

To learn type 1 input-conditional invariance, we train the invariance encoder according to our prior knowledge and then fix it. To learning type 2 input-conditional invariance, we train the invariance encoder along with the rest of the parameters.

4.3 Probabilistic Model

We now introduce the Bayesian version of the base model introduced in the previous section. Each input $\mathbf{x}_i$ is still associated with a hidden variable $\mathbf{h}_i$ and we use $\mathbf{h}_i$ to determine the input-conditional invariance. The difference is now we treat $\mathbf{h}_i$ as a random variable and infer its probability distribution. Also, note that the probabilistic model is only suitable to learn type 2 input-conditional invariance.

4.3.1 Model

As in the base model $\mathbf{h}_i$ is interpreted as the probabilities of $\mathbf{x}_i$ belonging to each hidden invariance class, it would be more natural to model it using the Dirichlet distribution. However, it is unclear how to incorporate prior knowledge of hidden invariance class assignment when using Dirichlet distribution. Therefore, we use Gaussian distribution to model $\mathbf{h}_i$ instead and abandon the interpretation of $\mathbf{h}_i$ here, i.e., we now just treat $\mathbf{h}_i$ as the coefficients in the linear combination of orbit parameters of hidden invariance class. When given $\mathbf{h}_i$, the input-conditional orbit parameters are still computed as Eq. (4.2).

Concatenating $\mathbf{x}_i$ and $\mathbf{h}_i$ together, now the input to the invariant GP $f(\cdot)$ becomes $\tilde{\mathbf{x}}_i = \{\mathbf{x}_i, \mathbf{h}_i\}$. The graphical model is given in Fig. 4.2 and the joint probability is

$$P(\mathbf{Y}, \mathbf{H}, f \mid \mathbf{X}) = P(f) \prod_{i=1}^{N} P(\mathbf{y}_i \mid f, \mathbf{x}_i, \mathbf{h}_i) P(\mathbf{h}_i \mid \mathbf{x}_i), \tag{4.3}$$

where $P(\mathbf{h}_i \mid \mathbf{x}_i) = \mathcal{N}\left(\mathbf{h}_i; \phi_P(\mathbf{x}_i), \sigma_P^2 \mathbf{I}\right)$, i.e., similar to what we did in the base model, we use a single neural network $\phi_P(\cdot)$ across all data points.

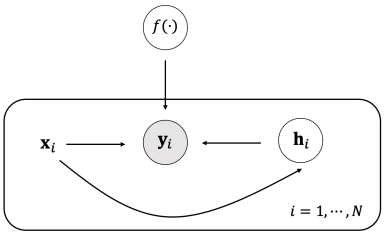


Figure 4.2: Graphical model of the probabilistic model.

4.3.2 Inference

Here we aim to find the posterior distribution of the invariant GP $f(\cdot)$ and the hidden variables $\mathbf{H} = \{\mathbf{h}_i\}_{i=1}^{H}$. Due to the intractability of marginal likelihood, we use the variational distribution to approximate the true posterior distribution and find the approximate posterior by minimizing the KL-divergence between the approximate and true posterior distribution. By assuming the independence of the GP posterior and hidden variables's posterior:

$$Q(f, \mathbf{H} \mid \mathbf{X}) \stackrel{\mathrm{d}}{=} Q(f)Q(\mathbf{H} \mid \mathbf{X}), \tag{4.4}$$

we will have

$$\begin{aligned}
\mathbb{KL}\left[Q(f, \mathbf{H} \mid \mathbf{X}) \mid\mid P(f, \mathbf{H} \mid \mathbf{y}, \mathbf{X})\right] &= \int Q(f, \mathbf{H} \mid \mathbf{X}) \log \frac{Q(f, \mathbf{H} \mid \mathbf{X})}{P(f, \mathbf{H} \mid \mathbf{y}, \mathbf{X})} \mathrm{d}f \mathrm{d}\mathbf{H} \\
&= \int Q(f, \mathbf{H} \mid \mathbf{X}) \log \frac{Q(f, \mathbf{H} \mid \mathbf{X}) P(\mathbf{y} \mid \mathbf{X})}{P(\mathbf{y}, \mathbf{H}, f \mid \mathbf{X})} \mathrm{d}f \mathrm{d}\mathbf{H} \\
&= \log P(\mathbf{y} \mid \mathbf{X}) + \int Q(f, \mathbf{H} \mid \mathbf{X}) \log \frac{Q(f, \mathbf{H} \mid \mathbf{X})}{P(\mathbf{y}, \mathbf{H}, f \mid \mathbf{X})} \mathrm{d}f \mathrm{d}\mathbf{H},
\end{aligned} \tag{4.5}$$

which implies

$$\min \mathbb{KL}\left[Q(f, \mathbf{H} \mid \mathbf{X}) \mid\mid P(f, \mathbf{H} \mid \mathbf{y}, \mathbf{X})\right] \iff \max \int Q(f, \mathbf{H} \mid \mathbf{X}) \log \frac{P(\mathbf{y}, \mathbf{H}, f \mid \mathbf{X})}{Q(f, \mathbf{H} \mid \mathbf{X})} \mathrm{d}f \mathrm{d}\mathbf{H}. \tag{4.6}$$

Therefore, we can equivalently find the approximate posterior by maximizing

$$\mathcal{L} = \int Q(f, \mathbf{H}) \log \frac{P(\mathbf{y}, \mathbf{H}, f \mid \mathbf{X})}{Q(f, \mathbf{H})} \mathrm{d}f \mathrm{d}\mathbf{H}. \tag{4.7}$$

For $Q(f)$, we use the standard sparse variational GP, which results $Q(f) = \mathcal{GP}(\mu(\cdot), \nu(\cdot, \cdot))$ and

$$\mu(\cdot) = \mathbf{k}_{\mathbf{u}}^{\top}(\cdot)\mathbf{K}_{\mathbf{uu}}^{-1}\mathbf{m}, \quad \nu(\cdot, \cdot) = k(\cdot, \cdot) - \mathbf{k}_{\mathbf{u}}^{\top}(\cdot)\mathbf{K}_{\mathbf{uu}}^{-1}\left[\mathbf{K}_{\mathbf{uu}} - \mathbf{S}\right]\mathbf{K}_{\mathbf{uu}}^{-1}\mathbf{k}_{\mathbf{u}}(\cdot). \tag{4.8}$$

For $Q(\mathbf{H})$, we assume

$$Q(\mathbf{H} \mid \mathbf{X}) \overset{\mathrm{d}}{=} \prod_{i=1}^{N} Q(\mathbf{h}_i \mid \mathbf{x}_i) = \prod_{i=1}^{N} \mathcal{N}\left(\mathbf{h}_i; \phi_Q(\mathbf{x}_i), \mathbf{\Sigma}_Q\right). \tag{4.9}$$

where $\mathbf{\Sigma}_Q = \mathrm{diag}[\sigma_{q1}^2, \ldots, \sigma_{qH}^2]$. Here we use amortized inference to alleviate computational cost and $\phi_Q(\cdot)$ is the neural network used to infer the posterior mean of $\mathbf{x}_i$.

Substituting Eq. (4.3), Eq. (4.4) and Eq. (4.9) into Eq. (4.7), we have

$$\begin{aligned}
\mathcal{L} &= \iint Q(f, \mathbf{H}) \log \frac{P(\mathbf{y}, \mathbf{H}, f \mid \mathbf{X})}{Q(f, \mathbf{H})} \mathrm{d}f \mathrm{d}\mathbf{H} \\
&= \iint Q(f) \prod_{j=1}^{N} Q(\mathbf{h}_j \mid \mathbf{x}_j) \log \frac{P(f) \prod_{i=1}^{N} P(\mathbf{y}_i \mid f, \mathbf{x}_i, \mathbf{h}_i) P(\mathbf{h}_i \mid \mathbf{x}_i)}{Q(f) \prod_{i=1}^{N} Q(\mathbf{h}_i \mid \mathbf{x}_i)} \mathrm{d}f \mathrm{d}\mathbf{H} \\
&= \sum_{i=1}^{N} \iint Q(f) \prod_{j=1}^{N} Q(\mathbf{h}_j \mid \mathbf{x}_j) \left(\log P(\mathbf{y}_i \mid f, \mathbf{x}_i, \mathbf{h}_i) + \log \frac{P(\mathbf{h}_i \mid \mathbf{x}_i)}{Q(\mathbf{h}_i \mid \mathbf{x}_i)} \right) \mathrm{d}f \mathrm{d}\mathbf{H} \\
&\qquad + \iint Q(f) \prod_{j=1}^{N} Q(\mathbf{h}_j \mid \mathbf{x}_j) \log \frac{P(f)}{Q(f)} \mathrm{d}f \mathrm{d}\mathbf{H} \\
&= \sum_{i=1}^{N} \left(\prod_{j \neq i}^{N} \int Q(\mathbf{h}_j \mid \mathbf{x}_j) \mathrm{d}\mathbf{h}_j \times \iint Q(f) Q(\mathbf{h}_i \mid \mathbf{x}_i) \log P(\mathbf{y}_i \mid f, \mathbf{x}_i, \mathbf{h}_i) \mathrm{d}f \mathrm{d}\mathbf{h}_i \right) \\
&\qquad + \sum_{i=1}^{N} \left(\prod_{j \neq i}^{N} \int Q(\mathbf{h}_j \mid \mathbf{x}_j) \mathrm{d}\mathbf{h}_j \times \int Q(f) \log \frac{P(f)}{Q(f)} \mathbf{d}f \times \int Q(\mathbf{h}_i \mid \mathbf{x}_i) \log \frac{P(\mathbf{h}_i \mid \mathbf{x}_i)}{Q(\mathbf{h}_i \mid \mathbf{x}_i)} \mathrm{d}\mathbf{h}_i \right) \\
&\qquad + \prod_{j=1}^{N} \int Q(\mathbf{h}_j \mid \mathbf{x}_j) \mathrm{d}\mathbf{h}_j \times \int Q(f) \log \frac{P(f)}{Q(f)} \mathbf{d}f \\
&= \sum_{i=1}^{N} \iint Q(f) Q(\mathbf{h}_i \mid \mathbf{x}_i) \log P(\mathbf{y}_i \mid f, \mathbf{x}_i, \mathbf{h}_i) \mathrm{d}f \mathrm{d}\mathbf{h}_i - \mathbb{KL}\left[Q(f) \mid\mid P(f)\right] - \mathbb{KL}\left[Q(\mathbf{H} \mid \mathbf{X}) \mid\mid P(\mathbf{H} \mid \mathbf{X})\right].
\end{aligned} \tag{4.10}$$

Because $\iint Q(f) Q(\mathbf{h}_i \mid \mathbf{x}_i) \log P(\mathbf{y}_i \mid f, \mathbf{x}_i, \mathbf{h}_i) \mathrm{d}f \mathrm{d}\mathbf{h}_i$ is intractable due to the intergal over $\mathbf{h}_i$, we use MC estimate:

$$\begin{aligned}
\iint Q(f) Q(\mathbf{h}_i \mid \mathbf{x}_i) \log P(\mathbf{y}_i \mid f, \mathbf{x}_i, \mathbf{h}_i) \mathrm{d}f \mathrm{d}\mathbf{h}_i &= \int Q(f) \left[\int Q(\mathbf{h}_i \mid \mathbf{x}_i) \log P(\mathbf{y}_i \mid f, \mathbf{x}_i, \mathbf{h}_i) \mathrm{d}\mathbf{h}_i \right] \mathrm{d}f \\
&\approx \int Q(f) \left[\frac{1}{K} \sum_{k=1}^{K} \log P(\mathbf{y}_i \mid f, \mathbf{x}_i, \mathbf{h}_i^{(k)}) \right] \mathrm{d}f, \quad \mathbf{h}_i^{(k)} \sim Q(\mathbf{h}_i \mid \mathbf{x}_i).
\end{aligned} \tag{4.11}$$

When using the Gaussian likelihood for the invariant GP, we will further have

$$\begin{aligned}
&\int Q(f)\left[\int Q(\mathbf{h}_i \mid \mathbf{x}_i)\log P(\mathbf{y}_i \mid f, \mathbf{x}_i, \mathbf{h}_i)\mathrm{d}\mathbf{h}_i\right]\mathrm{d}f \\
&\approx \int Q(f)\left[\frac{1}{K}\sum_{k=1}^{K}\log P(\mathbf{y}_i \mid f, \mathbf{x}_i, \mathbf{h}_i^{(k)})\right]\mathrm{d}f, \quad \mathbf{h}_i^{(k)} \sim Q(\mathbf{h}_i \mid \mathbf{x}_i) \\
&= \frac{1}{K}\sum_{k=1}^{K}\int Q(f)\log P\left(\mathbf{y}_i \mid f, \mathbf{x}_i, \mathbf{h}_i^{(k)}\right)\mathrm{d}f \\
&= \frac{1}{K}\sum_{k=1}^{K}\left[-\frac{1}{2}\log 2\pi\sigma_L^2 - \frac{1}{2\sigma_L^2}\left(y_n^2 - 2y_n\mu_i^{(k)} + {\mu_i^{(k)}}^2 + {\sigma_i^{(k)}}^2\right)\right],
\end{aligned} \tag{4.12}$$

where $\mu_i^{(k)} = \mu(\tilde{\mathbf{x}}_i^{(k)})$, ${\sigma_i^{(k)}}^2 = \nu(\tilde{\mathbf{x}}_i^{(k)}, \tilde{\mathbf{x}}_i^{(k)})$, $\tilde{\mathbf{x}}_i^{(k)} = \{\mathbf{x}_i, \mathbf{h}_i^{(k)}\}$. In practice to alleviate the computation cost, we set $K = 1$.

In summary, we can train probabilistic model by maximizing the following approximation to the ELBO

$$\log p(\mathbf{y}) \geq \sum_{i=1}^{N}\int Q(f)\log P\left(\mathbf{y}_i \mid f, \mathbf{x}_i, \mathbf{h}_i^{(k)}\right)\mathrm{d}f - \mathbb{KL}[Q(f)\|P(f)] - \mathbb{KL}[Q(\mathbf{H} \mid \mathbf{X})\|P(\mathbf{H} \mid \mathbf{X})], \tag{4.13}$$

where $\mathbf{h}_i^{(k)} \sim Q\left(\mathbf{h}_i \mid \mathbf{x}_i\right)$.

Chapter 5

Experimental Setup

In this chapter, we test which bound and likelihood are most suitable for learning input-conditional invariance. As type 2 input-conditional invariance is more difficult to learn, we focus on learning type 2 input-conditional invariance here. In Section 5.2, we compare the two bounds introduced in Section 3.6. In Section 5.3, we compare Gaussian and SoftMax likelihood.

5.1 Synthetic Rotated MNIST Dataset

To better compare different bounds and likelihoods, we create a synthetic dataset with distinct invariance in it by manually rotating digits by different amounts in a subset of the MNIST dataset. Specifically, we divide the digits into two classes, where class 1 contains the digits $\{0, 6, 7, 8, 9\}$ and class 2 contains the digits $\{1, 2, 3, 4, 5\}$. Then, the digits in class 1 are rotated by a randomly chosen angle $\alpha_1 \sim \text{Uniform}\,(-20°, 20°)$, and the digits in class 2 are rotated by a randomly chosen angle $\alpha_2 \sim \text{Uniform}\,(-150°, 150°)$. By doing so, we obtain a dataset with two hidden invariance classes. Examples of each digit class is given in Fig. 5.1.

In the synthetic dataset, the training set contains 10,000 data points with evenly distributed class, and the test set is the standard MNIST test set (10,000 data points) rotated by the methods described above.

Figure 5.1: Examples of digits in synthetic rotated MNIST.

5.2 Setup 1: Which Variational Bound to Use

In this section, we experiment with two variational bounds (both lower bound of log marginal likelihood) introduced in Section 3.6.2 and Section 3.6.3 to find which one is more suitable to learn input-conditional

invariance. As the first variational bound is only applicable to Gaussian likelihood, to compare these two variational bounds, the Gaussian likelihood will also be used for the second variational bound. For the model, we will use the base model introduced in Section 4.2.

5.2.1 Experiment Design

We use the following two experiments to find out which variational bound is more suitable to learn input-conditional invariance.

Experiment 1

The goal of this experiment is to find out which variational bound can better distinguish input-conditional invariant model and invariant model with uniform invariance. To do so, we use variational bound 1 and variational bound 2 to train an invariant GP with pre-encoded fixed ground-truth input-conditional invariance [1] (hereafter, we refer to this model as GP_I for simplicity) and an invariant GP with uniform invariance (hereafter, we refer to this model as GP_U for simplicity) on synthetic rotated MNIST dataset respectively. As GP_I has the ground-truth input-conditional invariance, it ideally should generalize better than GP_U and have a higher ELBO compared with GP_U. Therefore, the variational bound which is more suitable to learn input-conditional invariance should results in higher ELBO gap between GP_I and GP_U.

Experiment 2

The goal of this experiment is to find out which variational bound can better assign correct hidden invariance classes for different data. To do so, we train the base model with variational bound 1 and variational bound 2 respectively and compare the improvement of classification accuracy of the invariance encoder, which reflects the accuracy in assigning hidden invariance classes for different data. The variational bound which is more suitable should results in higher classification accuracy improvement in the invariance encoder.

5.2.2 Experiment Result

Experiment 1

The test accuracies of GP_I and GP_U when trained with variational bound 1 are given in Table 5.1, and the test accuracies of GP_I and GP_U when trained with variational bound 2 are given in Table 5.2. As expected, GP_I generalizes better than GP_U under both variational bounds.

[1] We can pre-encoded the ground-truth input-conditional invariance by initializing the orbit parameters of each hidden invariance class to the ground truth value, and then fix the pre-trained encoder.

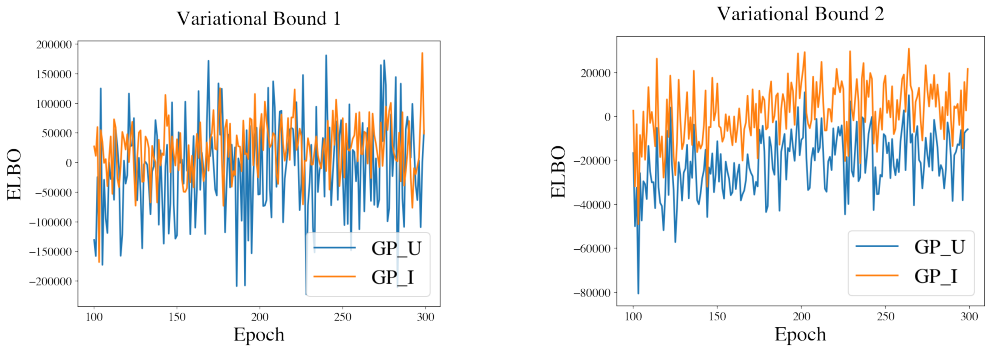
Model	Test Accuracy (%)
GP_U	91.45 ± 0.23
GP_I	93.97 ± 0.09

Table 5.1: Test Accuracies of GP_I and GP_U when trained with variational bound 1.

Model	Test Accuracy (%)
GP_U	91.99 ± 0.07
GP_I	93.82 ± 0.08

Table 5.2: Test Accuracies of GP_U and GP_I when trained with variational bound 2.

The ELBOs of GP_U and GP_I when trained with both variational bounds are plotted in Figure. 5.2. To help distinguish the ELBO gap when the model has converged, here we plot the ELBO from epoch 100, instead of epoch 1. As shown in Fig. 5.3(a), despite the fact that GP_I generalizes much better than GP_U, when trained with variational bound 1, these two invariant GPs have very similar ELBO when they converge. On the other hand, as shown in Fig. 5.3(b), when trained with variational bound 2, GP_I obtains higher ELBO than GP_U, which is consistent with GP_I's better generalization ability. Therefore, we conclude variational bound 2 is more suitable for learning input-conditional invariance from this experiment. We will discuss the reason for this later.



(a) ELBOs of variational bound 1.

(b) ELBOs of variational bound 2.

Figure 5.2: ELBOs of GP_U and GP_I when trained with two variational bounds. The ELBOs are plotted from epoch 100 to better distinguish the difference.

Experiment 2

The invariance encoder's accuracies on training set and test set when trained with variational bound 1 are given in Table 5.3, and the invariance encoder's accuracies on training set and test set when trained

with variational bound 2 are given in Table 5.4. We can see that when trained with variational bound 1, although the invariance encoder is initialized to a good spot, after training the accuracies drop to 50.32%. After investigating the output of the invariance encoder, it turns out almost all data points are assigned into the same hidden invariance class. As a result, the input-conditional invariant model reduces to the uniform invariant model. On the other side, when trained with variational bound 2, the accuracies on training set and test set both improved, which means the model learns meaningful hidden invariance class assignment from the data. Therefore, we conclude variational bound 2 is more suitable for learning input-conditional invariance from this experiment.

	Accuracy (%)
Training set, before training	81.46
Training set, after training	50.32
Test set, before training	80.40
Test set, after training	50.32

	Accuracy Improvement (%)
Training set	-31.14
Test set	-30.08

Table 5.3: Invariance encoders' accuracies on training set and test set when trained with variational bound 1.

	Accuracy (%)
Training set, before training	82.04
Training set, after training	89.20
Test set, before training	81.28
Test set, after training	86.25

	Accuracy Improvement (%)
Training set	7.16
Test set	4.97

Table 5.4: Invariance encoders' accuracies on training set and test set when trained with variational bound 2.

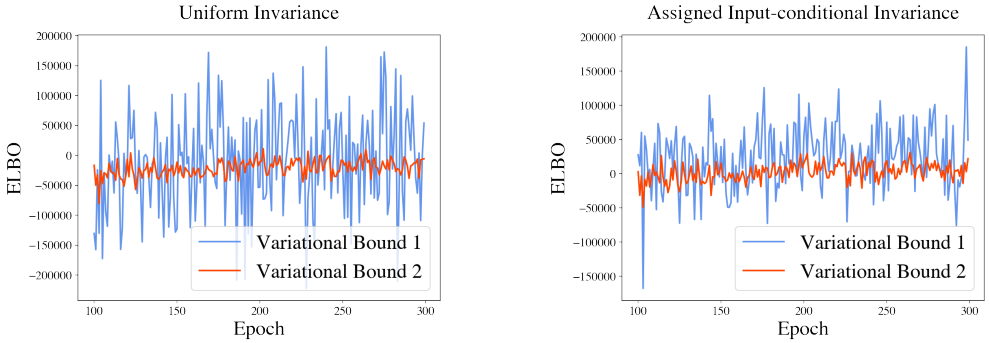
Conclusion

From the results of these two experiments, we conclude that variational bound 2 is more suitable for learning input-conditional invariance because it is capable of: (1) distinguishing between input-conditional invariant model and uniform invariant model; (2) assigning correct hidden invariance classes for different inputs.

5.2.3 Discussion

The main reason variational bound 1 is not suitable to learn input-conditional invariance might be because the estimated variational bound is very noisy, i.e., it has very high variance. Recall that in variational

bound 1, there are three separated terms which need Monte Carlo estimation (Eq. (3.15)). Then, as shown in Fig. 5.3, when compared with MC-estimated variational bound 2, MC-estimated variational bound 1 has much higher variance. As there are so much noise, the important signal becomes deeply buried in it. Therefore, despite the fact that variational bound 2 is a lower bound of variational bound 1, it is more suitable to learn input-conditional invariance as it has lower variance and therefore able to more clearly identify the better model.



(a) ELBOs of invariant GP with uniform invariance when trained with variational bound 1 and variational bound 2.

(b) ELBOs of invariant GP with input-conditional invariance when trained with variational bound 1 and variational bound 2.

Figure 5.3: ELBOs of GP_U and GP_I when trained with two variational bounds. The ELBOs are plotted from epoch 100.

5.3 Setup 2: Which Likelihood to Use

As variational bound 2 allows for likelihood which is log-concave such as SoftMax, in this section we compare SoftMax likelihood and Gaussian likelihood. Conceptually SoftMax likelihood is more suitable for classification task, however, as we will discuss below, our experiment results show that Gaussian likelihood is more suitable to learn input-conditional invariance.

5.3.1 Experiment Design

Assigning correct hidden invariance classes for different data is the fundamental requirement for the model to learn input-conditional invariance. Therefore, here we use the improvement in the accuracy of the invariance encoder in the base model as the metric for which we use to evaluate the models. The larger the improvement is, the more suitable the likelihood.

5.3.2 Experiment Result

The invariance encoders' accuracies when trained with SoftMax and Gaussian Likelihood are given in Table 5.5 and Table 5.6 respectively. Compared with Gaussian likelihood, when trained with SoftMax

likelihood the accuracies of hidden invariance class assignment only improve slightly. Therefore, we will only use the Gaussian likelihood in our experiments in the following chapter.

	Accuracy (%)
Training set, before training	81.49
Training set, after training	82.66
Test set, before training	80.39
Test set, after training	80.51

	Accuracy Improvement (%)
Training set	1.17
Test set	0.12

Table 5.5: Invariance encoders' accuracies on training set and test set when trained with SoftMax likelihood.

	Accuracy (%)
Training set, before training	82.04
Training set, after training	89.20
Test set, before training	81.28
Test set, after training	86.25

	Accuracy Improvement (%)
Training set	7.16
Test set	4.97

Table 5.6: Invariance encoders' accuracies on training set and test set when trained with Gaussian Likelihood.

5.3.3 Discussion

Similar to our result here, in experiment result in [30], Gaussian likelihood also performs much better when learning uniform invariance from dataset. This might be due to the fact that SoftMax is not really a likelihood, but a calibration function, i.e., SoftMax does not express incertitude and it is not a probability density function.

5.4 Conclusion

Based on the experiment results in the above sections, we use variational bound 2 and Gaussian likelihood in the following experiments because:

- Variational bound 2 is capable of: (1) distinguishing between input-conditional invariant model and uniform invariant model; (2) assigning correct hidden invariance classes for different inputs.
- Gaussian likelihood results in higher accuracy improvement in the invariance encoder, which means it can better assign correct hidden invariance classes for different inputs.

Chapter 6

Experiment

In this chapter, we test the proposed models by learning type 1 input-conditional invariance and type 2 input-conditional invariance. The task we are focused on is image classification. The invariance of interest is rotation invariance, which means the orbit distribution is parameterized by the maximal rotation angle.

6.1 Synthetic Rotated MNIST

In this section, we conduct experiments on the synthetic rotated MNIST introduced in Section 5.1.

6.1.1 Type 1 Input-conditional Invariance

Because there are two manually created invariance classes in the synthetic rotated MNIST dataset, we use the base model to learn type 1 input-conditional invariance and compare the performance with the invariant GP with uniform invariance. The test accuracies are given in Table 6.1. Invariant GP with input-conditional invariance successfully outperforms invariant GP with uniform invariance. As shown in the ELBOs plotted in Fig. 6.1, Invariant GP with input-conditional invariance also has higher ELBO compared with invariant GP with uniform invariance.

	Test Accuracy (%)
Uniform Invariance	91.99 ± 0.07
Assigned Hidden Invariance Class	94.57 ± 0.06

Table 6.1: Test accuracies of invariant GPs with uniform invariance and input-conditional invariance in the synthetic rotated MNIST. In the input-conditional invariance GP, hidden invariance class assignment is fixed, i.e., we are learning type 1 input-conditional invariance here.

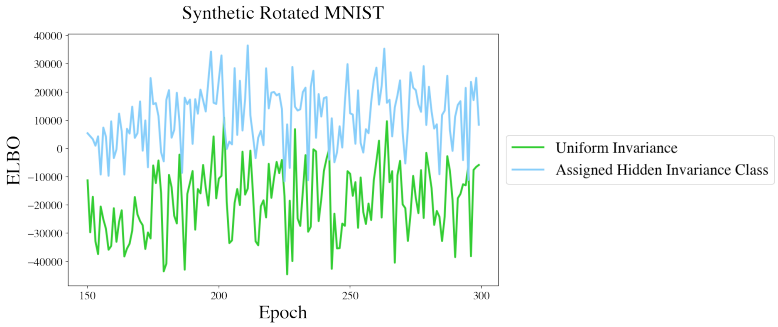


Figure 6.1: ELBOs of invariant GP with uniform invariance and input-conditional invariance in the synthetic rotated MNIST. ELBOs are plotted from epoch 150 to better compare the difference.

To further test whether the model truly learns meaningful input-conditional invariance, we compared the orbit parameters learnt by the model with the ground truth values. As shown in Table 6.2, the learnt orbit parameters mimic the behavior of the ground truth orbit parameters: hidden invariance class 1 has higher rotation angle and hidden invariance class 2 has smaller rotation angle.

	Hidden Invariance Class 1 (°)	Hidden Invariance Class 2 (°)
Ground Truth	150	20
Learnt Orbit Parameters	179.89	39.01

Table 6.2: Orbit parameters of different hidden invariance classes learnt by the model and the ground truth orbit parameters in subset of MNIST.

6.1.2 Type 2 Input-conditional Invariance

In this section we train the base model and probabilistic model to learn type 2 input-conditional invariance and compare the learnt invariance with ground truth values to see whether the proposed models are capable of learning type 2 input-conditional invariance. The test accuracies are given in Table 6.10 and the learnt orbit parameters are given in Table 6.4. The ELBOs of the different models are plotted in Fig 6.2.

As shown in Table 6.10, both the base model and the probabilistic model outperform the invariant GP with uniform invariance. And as shown in Table 6.4, the learnt orbit parameters in both models are close to the ground truth values. Therefore, we conclude that when there are distinct invariances in the dataset, the proposed models are capable of learning useful input-conditional invariance from data. Also, as shown in Fig. 6.2 ELBO can be used successfully for model selection (the model that generalizes better has a higher ELBO).

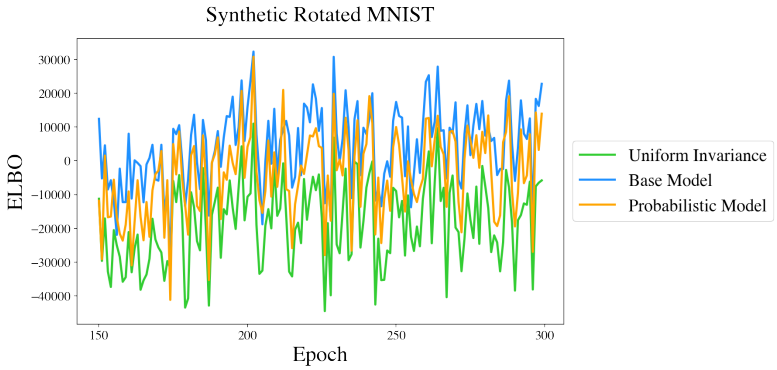


Figure 6.2: ELBOs of invariant GP with uniform invariance and input-conditional invariance in the synthetic rotated MNIST. ELBOs are plotted from epoch 150 to better compare the difference.

	Test Accuracy (%)
Uniform Invariance	91.99 ± 0.07
Base Model	92.93 ± 0.13
Probabilistic Model	92.41 ± 0.10

Table 6.3: Test accuracies of invariant GPs with uniform invariance and input-conditional invariance in synthetic rotated MNIST. In the input-conditional invariance GP, hidden invariance class assignment are learnt along with orbit parameters, i.e., we are learning type 2 input-conditional invariance here.

	Hidden Invariance Class 1 (°)	Hidden Invariance Class 2 (°)
Ground Truth	150	20
Learnt orbit parameters in base model	179.58	27.43
Learnt orbit parameters in probabilistic model	179.88	14.08

Table 6.4: Orbit parameters of different hidden invariance classes learnt by the base model and the probabilistic model in synthetic rotated MNIST.

6.1.3 Influence of Prior Knowledge

In this section, we investigate the influence of prior knowledge on the model's ability of learning input-conditional invariance. We use the base model here and incorporating different degrees of prior knowledge by initializing the invariance encoder and the orbit parameters differently. We consider the following four settings:

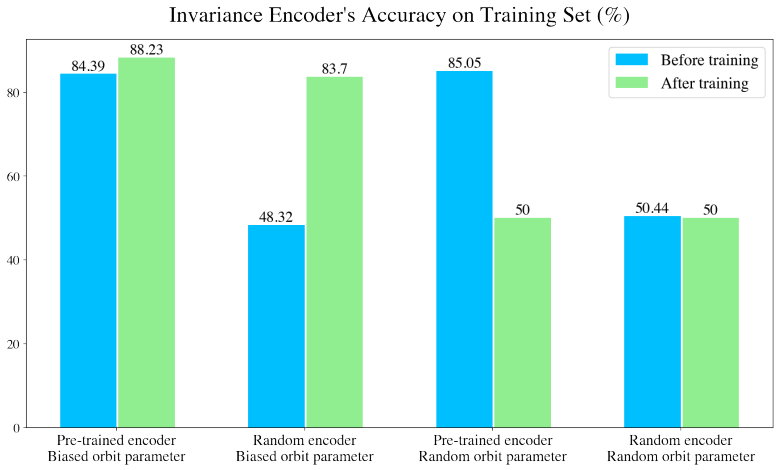
- Pre-trained invariance encoder (around 80% accuracy on both training and test set) + biased orbit parameters initialization (hidden invariance class 1 and 2 are initialized as 120° and 30° respectively)
- Randomly initialized invariance encoder (around 50% accuracy on both training and test set) + biased orbit parameters initialization (hidden invariance class 1 and 2 are initialized as 120° and 30° respectively)
- Pre-trained invariance encoder (around 80% accuracy on both training and test set) + randomly orbit parameters initialization (hidden invariance class 1 and 2 are both initialized as 1°)
- Randomly initialized invariance encoder (around 50% accuracy on both training and test set) + randomly orbit parameters initialization (hidden invariance class 1 and 2 are both initialized as 1°)

The invariance encoder's accuracies on training set and test set are plotted in Fig. 6.3. The learnt orbit parameters are given in Table 6.5. The test accuracies of base models with different initialization are given in Table 6.6. The ELBOs of base models with different initialization are plotted in Fig. 6.4.

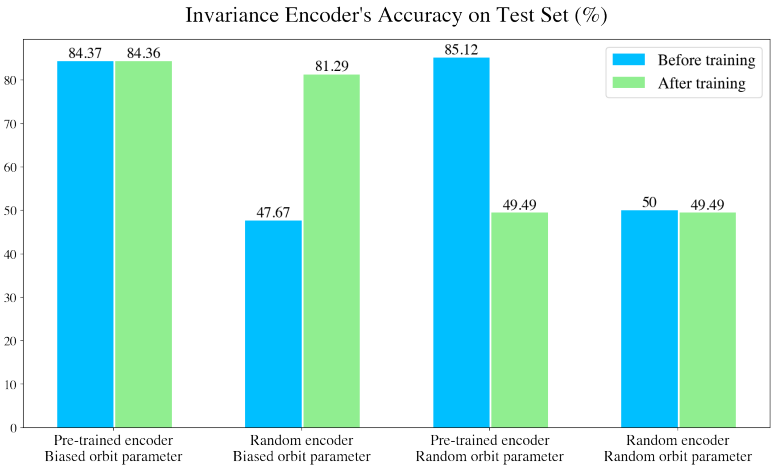
As shown in Fig. 6.3 and Table 6.5, the initialization of invariance encoder doesn't have much influence on the model's ability of assigning correct hidden invariance class to different data points. The initialization of orbit parameters however, plays the key role in model's ability of assigning correct hidden invariance class to different data points. When initialized with biased orbit parameters which reflects the difference between two hidden invariance classes, even with randomly initialized invariance encoder, the model can still correctly assign hidden invariance class to different data points and learn meaningful orbit parameters. When initialized with orbit parameters that doesn't reflect the difference between hidden invariance classes, even the invariance encoder is pre-trained to achieve about 80% accuracy on both training and test set, after training the input-conditional model still assigns the same hidden invariance class to all data points and therefore reduces to the invariant model with uniform invariance.

As shown in Table 6.6, consistent with the learnt input-conditional invariance in base model with different initializations (Fig. 6.3 and Table 6.5), when the model learns meaningful input-conditional invariance it generalizes better than the model with uniform invariance. As shown in Fig. 6.4, the difference in generalization is also reflected in the models' ELBOs.

In summary, we find that the initialization of orbit parameters has direct influence on model's ability of learning input-conditional invariance. To let the model learn meaningful input-conditional invariance, we need to initialize the model with biased orbit parameters which reflects the difference between hidden invariance classes.



(a) Invariance encoders' accuracies on training set with different initialization.



(b) Invariance encoders' accuracies on test set with different initialization.

Figure 6.3: Invariance encoders' accuracies on training set and test set with different initialization in the synthetic rotated MNIST dataset.

	Hidden Invariance Class 1 (°)	Hidden Invariance Class 2 (°)
Ground Truth	150	20
Pre-trained encoder Biased orbit parameter	179.58	27.43
Random encoder Biased orbit parameter	179.22	23.90
Pre-trained encoder Random orbit parameter	1.02	51.20
Random encoder Random orbit parameter	1.02	50.88

Table 6.5: Orbit parameters of different hidden invariance classes learnt by base model with different initializations in the synthetic rotated MNIST dataset. In the last two models, as all inputs are assigned into hidden invariance class 2, orbit parameter of hidden invariance class 1 is rarely updated.

	Test Accuracy (%)
Uniform Invariance	91.99 ± 0.07
Pre-trained encoder Biased orbit parameter initialization	92.93 ± 0.13
Random encoder initialization Biased orbit parameter initialization	92.70 ± 0.09
Pre-trained encoder Random orbit parameter initialization	92.20 ± 0.09
Random encoder initialization Random orbit parameter initialization	91.93 ± 0.11

Table 6.6: Test accuracy of base models with different initializations in the synthetic rotated MNIST dataset. To help compare with the invariant GP with uniform invariance, we also include its test accuracy here.

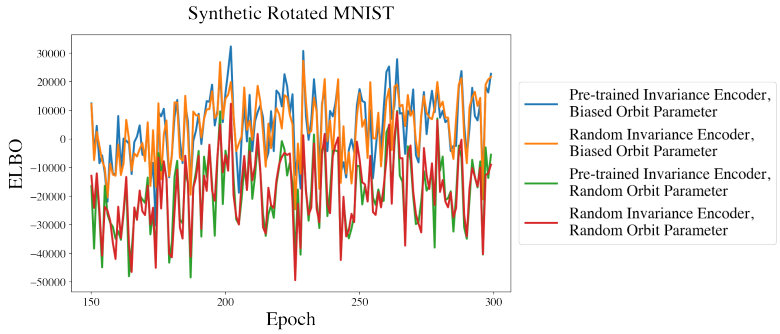


Figure 6.4: ELBOs of base model with different initializations in the synthetic rotated MNIST dataset. ELBOs are plotted from epoch 150 to better compare the difference.

6.2 Subset of MNIST

When training data are scarce it is usually hard to obtain a model with good generalization ability. Therefore, in this section we focus on low data regime, which is an ideal case to test model's generalization ability. Specifically, we train the proposed models on a subset of MNIST dataset (2500 training data in total with evenly distributed classes) and compare their performances with invariant model with uniform invariance.

6.2.1 Determine the Number of Hidden Invariance Classes

To determine the number of hidden invariance classes efficiently, we first assume each digit class has its own invariance, then learn type 1 input-conditional invariance from the subset of MNIST. Once we have the learnt orbit parameter of each hidden invariance class, we can determine the number of hidden invariance classes based on how the orbit parameters are clustered together after training.

The test accuracies of the invariant GP with uniform invariance and the invariant GP with input-conditional invariance are given in Table 6.9. As the invariant GP with input-conditional invariance has higher test accuracy, we can conclude that in this subset of MNIST dataset there exists useful input-conditional invariance.

	Test Accuracy (%)
Uniform Invariance	94.12 ± 0.08
Assigned Hidden Invariance Class	95.14 ± 0.11

Table 6.7: Test accuracies of invariant GPs with uniform invariance and input-conditional invariance in the subset of MNIST. In the input-conditional invariance GP, hidden invariance class assignment is fixed, i.e., we are learning type 1 input-conditional invariance here.

The learnt orbit parameters is given in Table 6.8. Due to the cluster structure exhibited by these orbit parameters, we use three hidden invariance classes in the following experiment, where hidden invariance class 1 consists of digit 0 and digit 2, hidden invariance class 2 consists of digit 3, and hidden invariance class 3 consists of the rest digits.

hidden invariance class	0	1	2	3	4	5	6	7	8	9
orbit parameter (°)	64.35	15.18	41.76	132.93	17.28	22.37	22.75	14.55	16.13	11.82

Table 6.8: Orbit parameters of each hidden invariance class in the subset of MNIST. Here each hidden invariance class corresponds to the digit class. To better illustrate the cluster structure of the orbit parameters, we use different color to highlight different cluster.

6.2.2 Type 1 Input-conditional Invariance

The test accuracies are given in Table 6.9. We can see that the invariant GP with input-conditional invariance successfully outperforms the invariant GP with uniform invariance. As shown in the ELBOs plotted in Fig. 6.5, invariant GP with input-conditional invariance also has higher ELBO compared with the invariant GP with uniform invariance.

	Test Accuracy (%)
Uniform Invariance	94.12 ± 0.08
Assigned Hidden Invariance Class	95.05 ± 0.05

Table 6.9: Test accuracies of invariant GPs with uniform invariance and input-conditional invariance in the subset of MNIST. In the input-conditional invariance GP, hidden invariance class assignment is fixed, i.e., we are learning type 1 input-conditional invariance here.

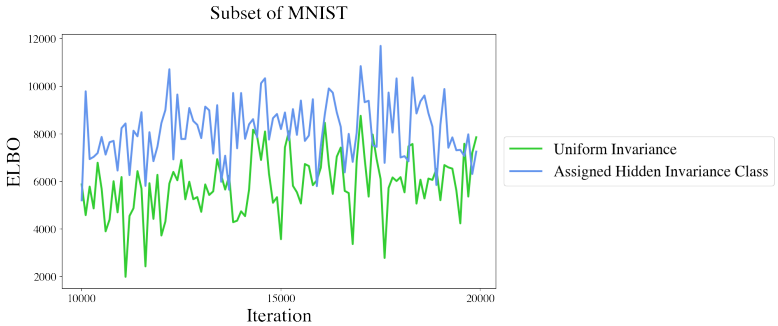


Figure 6.5: ELBOs of invariant GP with uniform invariance and input-conditional invariance in the subset of MNIST. ELBOs are plotted from iteration 10000 to better compare the difference.

6.2.3 Type 2 Input-conditional Invariance

The test accuracies are given in Table 6.10 and the ELBOs are plotted in Fig. 6.10. When learning hidden invariance class assignment along with orbit parameters, proposed models don't outperform the invariant GP with uniform invariance. This is due to the following two reasons: (1) In the subset of MNIST, the difference between different hidden invariance classes is more subtle than synthetic rotated MNIST. Therefore, it becomes harder to assign correct hidden invariance class assignment for each data point. (2) The hidden invariance classes are imbalanced in subset of MNIST. Here hidden invariance class 1 has 500 data points, hidden invariance class 2 has 1750 data points and hidden invariance class 3 has 250 data points. As assigning hidden invariance class is similar to classification, the imbalanced dataset makes it harder to correctly assign hidden invariance class.

	Test Accuracy (%)
Uniform Invariance	94.12 ± 0.08
Base Model	91.08 ± 0.14
Probabilistic Model	94.06 ± 0.08

Table 6.10: Test accuracies of invariant GPs with uniform invariance and input-conditional invariance in subset of MNIST. In the input-conditional invariance GP, hidden invariance class assignment are learnt along with orbit parameters, i.e., we are learning type 2 input-conditional invariance here.

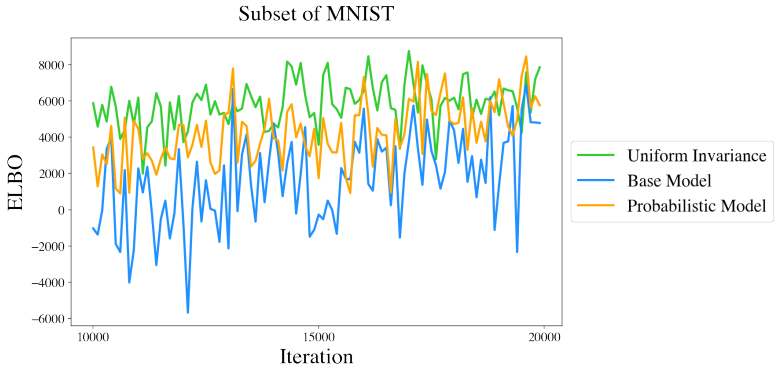


Figure 6.6: ELBOs of invariant GP with uniform invariance and input-conditional invariance in the subset of MNIST. ELBOs are plotted from iteration 10000 to better compare the difference.

To further analyze the learnt invariance, we give the learnt orbit parameters in Table 6.11 and the accuracies of invariance encoder in Table 6.12. As the learnt orbit parameters of different hidden invariance classes are very similar in the probabilistic model, it essentially becomes more like invariant GP with uniform invariance, and therefore has similar generalization ability as the invariant GP with uniform invariance. The reason base model generalizes the worst is because despite the fact that the learnt orbit

parameters are closer to ground truth values compared with probabilistic model, the accuracies of invariance encoder in the base model are very low. Then, by assigning hidden invariance class 3 to digits like 6 and 9, the aforementioned digits will be rotated too much and become hard to recognize.

	Hidden Invariance Class 1 (°)	Hidden Invariance Class 2 (°)	Hidden Invariance Class 3 (°)
"Ground Truth"	17.22	45.75	135.07
Base Model	11.73	27.41	116.58
Probabilistic Model	17.83	30.62	20.56

Table 6.11: Orbit parameters of different hidden invariance classes learnt by the base model and the probabilistic model in subset of MNIST. Here we used the learnt orbit parameters in type 1 input-conditional invariance as the ground truth values. In both base model and probabilistic model, the orbit parameters are initialized to be $[10, 40, 100]$.

	Training Set Accuracy (%)	Test Set Accuracy (%)
Base Model	19.08	20.62
Probabilistic Model	91.68	90.97

Table 6.12: The accuracies of invariance encoder on training set and test set in subset of MNIST.

Chapter 7

Conclusions

7.1 Summary

In this thesis, we aimed to design invariant GPs that were capable of learning different amount of transformations for different inputs such that the proposed models were expected to generalize well when given a limited amount of data. To do so, we proposed the idea of hidden invariance class and learnt input-conditional invariance through the linear combination of the orbit parameters of hidden invariance classes. Two models were designed where in the probabilistic model the aforementioned linear combination coefficients were treated as random variables and in the base model we used the point estimation of the aforementioned linear combination coefficients. To test the proposed models' ability of learning input-conditional transformation, we first compared two variational bounds and two likelihoods experimentally to decide which setting is most suitable to learn input-conditional invariance. Then, we conducted experiments on two datasets: (1) synthetic rotated MNIST where the difference between hidden invariance classes is distinct; (2) subset of MNIST where the difference between hidden invariance classes is subtle. From the experiment results, the proposed models are capable of learning input-conditional transformation when the difference between hidden invariance classes is distinct.

The advantages of the proposed models are:

- When different subsets of data have distinct levels of tolerance to a certain type of transformation, the proposed models are capable of learning different amount of transformations for different inputs from training data alone.
- By allowing the model to assign different amount of transformation to different inputs, when the model learns meaningful input-conditional invariance, it will be very data-efficient and generalize well.

The limitations of the proposed models are:

- Pre-defining the number of hidden invariance classes requires strong prior knowledge about the dataset, which limits the usage of the proposed models.

- When different subset of data don't have distinct levels of tolerance to a certain type of transformation, it is very hard for the proposed models to learn different amount of transformations for different inputs from training data alone.

7.2 Future Work

This project has raised many questions and we list some of them as future directions below:

- A major limitation of the proposed models is that we need to pre-define the number of hidden invariance classes, which in practice might be a hard thing to do. One way to improve the current model is to learn the number of hidden invariance class as well through Bayesian nonparametric clustering such as Dirichlet process mixture model.
- As shown in the experiment setup, the estimation of ELBO plays an important role in learning meaningful input-conditional invariance. By designing new estimation of ELBO which further reduces the variance of the estimated ELBO, it is possible to learn input-conditional invariance more accurately.
- In this project we only experiment with rotation, which is a rather simple transformation. As the difference between different digits in MNIST are more general than the rotation degree, it would be beneficial to try affine transformation in the future.
- In this project we only use the GP setting to learn invariance. Because the neural network is more flexible compared with the GP, it is worth investigating how to learn input-conditional invariance in the neural network setting such that the proposed models can be used in broader application domains such as molecular property prediction.

Bibliography

[1] Antreas Antoniou, Amos J. Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. CoRR, abs/1711.04340, 2017.

[2] Gregory W. Benton, Marc Finzi, Pavel Izmailov, and Andrew Gordon Wilson. Learning invariances in neural networks from training data. In Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.

[3] David Beymer and Tomaso A. Poggio. Face recognition from one example view. In Procedings of the Fifth International Conference on Computer Vision (ICCV 95), Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, June 20-23, 1995, pages 500–507. IEEE Computer Society, 1995.

[4] Léon Bottou, Olivier Chapelle, Dennis DeCoste, and Jason Weston. Training invariant svms using selective sampling. Large scale kernel machines, pages 301–320, 2007.

[5] Olivier Chapelle and Bernhard Schölkopf. Incorporating invariances in non-linear support vector machines. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada], pages 609–616. MIT Press, 2001.

[6] Taco Cohen and Max Welling. Group equivariant convolutional networks. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016, volume 48 of JMLR Workshop and Conference Proceedings, pages 2990–2999. JMLR.org, 2016.

[7] Ekin D. Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019, pages 113–123. Computer Vision Foundation / IEEE, 2019.

[8] Alexander G. de G. Matthews, James Hensman, Richard E. Turner, and Zoubin Ghahramani. On sparse variational methods and the kullback-leibler divergence between stochastic processes. In

Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016, volume 51 of JMLR Workshop and Conference Proceedings, pages 231–239. JMLR.org, 2016.

[9] Marc Peter Deisenroth. Inference in time series, 2020. https://deisenroth.cc/teaching/2020-21/ml-seminar/time-series.pdf,.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA, pages 248–255. IEEE Computer Society, 2009.

[11] Pascal Germain, Francis R. Bach, Alexandre Lacoste, and Simon Lacoste-Julien. Pac-bayesian theory meets bayesian inference. In Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, pages 1876–1884, 2016.

[12] David Ginsbourger, Xavier Bay, Olivier Roustant, and Laurent Carraro. Argumentwise invariant kernels for the approximation of invariant functions. In Annales de la Faculté des sciences de Toulouse: Mathématiques, volume 21, pages 501–527, 2012.

[13] David Ginsbourger, Olivier Roustant, and Nicolas Durrande. On degeneracy and invariances of random fields paths with applications in gaussian process modelling. Journal of statistical planning and inference, 170:117–128, 2016.

[14] Thore Graepel and Ralf Herbrich. Invariant pattern recognition by semi-definite programming machines. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf, editors, Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada], pages 33–40. MIT Press, 2003.

[15] Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Faster autoaugment: Learning augmentation strategies using backpropagation. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXV, volume 12370 of Lecture Notes in Computer Science, pages 1–16. Springer, 2020.

[16] Søren Hauberg, Oren Freifeld, Anders Boesen Lindbo Larsen, John W. Fisher III, and Lars Kai Hansen. Dreaming more data: Class-dependent distributions over diffeomorphisms for learned data augmentation. In Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016, volume 51 of JMLR Workshop and Conference Proceedings, pages 342–350. JMLR.org, 2016.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, pages 770–778. IEEE Computer Society, 2016.

[18] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. In Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada, pages 2017–2025, 2015.

[19] Diederik P. Kingma and Jimmy Ba. In 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.

[20] Imre Risi Kondor. Group theoretical methods in machine learning. Columbia University, 2008.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States, pages 1106–1114, 2012.

[22] Miguel Lázaro-Gredilla and Aníbal R. Figueiras-Vidal. Inter-domain gaussian processes for sparse inference using inducing features. In Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada, pages 1087–1095. Curran Associates, Inc., 2009.

[23] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. Neural Comput., 1(4):541–551, 1989.

[24] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast autoaugment. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pages 6662–6672, 2019.

[25] David J. C. MacKay. Information theory, inference, and learning algorithms. Cambridge University Press, 2003.

[26] Diego Marcos, Michele Volpi, Nikos Komodakis, and Devis Tuia. Rotation equivariant vector field networks. In IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017, pages 5058–5067. IEEE Computer Society, 2017.

[27] Partha Niyogi, Federico Girosi, and Tomaso Poggio. Incorporating prior information in machine learning by creating virtual examples. Proceedings of the IEEE, 86(11):2196–2209, 1998.

[28] Carl Edward Rasmussen and Zoubin Ghahramani. In Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA, pages 294–300. MIT Press, 2000.

[29] Carl Edward Rasmussen and Christopher K. I. Williams. Gaussian processes for machine learning. Adaptive computation and machine learning. MIT Press, 2006.

[30] Pola Elisabeth Schwöbel, Martin Jørgensen, Sebastian W. Ober, and Mark van der Wilk. Last layer marginal likelihood for invariance learning. CoRR, abs/2106.07512, 2021.

[31] Matthias Seeger. Bayesian Gaussian process models: PAC-Bayesian generalisation error bounds and sparse approximations. 2003.

[32] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005, December 5-8, 2005, Vancouver, British Columbia, Canada], pages 1257–1264, 2005.

[33] Ivan Sosnovik, Michal Szmaja, and Arnold W. M. Smeulders. Scale-equivariant steerable networks. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020.

[34] Michalis K. Titsias. Variational learning of inducing variables in sparse gaussian processes. In Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009, volume 5 of JMLR Proceedings, pages 567–574. JMLR.org, 2009.

[35] Mark van der Wilk, Matthias Bauer, S. T. John, and James Hensman. Learning invariances using the marginal likelihood. In Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, pages 9960–9970, 2018.

[36] Mark van der Wilk, Carl Edward Rasmussen, and James Hensman. Convolutional gaussian processes. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 2849–2858, 2017.

[37] James T. Wilson, Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, and Marc Peter Deisenroth. Efficiently sampling functions from gaussian process posteriors. In Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, volume 119 of Proceedings of Machine Learning Research, pages 10292–10302. PMLR, 2020.

[38] James T. Wilson and Alexander Terenin. Efficiently sampling functions from gaussian process posteriors, 2020. `https://sml-group.cc/blog/2020-gp-sampling/`,.

[39] Daniel E. Worrall, Stephan J. Garbin, Daniyar Turmukhambetov, and Gabriel J. Brostow. Harmonic networks: Deep translation and rotation equivariance. In 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017, pages 7168–7177. IEEE Computer Society, 2017.

[40] Daniel E. Worrall and Max Welling. Deep scale-spaces: Equivariance over scale. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pages 7364–7376, 2019.

[41] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, pages 13001–13008. AAAI Press, 2020.