Computational Science and Engineering
(International Master's Program)

Technische Universität München

Master's Thesis

Data-driven multi-fidelity modeling for terramechanics

Tzu-Chien Chang

Computational Science and Engineering
(International Master's Program)

Technische Universität München

Master's Thesis

Data-driven multi-fidelity modeling for terramechanics

Author:	Tzu-Chien Chang
Supervisor:	Prof. Dr. Hans-Joachim Bungartz
Advisor:	Dr. Felix Dietrich
Submission Date:	April 15th, 2022

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

April 15th, 2022 Tzu-Chien Chang

Acknowledgments

First of all, I would like to thank my advisor, Dr. Felix Dietrich, for his continuous support and guidance over the entire duration of the thesis work. His experiences and suggestions have been really helpful for me to plan and organize my thesis.

Secondly, I would like to thank Vladyslav Fediukov for his help about generating the data, giving me directions in the beginning and discussing with me. Good luck on the PhD journey!

Furthermore, I feel extremely blessed and lucky for being in CSE and TUM community. Words cannot describe how much I have learned and grown here. I also made several close, like-minded and interesting friends who all made this adventure worthwhile.

Last but not least, I am very grateful to my family and girlfriend for their unconditional support and love. Thank you!

"When something is important enough, you do it even if the odds are not in your favor."

-Elon Musk

Abstract

Our knowledge to wheel-soil interaction is crucial to the success of Mars rovers, since it can affect the desired forces and torques of the engine and thus determine the movement of the robot. Both terrestrial simulations and real-world experiments have been employed to assist the decision of rovers. However, expensive computations and costs resulting from intensive simulations and experimental setup have hindered their further development.

In this thesis, we use multi-fidelity modeling to fuse data such as wheel trajectory and surface geometry from different fidelity sources. These data act as the input to both neural networks and Gaussian process to build efficient multi-fidelity models. In low-fidelity models, we use lightweight Multilayer perceptron (MLP), Gaussian process regression (GPR) and convolutional neural networks (CNN) trained by relatively unreliable but abundant data sources, aiming to extract useful patterns at a lower evaluation cost. These models are then incorporated with another GP model trained by limited medium fidelity data using a multi-fidelity framework, Nonlinear autoregressive Gaussian processes (NARGP) to capture more trustworthy information. Throughout this work, we use two levels of fidelity sources, build six different models and compare with one another. Among all, NARGP+CNN model gives the best prediction to the force of the wheeled rover by fusing features of the robots and surface images. It also performs well in terms of certainty of prediction (CoP), a term we coined. In addition, several choices of kernels are also researched in order to give directions to the future endeavors. Our result demonstrates the applicability of data-driven approaches in the realm of terramechanics.

Contents

1 Introduction

Data-driven approaches have gained tremendous attention over the past decade. On the one hand, we are generating much more data and having the ability to store and compute such exponentially growing amount of data. On the other hand, the rapid development of machine learning allows us to extract useful information from the data and has potential to enhance or replace existing methods in many domains. Under the giant umbrella of machine learning, Gaussian process has always been one of the popular methods due to its nature to provide reliable estimate to the prediction and the uncertainty [1]. When we stack multiple Gaussian process together, possibly with many other trivial models, and feed each one of them with different data sources, it is called multi-fidelity modeling [2]. Multi-fidelity modeling has been widely used in many disciplines because of its ability to learn from rather less high fidelity data points and capture useful information from the abundant low fidelity, noise data. For instance, it is proved to be really successful in aerospace engineering design [3, 4] such as designing rotor blade for a helicopter [5].

Space exploration is always known to be expensive, since each mission can cost billions of dollars, not to mention the sophisticated preparations beforehand [6]. In order to drive the technological progress forward, scientists and engineers have developed numerous experiments to reduce cost without directly accessing the hardware itself. Virtual simulations using theoretical or empirical models are particularly successful. For example, normal and shear force calculations are employed widely in previous models [7, 8]. However, most of them suffer from either high computational complexity [9] or unsatisfactory results [10]. In addition, the extraterrestrial environment has also been replicated as far as possible in a laboratory to allow researchers to run relatively cheap and frequent experiments [11]. Although the cost is high, we can still generate real and high-fidelity data that might be identical to the data retrieved at the surface where real mission is placed.

The space community strongly needs to have fast and accurate models to describe the exploration rovers. Thankfully, both simulations and real-world experiments can provide us an enormous amount of data. This brings us to the initial motivation of the thesis.

The goal of the thesis is to explore the applicability of data-driven methods on the locomotion of wheeled mobile rovers. Since we have data from different fidelities and each one can provide different perspectives to the underlying problem, the choice of multi-fidelity modeling becomes straightforward. With the help of multi-fidelity models, we can fuse data from different sources, such as low-cost virtual simulations or real-world experi-

ments. Not only do we add engine related data to the models, but we also provide surface geometry information and use CNN to boost the performance of the models.

In section 2, foremost, a small portion of domain knowledge about rover mechanics is explained. Although it is just the tip of the iceberg, it should be enough to understand current development of wheeled mobile robot and should help you grasp the concepts of terramechanics easier. Furthermore, basic idea about neural networks including multi-layer perceptron and convolutional neural networks is also covered. We will also explain Gaussian process and then step by step to different forms of multi-fidelity modeling. At the end of the section, we spend some paragraphs on uncertainty quantification and its unavoidable existence in machine learning.

Section 3 is devoted to the result of the thesis, which is the major contribution of the work. We start from a few paragraphs on the motivations and project overview. We then explain the shape and the properties of the training data and move on to the goal of the thesis and our chosen evaluation metrics. The majority of the section is the computational experiments section, where we tried multiple models, ranging from MLP to pure GP and GP+CNN merged by the NARGP framework. The models architectures as well as their hyperparameters will be revealed in order to reproduce the results. At the end of the section, we will discuss all the models altogether and perform another small experiment on the influence of kernels.

To summarize the thesis, the final section will conclude our results and provide directions for the further endeavors.

2 Background Theory

2.1 Rover Mechanics

Over the past few decades, humans have developed many rovers to explore the unknowns, including but not limited to moon and Mars exploration [6, 12]. Rovers typically consist of wheels since it is proved to be easier to design and for the rovers to navigate around [13], so we sometimes also call them wheeled mobile robots. The surfaces being explored are generally soft materials like soil or sand. Thus, it is not uncommon for the rovers to be trapped in such surfaces [12]. Since the wheel is such an important part of the rovers, and it is also the only contact to the ground, we can simplify rover mechanics to the physical interactions between wheels and soil. Scientists use both real-world experiments and virtual simulations to study the behaviors of the rovers during potential planetary explorations. In the case of real-world experiments, it is typically set up in a laboratory and mimics the environment of, for example, Mars surface, where there are containers full of sand and wheels with multiple sensors being installed to measure properties like forces and torques in multiple directions [11]. Please refer to Figure 2.1 for graphical illustration. As for virtual simulations, various models were proposed, ranging from high-cost but high fidelity to cheap but relatively deviated models [13, 14]. For instance, one can construct the virtual spaces based on physical formulas and use numerical methods like finite elements to abstract and approach the problems [15]. Apart from the physical properties such as movements, velocities and accelerations, images of the surfaces can also be useful for us to extract information that can potentially facilitate the decision-making process. In summary, understanding the wheel-soil interactions is the key to any rover-dependent surface exploration.

2.2 Neural Networks

Neural networks, as its name suggests, are inspired biologically by the human nervous system. It is composed of interconnected neurons inspired by the connections in the human brain. The input of the neurons is a vector of values which is averaged according to the weights, and activation function can decide how much signal this neuron is going to send to the next neurons, while in human brain the signal is electrically transferred and stimulated. During the training process, the predicted result will be compared with

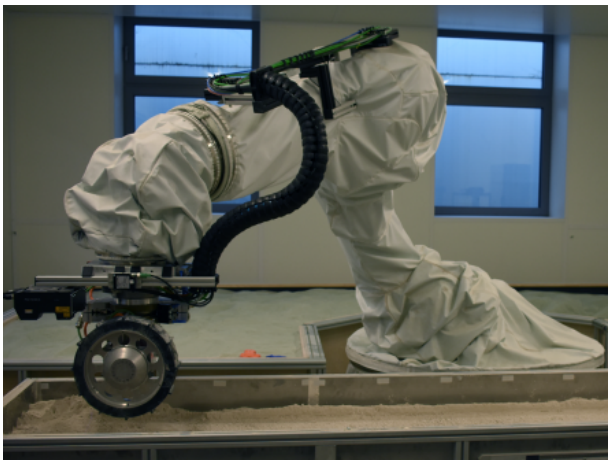


Figure 2.1: Real-world experiment setting of a wheel in a laboratory. This figure is cropped from [11].

the ground truth based on objective function (loss function), and the weights will soon be updated by gradient based methods to achieve better representation of the data.

This section explains the basics of the artificial neural networks (ANN) [16]. We start by talking about multilayer perceptron (MLP) and closely elaborate its building blocks. Moreover, we take an overview of the convolutional neural network (CNN) including its structures and components and elaborate how it differs from the brute force neural networks. We will use these two architectures in chapter 3, so it is necessary to understand its theoretical fundamentals and how it works so as to modify and optimize the models.

2.2.1 Multilayer perceptron

MLP is the most common type of neural network, and it often forms the cornerstone of other advanced neural networks. We also call it dense neural network, where each neuron in n layer is connected to each neuron in n+1 layer. Figure 2.2 shows an example of an MLP architecture with three hidden layers. In the following subsections, we are going to decompose MLP and explain each component one by one in details.

Neurons

In Figure 2.3a, we can see that a neuron can receive impulses from any dendrite. The impulses then travel through the cell body and to the end of the axon before sending the impulse to the next neurons. Artificial neuron was created as an analogy to the biological one. As illustrated in Figure 2.3b, normally we numericalize the property into x_i, x_i will then be multiplied by w_i. Each neuron summarizes all the $w_i \cdot x_i$ sent from all previous

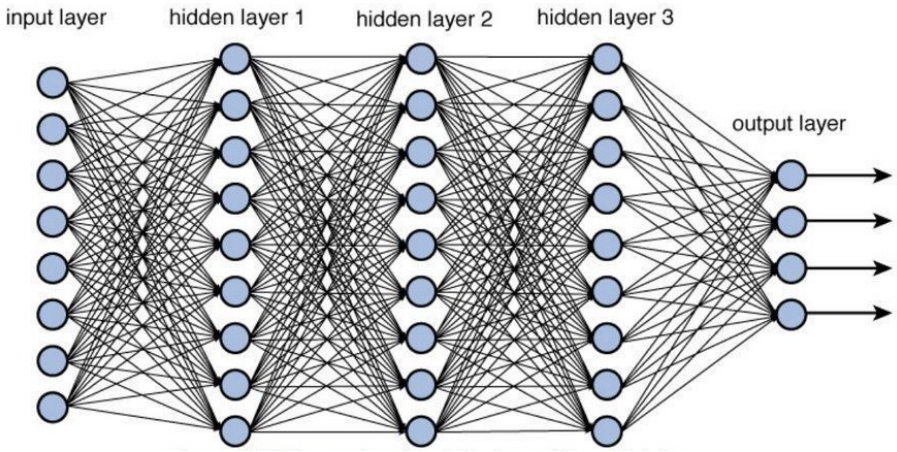
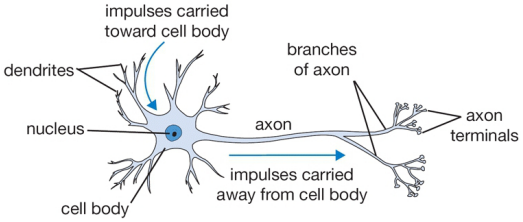
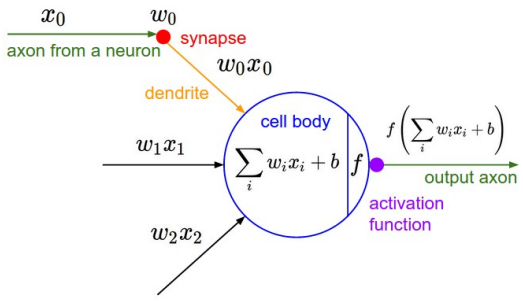


Figure 2.2: Graphical representation of an MLP with three hidden layers. Each neuron is connected to each neuron in the previous and next layer. This figure is downloaded from [17].

neurons and adds a bias b_i. Before outputting the signal to the next neurons, the sum will have to pass through an activation function to give non-linearity to the operation. Both w_i and b_i are trainable. Activation function should also be carefully chosen to maximize the model performance.



(a) A biological neuron.



(b) Single neuron in an ANN. w: weights, x: input from previous neuron, b: bias.

Figure 2.3: Comparison between a biological neuron and an artificial neuron. Both figures are from [18].

Activation functions

The main reason of the activation function is to give non-linearity to the networks so that it can learn more complex patterns. Since generally people choose gradient-based meth-

ods to optimize the networks' parameters, the activation functions should be differentiable across $\mathbb{R}$. Moreover, it is more favorable for the calculation of the gradient to be computationally cheap. In the following, we introduce some of the most commonly used activation functions.

- **Sigmoid**: The function is defined as

$$\sigma(z) = \frac{1}{1 + e^{(-z)}}$$

and its curve is plotted in Figure 2.4. Ideally, we want the input to be as close to the middle as possible, where the gradient is at a reasonable range. Otherwise, we can see from the graph that as $z > 5$ or $z < -5$, the gradient is extremely close to 0, which would prevent the networks from updating the weights effectively.

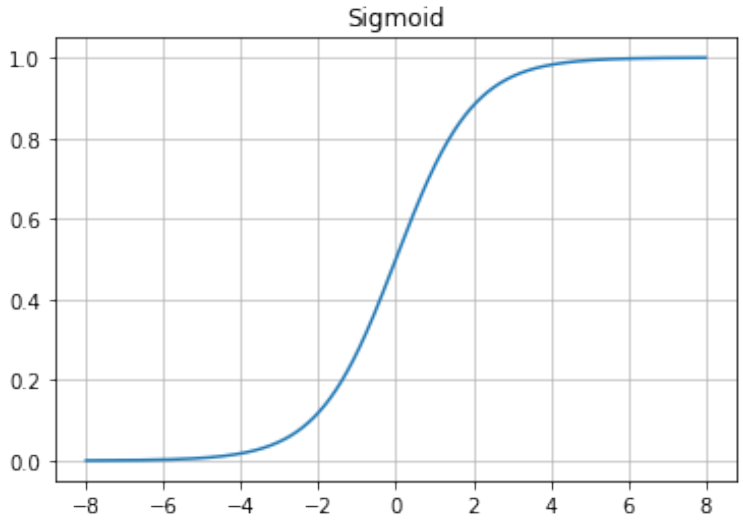


Figure 2.4: Sigmoid Function.

- **Relu**: Relu as shown in the following

$$R(z) = max(0, z)$$

is the default choice for a neural network at the moment. Despite empirically having better performance, it also has several advantages over others. First of all, the function itself is really easy to compute, which is fairly significant for inference and training. Secondly, although it can lead to dead neurons, we can think of it as a way to filter out weak signal and thus result in a relatively sparse output. Lastly, its derivative is quite simple. Note that relu is actually not differentiable at $z = 0$, so normally the derivative 0 is used at that point. The shape of the function is depicted in Figure 2.5.

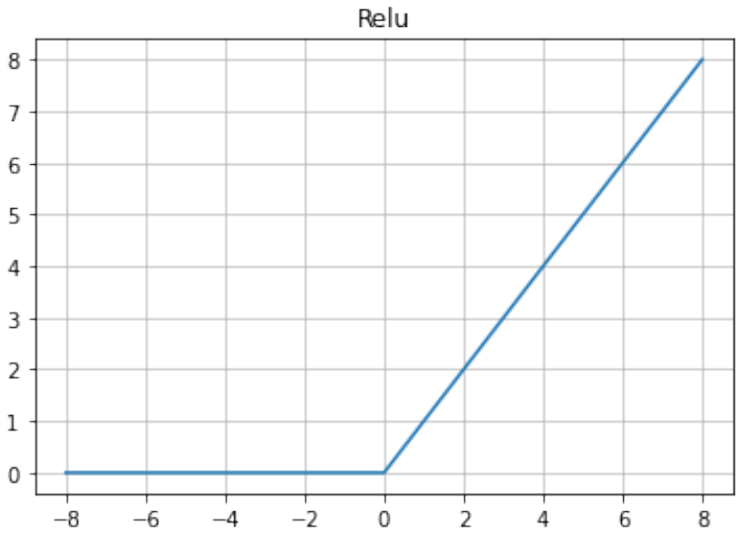


Figure 2.5: Rectified linear Unit (relu) Function.

- **Tanh**: The tanh non-linearity, defined as

$$f(z) = tanh(z)$$

, is demonstrated in Figure 2.6. Its functionality and shape is very similar to sigmoid, where all the real value is mapped to $[-1, 1]$. Also, we would hope the input to be close to zero so that the gradient can be more meaningful. Batch normalization [19] is really helpful when tackling saturating nonlinearities, so it is also being used in this thesis.

Loss functions

Loss function is one of the key determining factors to the performance of a neural network, since it directly defines the objective of a model. The existence of a loss function makes the training of a neural network possible, since it makes the training process a minimization problem. In supervised learning, what we're trying to minimize is the customized distance between the output of the parameterized neural network and the ground truth.

The choice of a loss function is largely dependent on the concrete problem that we are trying to solve. Most commonly, Mean Square Error (MSE) (L2 loss) shown in Equation 2.1 is used as a default choice in regression problem, whereas Cross Entropy Loss in Equation

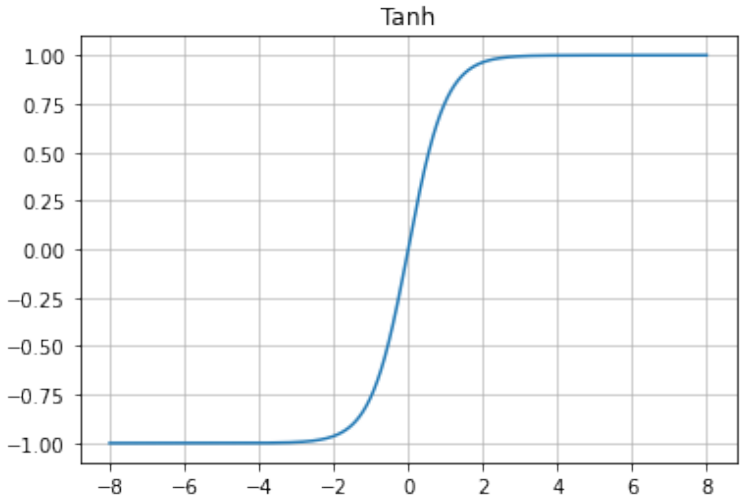


Figure 2.6: Tanh Function.

2.2 is popular when dealing with classification problem.

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} ||f(x_i, \theta) - y_i||_2^2 \tag{2.1}$$

$$L(\theta) = -\sum_{i=1}^{n}\sum_{k=1}^{n}(f(x_{ik}, \theta) \cdot \log y_{ik}) \tag{2.2}$$

Typically, in a neural network, there are tens of thousands of parameters θ if not millions. This leads to a multidimensional optimization problem when trying to find the best model. In the next section, we will show the foundation of all the gradient-based method, gradient descent.

Gradient descent

Gradient descent (GD) is an iterative optimization algorithm aiming to find the minimum of a function. The formula of gradient descent is defined as:

$$\theta_{i+1} = \theta_i - \alpha\nabla_\theta L(\theta_i) \tag{2.3}$$

where θ represents all the parameters in the neural network, α is the learning rate and L is the loss function. Theoretically, the minimum point from the initial set of parameters

can be found if we use a while loop to wrap the above formula. We can see that gradient descent works by subtracting a step of the gradient at that specific point in the parameter space. Since GD takes all the training data into account, it is not uncommon for the algorithm to be trapped in the local minimum instead of the global one. In addition, by tuning the learning rate, the convergence speed can be controlled, while a large α can cause oscillating training curve and a small one can result in tedious training time.

While being a popular method, the followings are some of the most used variants of gradient descent.

- **Stochastic Gradient Descent**: Stochastic Gradient Descent (SGD) is one of the most simple and effective way to avoid being trapped in the local minimum [20]. The way it works is by setting $n = k$ in Equation 2.1. k is a subset of samples in the training set, and we call it *batch size*. It is reported that the expectation can be approximated with a small subset of the data. It is also somewhat empirical to find the best batch size. On one hand, while smaller batch size means more variance across gradients and hence noisier, it has the possibility to help the algorithm break through local minimum. On the other hand, the choice of larger batch size is mostly limited by training hardware's memory.

- **Adam**: Adam optimizer is often the method of choice for neural networks [21]. Adam takes the concept of momentum and RMSProp [22]. Momentum is a way to take previous gradients into account while stepping toward the next point. For example, we would hope the algorithm go faster in the directions where gradients are accumulated over time. RMSProp was invented by dividing the learning rate by an exponentially decayed average of squared gradients. It can help dampen the oscillations in the high-variance directions and travel faster in flat surface. Adam combines first and second order momentum, meaning the mean and variance of gradients. The superior performance comes with more hyperparameters, but it generally works fine to use the default value suggested by authors [21]. This is also the optimizer being used in this thesis.

Back-propagation

Back-propagation leverages chain rule to compute gradients in a modular way [23]. Please have a look at Figure 2.7. Firstly, there is a forward pass where we follow the black arrows and calculate the output of the function until the result of the loss function is acquired. Subsequently, the backward pass involves the flow of the gradients (navy blue arrows). Basically, it allows us to calculate the derivatives of given variables with respect to loss function easily.

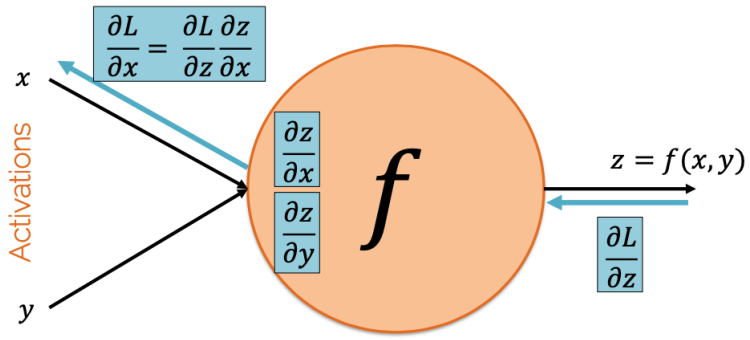


Figure 2.7: The flow of gradients. The graph is cropped from [24].

2.2.2 Convolutional Neural Network

Theoretically, with MLP, we should be able to approximate any function. However, when it comes to spatial or temporal data, blindly flattening the input could make us lose significant information. Nowadays, Convolutional Neural Network (CNN) has become de facto choice for image recognition problem [25]. In the following subsections, we are going to discuss some of the basic components of a CNN.

Convolutional layer

We can think of convolution as a way to do image filtering. For example, some kernels can capture the edges better, while some can transform a colorful image into a black-and-white one. As we can see in Figure 2.8, for an input of size $h \times w \times d$, we need to prepare k kernels with the shape $h' \times w' \times d'$, where $h' \leq h, w' \leq w$ and $d' = d$. Each kernel scans through the image from left to right, from top to bottom, and does some basic mathematical operations in the overlapped region. The number of the kernels k becomes the depth d of the next layer. As for height and width, there are two more parameters called stride s and padding p. These allow us to define the output height and width as:

$$h_{output} = \frac{h - h' + 2p}{s} + 1, w_{output} = \frac{w - w' + 2p}{s} + 1 \tag{2.4}$$

where stride denotes how long the convolution jumps that the kernel is moving each step along the direction, and padding is a technique that wraps the input image with layers of zeros. The output resolution can be reduced with higher stride; hence with reasonable choice of stride the convolution operation can be more efficient.

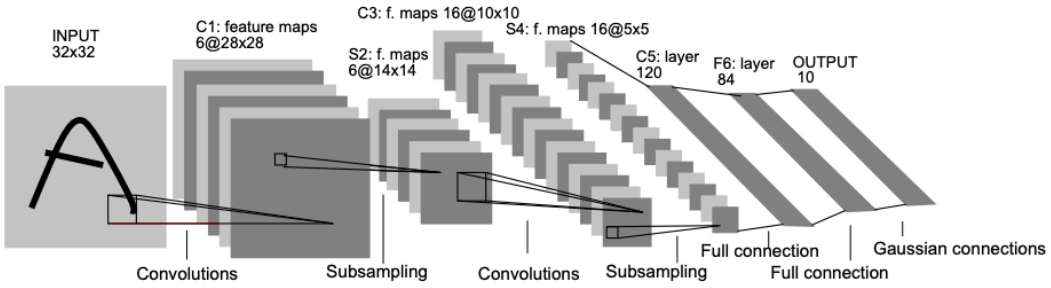


Figure 2.8: Example of a LeNet-5 CNN architecture. Figure from [25].

Pooling layer

Please take a look at the step *Subsampling* in Figure 2.8. Pooling layer is a process to reduce the output size by summarizing the values in the pooling window. Some of the common choices of pooling are max pooling and average pooling, where the output is the maximum and average value within the pooling window, respectively. It is a useful way to boost the efficiency of the neural networks since the parameters are significantly reduced.

Architectures

Over the past decade, several popular CNN architectures were proposed. For instance, LeNet-5 [25] was first initiated, followed by AlexNet [26] in 2012 and VGGNet [27] in 2014. ResNet [28] was created in 2015 to resolve stagnated progress in deeper networks by introducing the residual block. In general, the arrangement of CNN architecture starts by several convolutional + pooling layers and ends with few fully connected layers, and we get the intuition from these masterpieces and design similar CNN models.

2.3 Gaussian Process

In addition to neural networks, we have used Gaussian process (GP) extensively in this thesis, ranging from pure Gaussian process regression to multi-fidelity Gaussian process modelling. Therefore, it is necessary to know how GP works.

2.3.1 From Gaussian distribution to Gaussian Process

We cannot start discussing Gaussian process without briefly mentioning Gaussian distribution.

Gaussian distribution, also known as normal distribution, is typically visualized as the bell

shape curve in our mind, where it can simply be represented using mean and variance. The mean μ determines the center of the bell curve, and variance σ^2 influences how widespread the curve is. The 1D form of its probability function is defined as:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{2.5}$$

If we intend to sample a point from the distribution, meaning performing an inference, we are going to get a point. Let us generalize the 1-D Gaussian distribution to n-D multivariate Gaussian distribution which is often defined as:

$$\mathbf{X} \sim N_n(\mu, \mathbf{\Sigma}) \tag{2.6}$$

where here $\mu \in \mathbb{R}^n$ and $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$ is a covariance matrix. Furthermore, μ and $\mathbf{\Sigma}$ are also defined as:

$$\mu = \mathbb{E}[X], \mathbf{\Sigma} = \mathbb{E}[(X - \mu)(X - \mu)] \tag{2.7}$$

We can get its expectation value simply from the mean and the correlation between X_i and X_j by looking at the matrix entry Σ_{ij}. In this case, if we sample it once, we will get n points, which is essentially a function. This brings us to one of the many statements of GP where it is a non-parametric model that is used to represent the distribution of functions, as opposed to parametric models like neural networks. A non-parametric model like GP means that the data distribution cannot be represented using a finite set of parameters, so it uses distribution to sample the infinite number of parameters. GP is also defined as a collection of infinite random variables, any finite number of which have a joint Gaussian distribution [1]. Note the word "infinite", meaning the number of random variables is not fixed; therefore, this leads to the typical expression of a Gaussian process:

$$f(x) \sim GP(m(x), k(x, x')) \tag{2.8}$$

where, similar to a multivariate Gaussian distribution as describe in Equation 2.6, $m(x)$ refers to the mean function as well as the expectation value on each input x and $k(x, x')$ is commonly known as the kernel function.

2.3.2 Kernel functions

Let us introduce the concept of kernel functions in detail with the most popular one, radial basis function (RBF) kernel or squared exponential kernel, which is defined as:

$$k_{rbf}(x, x') = \sigma^2 exp(-\frac{||x - x'||^2}{2l^2}) \tag{2.9}$$

where σ^2 is the variance and l is the lengthscale. σ^2 determines the distance that the given point is away from the mean, and l describes how significant given point's influence on neighboring points. Both are free parameters, while the default ones provided by common packages are sufficient. One interesting thing to know is that x and x' refer to any arbitrary point, so what RBF does is that it smoothens the functions sampled by corresponding GP. Without the kernel, the curves can be quite noisy since there are no constraints on the sampled points, especially the neighboring ones. As we can see from Equation 2.9, when x and x' are close to each other, the covariance matrix entry would be close to σ^2. As two points are further apart, the value is getting closer to zero, meaning less interference between one another. In addition, we can also notice that RBF kernel is infinitely differentiable, which makes it extremely smooth. The choice of a kernel is extremely important, since, at the end of the day, we are trying to find the function $f(\mathbf{X})$ that best represents the correlation between input and output features. RBF has been proven to be robust and reliable in common machine learning problems, so it will be used throughout this thesis [1].

In addition to RBF, we have run some small experiments regarding the influence of kernels at the end of the thesis, and we are going to briefly cover them here.

- **Linear kernel**: Linear kernel is non-stationary, so absolute location of the inputs can influence the covariance function. The formula of linear kernel is defined as:

  $$k_{Linear}(x, x') = \sigma_b^2 + \sigma^2(x - c)(x' - c)$$

  where σ_b^2 is a constant variance which adds a height to the function, and c is a offset that all the functions sampled by posterior have to go through [29]. When we choose to use linear kernel in GP, we are essentially using Bayesian linear regression, and its computational complexity is O(n).

- **Matérn kernel**: Since RBF kernel is such as success, so we think it might be useful to try its variant. Matérn kernel is stationary, and it is a generalization of RBF kernel. It generalizes RBF as well as many others by adding a parameter ν. Its formula is defined as:

  $$k_{matérn}(x, x') = \frac{1}{\Gamma(\nu)2^{\nu-1}} \left(\frac{\sqrt{2\nu}}{l} d(x, x') \right)^{\nu} K_{\nu} \left(\frac{\sqrt{2\nu}}{l} d(x, x') \right)$$

where K_ν is the modified Bessel function, $d(x, x')$ refers to the Euclidean distance between x and x', and Γ is the gamma function. When $\nu = \frac{1}{2}$, the Matérn kernel becomes the absolute exponential kernel. Particularly, $\nu = \frac{3}{2}$ and $\nu = \frac{5}{2}$ are some of the most used kernel in physical science since the former one is at least once differentiable and the latter one is at least twice differentiable. Interestingly, it becomes RBF kernel when $\nu = \infty$, since we assume the function being sampled by RBF kernel to be infinitely differentiable. The flexibility of ν offers us a way to control the smoothness of the function according to the underlying system. We provide the results when $\nu = \frac{3}{2}$ (Matérn32) and $\nu = \frac{5}{2}$ (Matérn52) in a later section.

- **Rational quadratic kernel**: Rational quadratic kernel can also be seen as a variant of RBF kernel. It is an infinite sum of RBF kernel over different lengthscales. Therefore, the functions sampled by this kernel's prior might vary a lot in terms of their smoothness. Its mathematical formula is defined as:

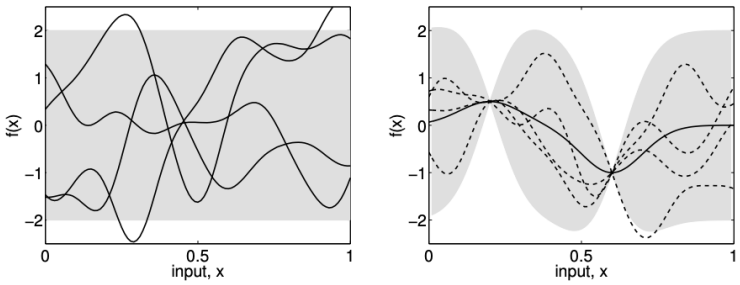
$$k_{RationalQuadratic}(x, x') = \left(1 + \frac{d(x, x')^2}{2\alpha l^2}\right)^{-\alpha}$$

where α is a scale mixture parameter and l is a lengthscale parameter. Interestingly, while α gets closer to ∞, this kernel would be the same as RBF kernel.

I recommend reading the kernel overview given by Dr. Duvenaud before diving into the problem [29]. Moreover, numerous researches are being done to find more suitable kernels for GP, while deep learning has been widely used to find specialized kernels from the training data [30].

2.3.3 Gaussian Process Regression

The idea of performing regression on the training data is basically a Bayesian inference problem [31]. The essential belief of Bayesian inference is that it will update the probability after receiving new information. In the case of GP regression, the new information refers to the training data. In this thesis, we start from making $m(x) = 0$ and $k(x, x') = k_{rbf}$ in Equation 2.8. We can understand kernel function as a guess on how the distributions should be like, or the prior. Before training, we can sample functions from this prior, while after training, GP can make predictions based on the prior and the measurements, which is also called posterior. Please kindly refers to Figure 2.9 for graphical representation. We can see that the sampled functions are fixed after receiving information at two different points.



(a), prior (b), posterior

Figure 2.9: (a) shows four functions sampled by prior distribution. (b) shows four functions (dash line) sampled by posterior distribution, while the solid line is the mean of these four functions. Gray zone represents the confidence interval. Figure is adapted from [1].

Here, we can formulate the joint distribution as the following:

$$P_{X,Y} = \begin{bmatrix} X \\ Y \end{bmatrix} \sim N(0, \Sigma) = N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \Sigma_{XX} & \Sigma_{XY} \\ \Sigma_{YX} & \Sigma_{YY} \end{bmatrix} \right) \tag{2.10}$$

where X is the training data, Y is the testing data and Σ is the covariance matrix or the kernel. Since our goal is to do inference with Gaussian process regression, let us move from joint distribution to conditional distribution:

$$f_* | X, Y, f \sim N(\mu_{pred}, \Sigma_{pred}) \tag{2.11}$$

where

$$\mu_{pred} = \Sigma(Y, X)\Sigma(X, X)^{-1} f$$
$$\Sigma_{pred} = \Sigma(Y, Y) - \Sigma(Y, X)\Sigma(X, X)^{-1}\Sigma(X, Y)$$

and f, f_* are the training and testing value, respectively [1]. From Equation 2.11, we can tell that inference with GP regression is simply a condition of posterior on the training data. This operation involves an inverse of the covariance matrix, which requires $O(n^3)$ to compute using Cholesky decomposition. While there are other methods to speed up the computation [1], since the size of the training data is generally huge which results in a gigantic matrix, this has been a bottleneck for GP to learn from a massive amount of data. However, a big advantage of GP regression is its natural ability to provide uncertainties, Σ_{pred}, to its predictions. As illustrated in Figure 2.9, the wider the gray zone is, the less confident GP is on its prediction at that point. This can be seen from a large Σ_{pred} that is generated.

2.3.4 Training a GP model

As shown in previous subsections, there are several hyperparameters that can affect the performance of the GP model. Therefore, choosing a suitable set of hyperparameters that can generate more representative functions becomes absolutely necessary. This is typically being done by maximizing the log likelihood of the result of the training data with respect to each hyperparameter. Similar to optimizing artificial neural networks, we tend to use gradient-based algorithms to find the minimum [32]. In reality, we generally restart the optimization algorithms for a couple of times in order to spread out the initial combinations in the space, reducing the risk of being trapped in an undesirable local minimum.

Automatic relevance determination (ARD) is a well-known and useful way to further boost GP's performance, especially in a multidimensional input setting. The idea is that it adds a weight parameter w_i to each input dimension in Equation 2.9. The possible scenario is that some dimensions are more important than others, which can be seen on the value of its weight. However, the introduction of new hyperparameter in each dimension can drastically increase the training time, since the searching space is growing exponentially. This is something which is worth considering before applying ARD to the algorithm.

2.4 Multi-fidelity modeling

As we covered the basics of Gaussian process, now we are using it as a building block to explain multi-fidelity (MF) modelling which we have used extensively in this thesis. In modern scientific or engineering problems, it is inevitable for our decision to rely on real-world experiments or measurements. In such areas, data sources usually vary in quantity or quality, such as measurement errors or simplification of computer simulations. Different decisions regarding the adoption of the data sources could be made based on the resources at hand and the goal that we are trying to achieve. This is where multi-fidelity modelling comes into play. MF was developed to take advantage of the correctness of the high-fidelity data as well as the abundance of the low-fidelity data. In this section, we will discuss the evolution of multi-fidelity algorithms. We start from the definition of MF models and moved on to linear and non-linear models, followed by a small example to demonstrate its concepts.

2.4.1 Multi-fidelity models

Multi-fidelity models, as its name suggests, consists of several models consuming data from different fidelity sources. In fact, each individual model itself does not have to be a GP regression model. For example, the model can either be a simple function taking average across inputs $\sum_{i=1}^{n} \frac{x_i}{n}$ or a complex deep neural network. Lots of flexibility are offered here depending on the use cases. Let us simplify the models into low and high fidelities

ones. Being a low fidelity model means the evaluation workload is much less than evaluating a high fidelity one. Although high fidelity model is either more computationally expensive, more time-consuming or simply costing more, we often assume the result f_{high} to be comparatively similar to the exact underlying function f_{exact}, $f_{high} \approx f_{exact}$, while f_{low} often gives terrible approximation if being used alone. Figure 2.10 provides a great visual overview to their fundamental differences in terms of error and cost. Basically, most of the models lie close to the line $error + costs = constant$. Note that normally there can be several low-fidelity models to choose from according to the needs.

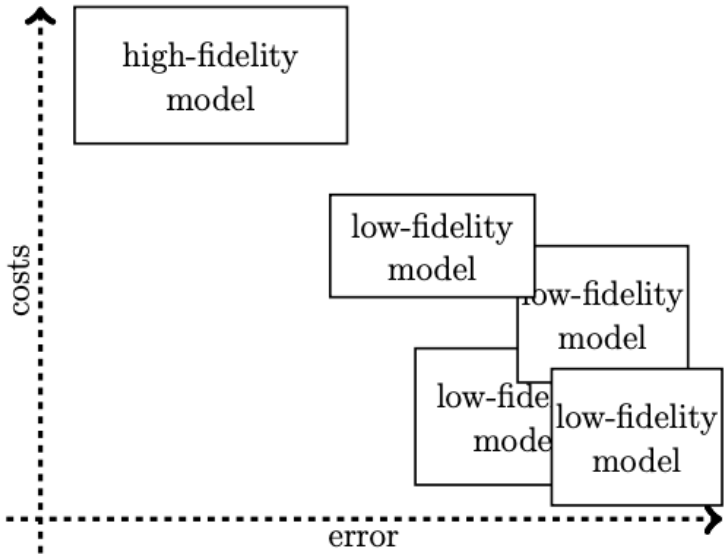


Figure 2.10: Error and cost comparison of high and low fidelity models. Figure from [33].

2.4.2 Auto-Regressive models (AR1)

Auto-Regressive models (AR1) is the most classical multi-fidelity algorithms, and it has been used extensively in various engineering design problems [34]. It is simplistic yet provides a foundation to all the methods in the future. AR1 is a linear information fusing framework proposed by Kennedy and O'Hagan [35]. They assume the models of different fidelities to be linearly dependent on each other, which can be formulated as the following:

$$f_t(x) = \rho_{t-1}(x) f_{t-1}(x) + \gamma_t(x) \tag{2.12}$$

where $f_t(x)$ and $f_{t-1}(x)$ refer to high and low fidelity models, respectively. $\gamma_t(x)$ is a Gaussian process prior assigned to each level, and $\rho_{t-1}(x)$ is the scaling factor that links

different models together. Since it is typically a constant, we can further re-write it to $\rho_{t-1}(x) = \rho_{t-1} \in \mathbb{R}$. We also call $\gamma_t(x)$ bias function which is a GP with trainable hyperparameters θ.

Figure 2.11 provides a graphical overview of this recursive process. When it comes to inference, the lowest fidelity input has to be passed in $f_1(x)$ and follow Equation 2.12. The result would be fused with higher fidelity models in the next levels. The prediction is given by the last model $f_s(x)$.

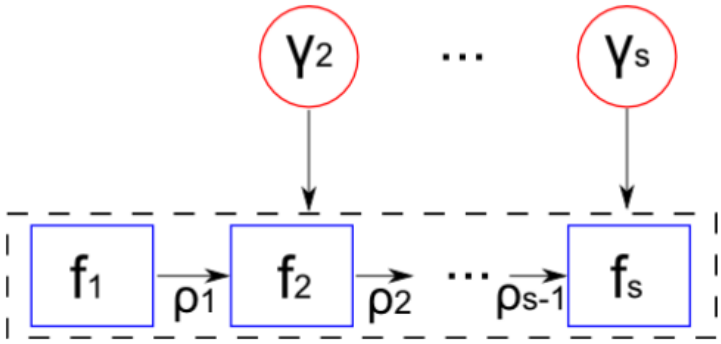


Figure 2.11: AR1 Schematic overview cropped from [34].

In a MF setting, AR1 shows that it should be used instead of plain GP models in order to take advantage of our prior knowledge to the data sources. Despite the success, AR1 has been reported to be limited when it comes to more complicated problems [34]. We will move on to the generalization of AR1 in the next section.

2.4.3 Nonlinear autoregressive Gaussian processes (NARGP)

Similar to the reason people introduce non-linearity to the neural networks, nonlinear autoregressive Gaussian processes (NARGP) was also introduced to tackle more complex relationships between fidelities [36]. NARGP is a framework defining how we should interconnect different data sources together. Its concepts can be seen in the following equation:

$$f_t(x) = g_t(x, f^*_{t-1}(x)) \tag{2.13}$$

where $g_t \sim GP(f_t|0, k_t((x, f^*_{t-1}(x)), (x', f^*_{t-1}(x'))))$. The scaling factor ρ_{t-1} and the GP prior $\gamma_t(x)$ in AR1 can be implicitly included by g_t in special case. As we can see from Equation 2.13, NARGP defines a mapping of $\mathbb{R}^{f_{t-1}+1} \to \mathbb{R}$. In simple words, it means that NARGP collects one output from previous level, adds any given input in current level and

produces one output as an input to the next fidelity model. We can think of it as a way to consolidate the information in lower fidelity model and only provide useful patterns to higher fidelity models.

Let us illustrate NARGP further by a simple example. In this example, high fidelity function is defined as:

$$f_{exact}(x) = f_{high}(x) = (x - \sqrt{2})sin^2(8\pi x)$$

, and low fidelity function is defined as:

$$f_{low}(x) = sin(8\pi x)$$

We can tell that f_{low} has a certain level of connection to f_{exact} but not quite close. Figure 2.12 shows the prediction where only 12 points (blue) are provided to the GP model. Since given inputs gather mostly on the right side, the prediction is a bit better than left side, but it is still far from perfect. The confidence interval is huge throughout all x, and the posterior mean is only close to f_{exact} in the area where data are denser. One interesting discovery is that between the first and third points, the expectation value is mostly unchanged, *i.e. 0*, indicating that the GP does not even modify the values due to the remoteness to the nearest known points.

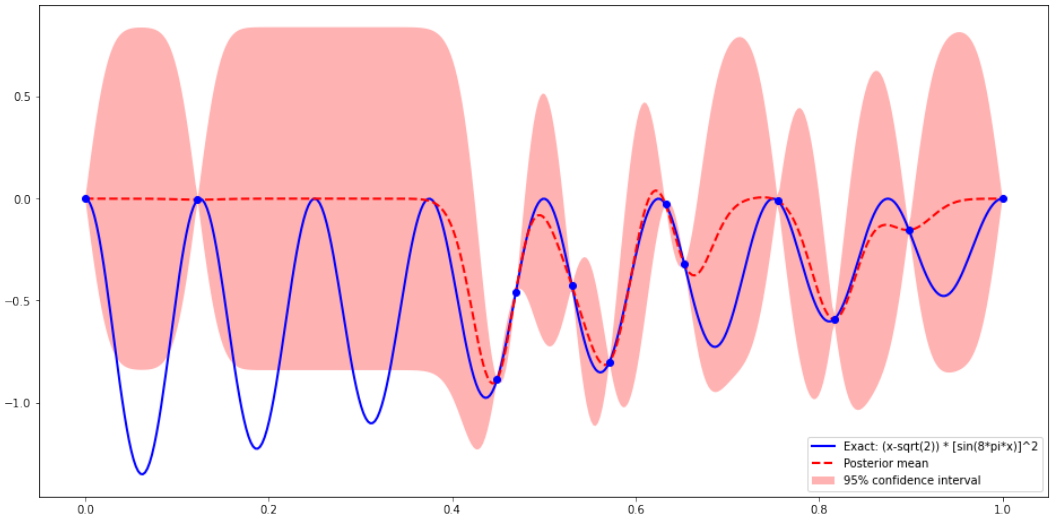


Figure 2.12: Pure GP result.

As a comparison, we provide 50 more points from low-fidelity model and fuse with high-fidelity model using NARGP framework. The result is shown in Figure 2.13. The extra

50 points are evenly spread across the input domain. The graph demonstrates that the addition of data, despite being low fidelity, can boost the performance of the model significantly. The uncertainty is drastically reduced and accurately overlaps with f_{exact}. With NARGP, both Mean Square Error (MSE) and Mean Absolute Error (MAE) decrease from 0.237 to 0.0005 and from 0.311 to 0.013, respectively.

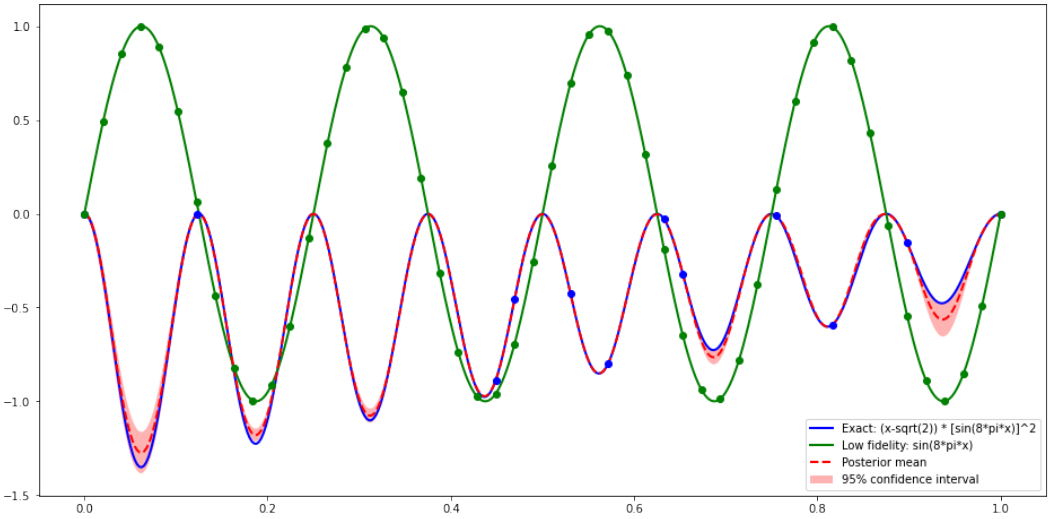
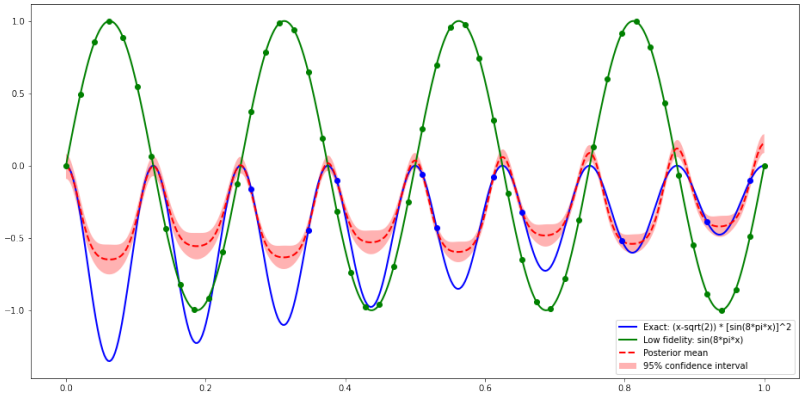


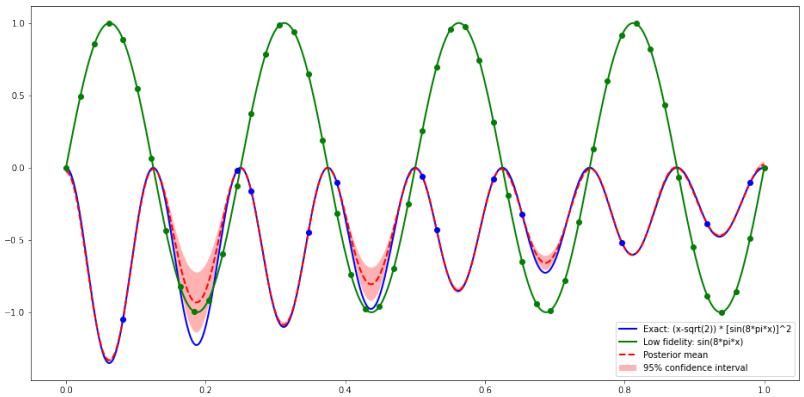
Figure 2.13: NARGP result.

The influence of the number of high fidelity points

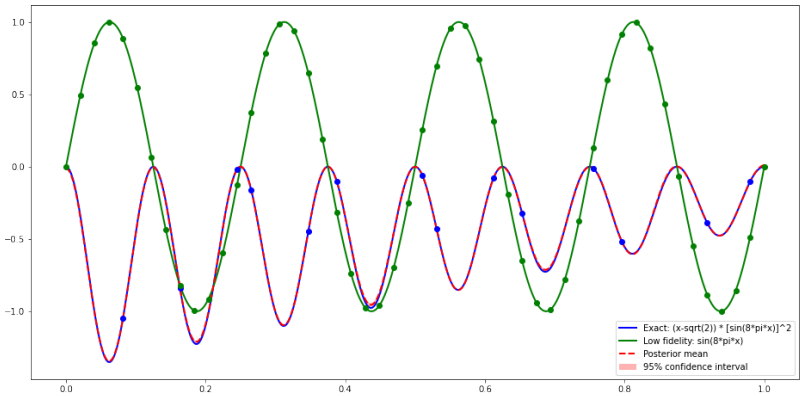
In previous subsection, we saw that with NARGP framework, the model can learn from the loosely related low fidelity data points and increase the performance of the model drastically. Here, we discuss how the number of high fidelity data points can affect the effectiveness of NARGP. Since we assume the evaluation cost for high fidelity data is much higher than low fidelity data, the ideal case would be to have acceptable result with the least number of high fidelity data. In Figure 2.14a, we use only 10 high fidelity points, and we can see that the prediction is far from ideal, especially in the regions where there are scarce numbers of high fidelity points. When we add two more points, as shown in Figure 2.14b, the prediction is much better than before. Interestingly, the two new points are added in front, but they are able to improve the predictions in the regions which are far away from them. While 14 high points are used, the prediction is almost the same as f_{exact} (2.14c). This is where we know that there is no need to further increase the number of high fidelity points, since the accuracy gain is reaching a plateau. In summary, we conclude the evaluation metrics in Table 2.1 where both MSE and MAE are approximately improved by an order of 10 whenever 2 more high fidelity points are appended.



(a) NARGP result using 10 high fidelity and 50 low fidelity points.



(b) NARGP result using 12 high fidelity and 50 low fidelity points.



(c) NARGP result using 14 high fidelity and 50 low fidelity points.

Figure 2.14: The results of 10, 12 and 14 high fidelity data points using NARGP framework with 50 low fidelity samples.

Table 2.1: Number of high fidelity points and their corresponding MSE and MAE

Evaluation metrics	10 (2.14a)	12 (2.14b)	14 (2.14c)
MSE	0.0519	0.0046	$3.86 * 10^{-5}$
MAE	0.1415	0.0315	0.004

2.5 Uncertainties in machine learning

In the last section of background theory, we are going to briefly explain the uncertainties in machine learning methods. There are two types of uncertainties, aleatoric and epistemic uncertainties. The former one refers to the natural and irreducible uncertainty from either the data itself or the system. In short, this type of uncertainty is unavoidable. Therefore, the type of uncertainty we are going to cover is epistemic. It represents the ignorance and reducible part of the uncertainty, which is the part we strive to reduce when it comes to making prediction in machine learning. Specifically, this is the type of uncertainty that comes with the prediction in Gaussian process regression.

In Equation 2.10, we show that Σ is the covariance matrix of the distribution. However, in many real-world applications, the observed data can be noisy, so we tend to add a noisy Gaussian noise to the observed ground truth:

$$\hat{Y}_i = Y_i + \epsilon_i$$

where the noise to defined as $\epsilon_i \sim N(0, \sigma^2)$. In this case, the updated covariance matrix becomes $\hat{\Sigma} = \Sigma + \sigma^2\mathbf{I}$. It can be inserted into Equation 2.11 and get updated $\hat{\Sigma}_{pred}$ which is denoted as :

$$\hat{\Sigma}_{pred} = \Sigma(Y, Y) + \sigma^2\mathbf{I} - \Sigma(Y, X)(\Sigma(X, X) + \sigma^2\mathbf{I})^{-1}\Sigma(X, Y)$$

For reference, now the predicted mean becomes $\Sigma(Y, X)(\Sigma(X, X) + \sigma^2\mathbf{I})^{-1}f$. This small tune can make the computation more numerically stable since the matrix $\hat{\Sigma}_{pred}$ is always invertible even though we do not really inverse the matrix to obtain the results. Therefore, for the prediction, we can have the predictive variance at given points for free. With the variance, we can calculate standard deviation as well as the confidence interval to be able to quantify our predictions.

3 Data-driven multi-fidelity modeling for terramechanics

3.1 Motivation and Overview

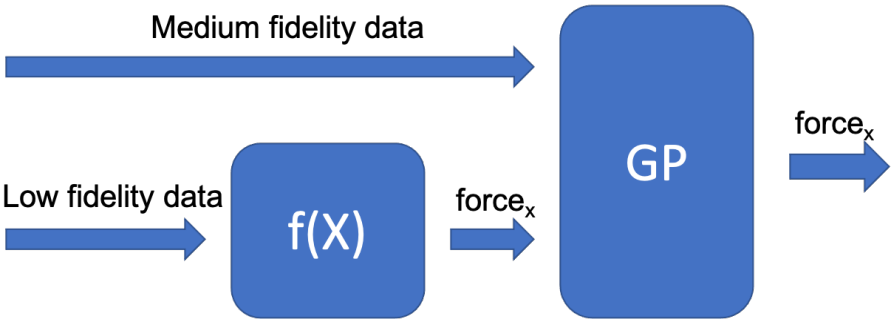


Figure 3.1: General overview of the flow of multi fidelity modeling.

Before we enter the main part of the thesis, we are going to spend a short paragraph explaining the goals, a brief overview about the data, the models we use as well as the data flow.

Long story short, the goal is to accurately predict the force of the mobile wheeled robot with the data coming from two different fidelity sources, and the way we realize the goal is to build multi-fidelity models linked by Gaussian processes. We have two set of data describing the same movement of a robot, but they are simulated in different fidelities. Both of the data sources possess features including force, torque, velocity, position, angular velocity and gravity that the robot experiences. Furthermore, the surface images under the robot are also used to aid the judgement of the models. We decided to let the models infer force from the rest of the features.

In Figure 3.1, we present a schematic overview of a common nonlinear multi fidelity model. We generally feed low fidelity data to a $f(X)$ with relatively low evaluation cost. It can range from a simple average function to a neural network model. In the thesis, we have employed MLP, GP and CNN as our choices for $f(X)$, and we train this $f(X) = force_x$ to the best of its ability. In second layer, a GP is used to merge medium fidelity data as well as the results coming from $f(X)$. This GP is thus trained with information from multi

fidelity sources, and the result it produces is the final prediction for $force_x$. In addition to the multi fidelity settings, we have also built an MLP model as well as GP model to compare their results.

In the coming sections, we will first elaborate the data in greater details and then move on to the objective of the models. Before we dive into a variety of the models we use, some evaluation metrics will be discussed to better understand further analysis. Six different models will be introduced later in the sections and compared with one another about their pros and cons. Last but not least, we experiment several promising kernel candidates, review each one of them and provide directions for future endeavors.

3.2 Training data from the simulation of wheeled mobile robot

We produced the data by conducting experiments ourselves. According to the ways the data are generated and their reliability, we can categorize them into two different fidelities, and we call them low and medium fidelity data.

Low fidelity data is generated using TerRA methods [13]. It is a simulation model designed specifically for the movement of a wheeled robot on sandy terrain. Being an empirical model, both normal and longitudinal forces are calculated to describe the behavior. Normal force calculation includes an elasto-plastic model [37]. Plastic and dynamic sinkage computations are also taken into account for wheeled locomotion. The calculation of longitudinal force depends on the previous records of slip and velocity. Despite being a low fidelity source, it is proven to require low computational effort; thus, TerRA is an ideal candidate for on-board applications.

Medium fidelity data is produced by soil contact model (SCM) with improved algorithm on soil deformation [14]. SCM is used to model a multi-body system which is composed of two steps, contact dynamics and soil update steps. In order to better describe the soil flow, modified SCM introduced three independent parameters and improved the soil deformation calculation in the original model. The data here is more reliable and hence is used as the "pinned points" for our multi-fidelity models. Because of its higher computational cost, there are fewer data points being used in our modelling.

In the following, we will explain the shape and the properties of both data sources. Regardless of their fidelities, we can get 101 records, namely 101 timestamps, every time we perform the simulation. Within the time window, we can simulate a wide variety of behaviors of the rover and collect relevant data. For each record, there are 4114 columns of features, which include force, torque, coordinate, velocity, angular velocity and gravity in x, y, z directions and a 64*64 pixels image. All the values mentioned above are numeric. Figure 3.2 depicts some surface images being scanned by wheeled mobile robot when performing simulations, which are used as extra information when constructing the models.

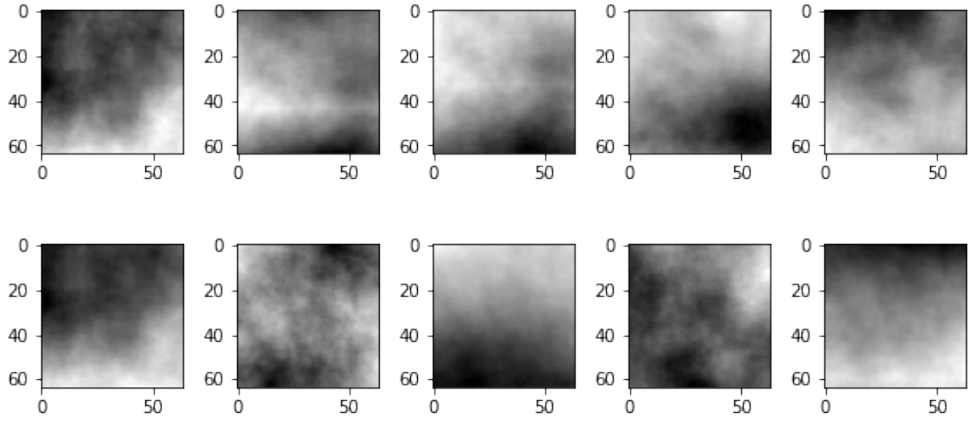


Figure 3.2: Example of ten gray-scale surface geometry images.

For each fidelity, we have conducted 100 simulations, with each one giving us 101 records. Thus, we have 10100 data points to train our models. In theory, we can produce as many results as we can, but in order to keep the experiments simple and to have reasonable training time, we decide to keep the number of samples at around 10k before feeding in more data and create more sophisticated models. For all the experiments listed in this chapter, we choose 5000 low fidelities samples randomly as the training data. As for multi-fidelity settings, 1000 corresponding medium fidelities samples out of the 5000 chosen low fidelities entries are selected. In addition to the training data, 1000 samples are picked for testing the model performances.

3.3 Prediction problem: Forces and Torques

As mentioned in previous section, except for surface images, there are 18 features being generated from the simulations, namely force, torque, coordinate, velocity, angular velocity and gravity in x, y, z directions. For instance, here we use $force_x$ to denote force in x direction. Firstly, we can categorize the features into two groups, active feature and passive feature. Active features refer to force and torque. These are the characteristics that are actively controlled by the mobile wheeled robot. Therefore, these are the features that we would like to predict. Passive features include coordinate, velocity, angular velocity, gravity and surface image. More elements like temperature, pressure and wind speed also fall into this category. These are the features that are measured by external sensors. We treat these features as potential inputs to our models. In the future but not within the scope of this thesis, the ideal scenario is that the mobile wheeled robot can detect the surrounding environment including arbitrary passive features, feed the input data stream to the model, and make reasonable decision by controlling its active features.

In order to make the model as simple as possible, we filtered out $coordinate_y$, $angular\ velocity_x$, $angular\ velocity_z$ and $gravity_y$ since they remain constant or nearly identical throughout the whole simulation. This benefits us not only from the model simplicity point of view, but it also reduces the training time significantly, since less parameter are being optimized during the training.

At the end, we choose $coordinate_x$,$coordinate_z$, $velocity_x$, $velocity_y$, $velocity_z$, $angular\ velocity_y$, $gravity_x$ and $gravity_z$ as our input features. For the feature being predicted, we use $force_x$ as our preliminary target. Ultimately, we are trying to find the models that best fit $f(X) = Y$ where the length of X vector is 8 and the length of Y vector is 1.

In Figure 3.3 and 3.4, we can get a glimpse on how features correlate with one another in both dataset. There are 9 features, with the first eight being the input features and the last one being the output feature. We can obviously observe some interesting trends between features, and we are hoping that the models can learn and extract useful patterns for the predictions.

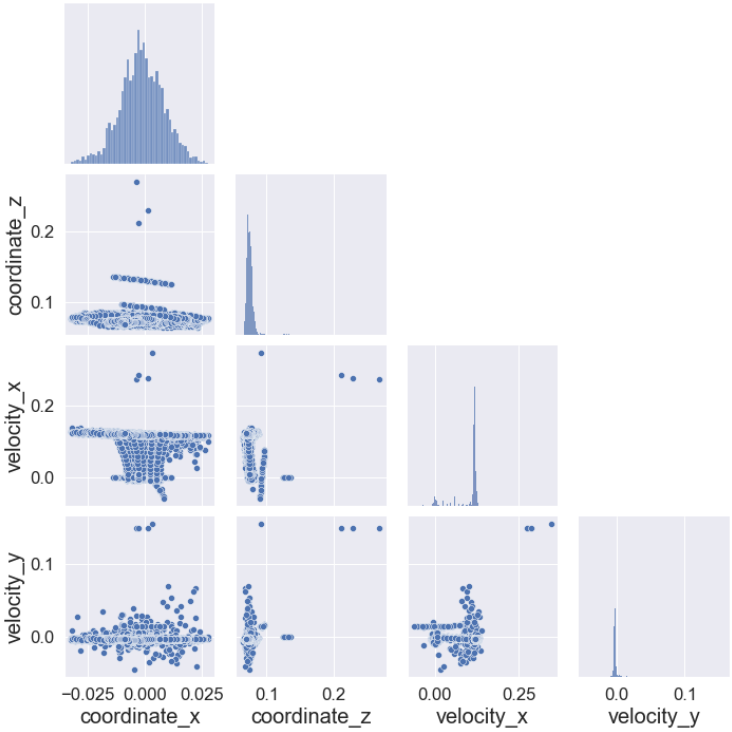
3.4 Evaluation metrics

In this thesis, we have employed several metrics to evaluate the performance of different models. The metrics are further explained in the following:

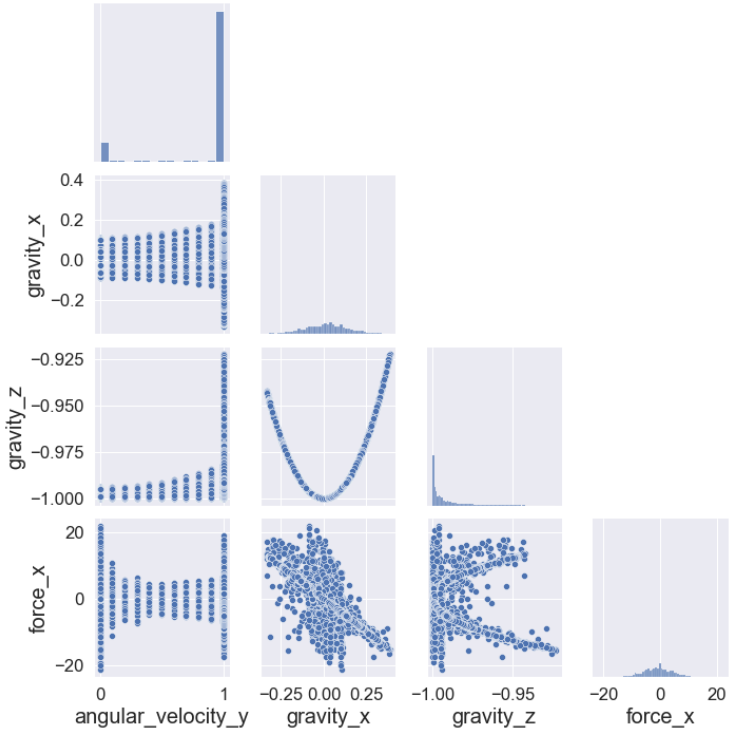
- **Time**: When it comes to training a machine learning model, time is obviously one of the most important metrics, since less training time means we can perform more iterations of experiments within a given time frame. It is even more important when the amount of data is huge. We want the model to learn as many as possible from the data under the condition that the training time is reasonable. While we pay lots of attention to the training time of neural networks, we generally discuss less about the computing time it takes to train a Gaussian process model. This is due to the fact that cubic computational complexity has been a bottleneck for GP to be used extensively in big data applications. For all the computing tasks in this thesis, we are using a personal laptop with 2.7 GHz Dual-Core Intel Core i5 processor and 8 GB RAM.

  There are plenty of ways to measure the time it takes to perform a task. Different profiling tools and methods come in handy in different scales. The code snippet shown in Code 3.1 demonstrates how we calculate the training time for a model. We take a straightforward approach where the training time is calculated by subtracting start timestamp from end timestamp. For each experiment, we perform it for several times and take an average to mitigate the CPU performance fluctuations. Since the order of the training time is around the range of hours, individual instructions or microseconds differences are negligible.

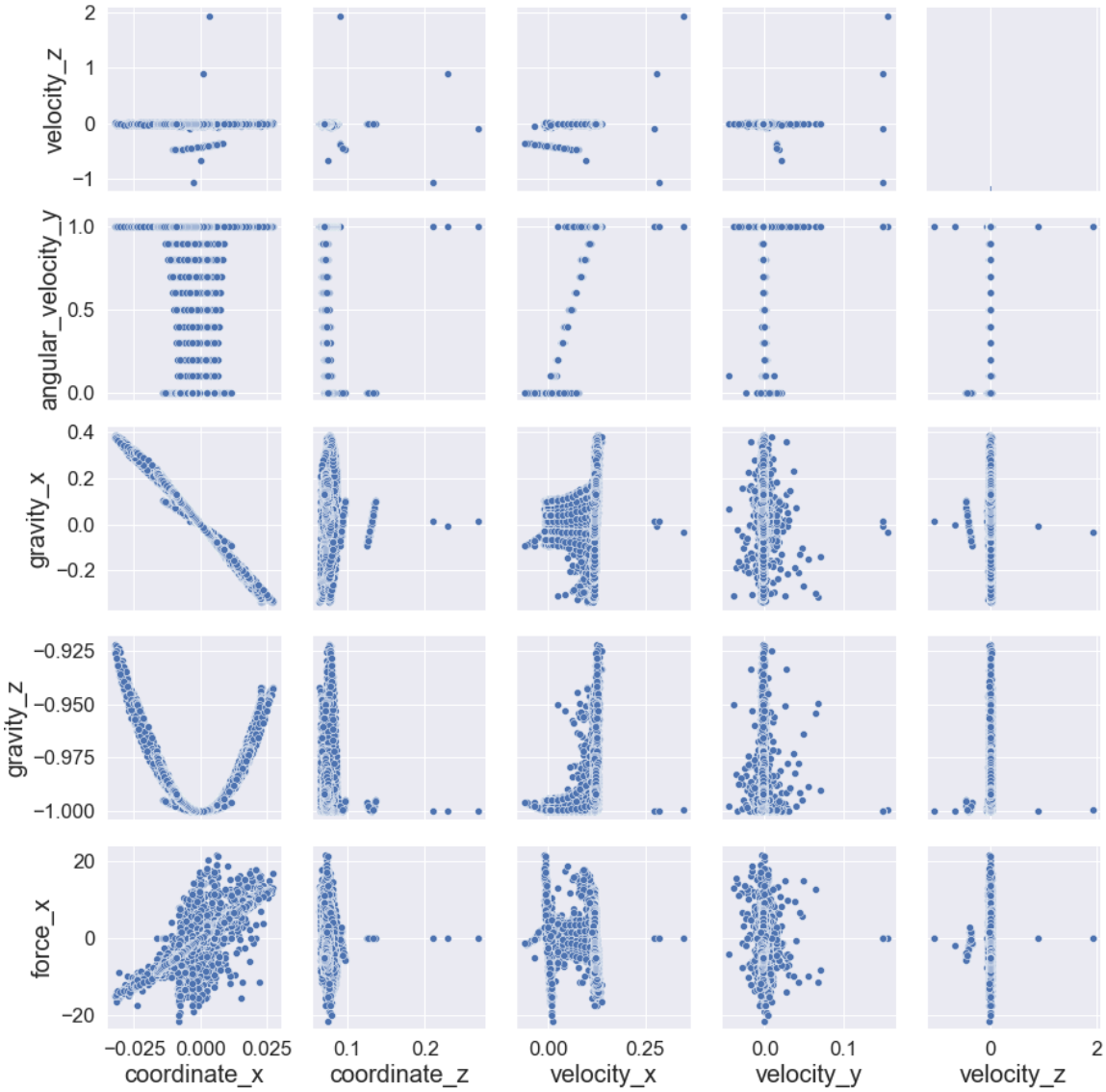
- **Mean Absolute Error (MAE)**: The formula of MAE is defined as:



(a) Part of the pairwise plots in the low fidelity dataset.

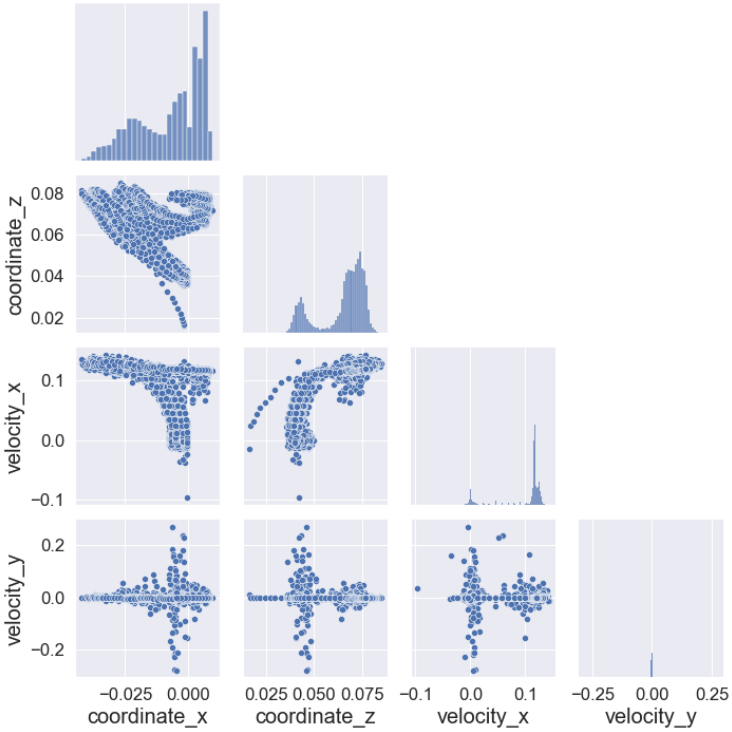


(b) Part of the pairwise plots in the low fidelity dataset.

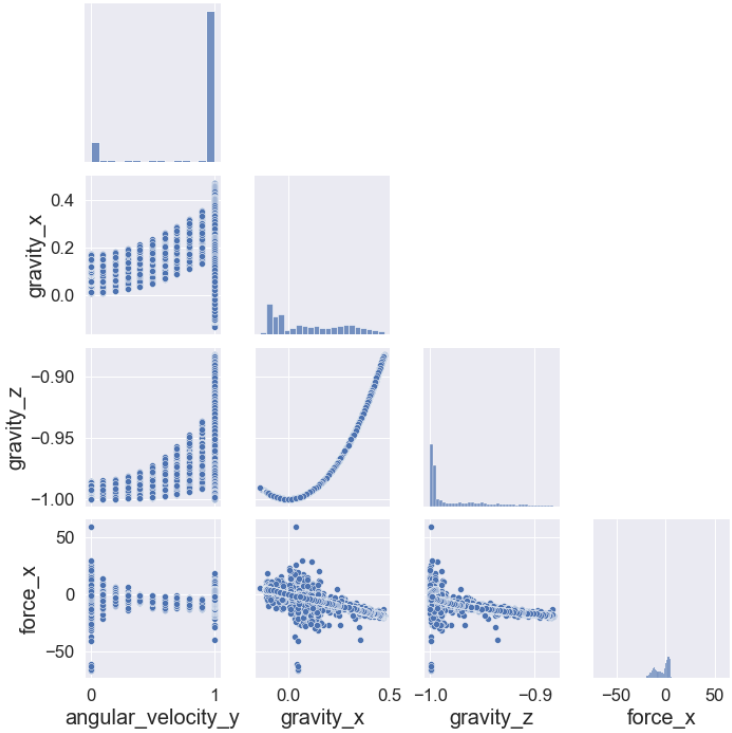


(c) Part of the pairwise plots in the low fidelity dataset.

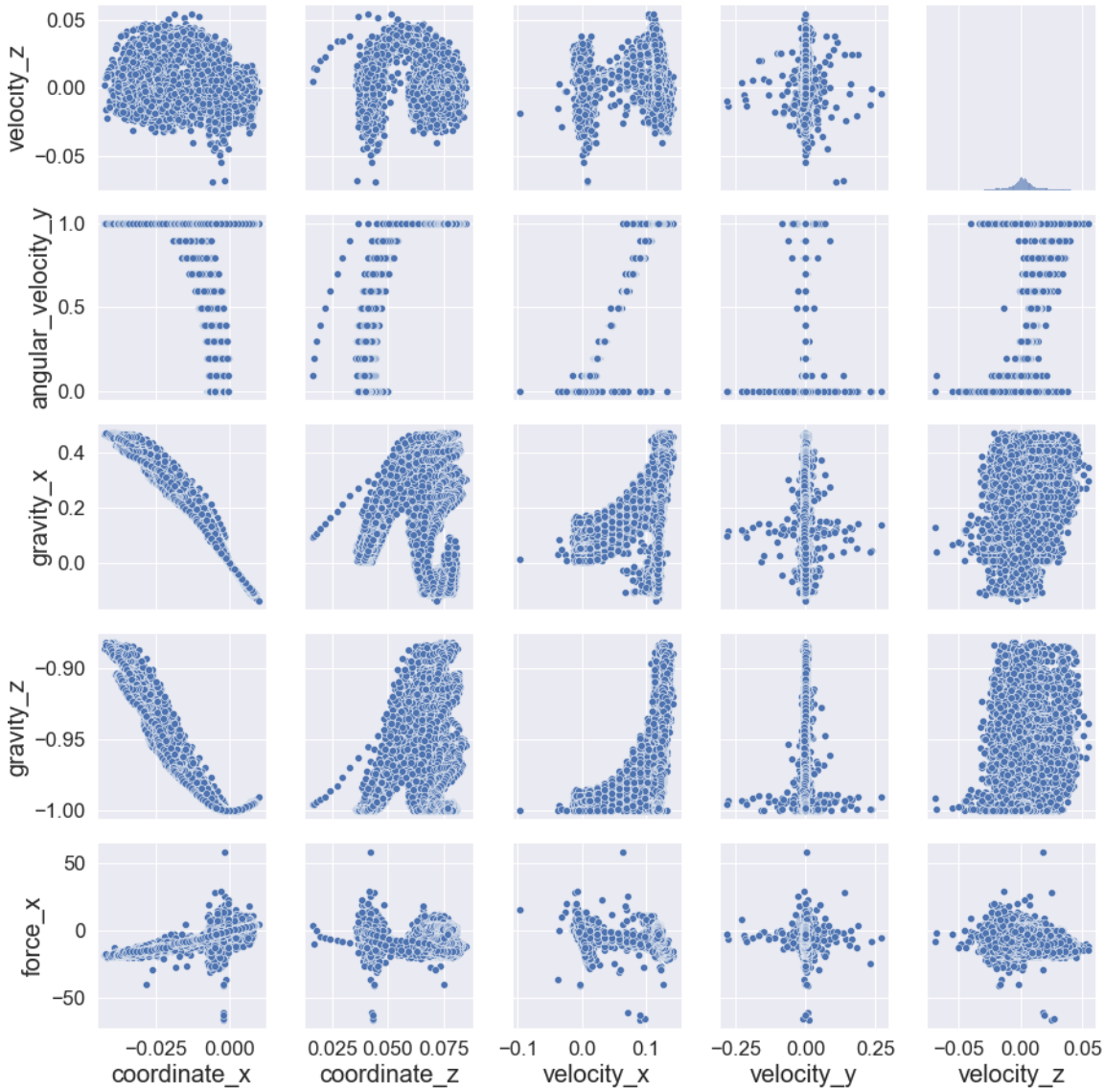
Figure 3.3: Pairwise relationships of low fidelity dataset. If the axes are the same, a histogram is shown; if not, a scatter plot is shown with each point being a sample.



(a) Part of the pairwise plots in the medium fidelity dataset.



(b) Part of the pairwise plots in the medium fidelity dataset.



(c) Part of the pairwise plots in the low fidelity dataset.

Figure 3.4: Pairwise relationships of medium fidelity dataset. If the axes are the same, a histogram is shown; if not, a scatter plot is shown with each point being a sample.

```
import time

start_time = time.time()

# This is where the training is performed

end_time = time.time()
training_time = end_time - start_time
```

Source Code 3.1: The way we measure the model training time in Python.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

where n is the number of training samples, y_i is the predicted output vector (length is one in our case), $\hat{y}_i$ is the ground truth of the output vector. MAE is one of the most common ways to measure the errors of the observations. We use it to assist the judgements of other metrics.

- **Mean Squared Error (MSE)**:The formula of MSE is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}||y_i - \hat{y}_i||_2^2$$

where n is the number of training samples, y_i is the predicted output vector (length is one in our case), $\hat{y}_i$ is the ground truth of the output vector. MSE is so widely used, which makes it the default metrics for any measurement. It penalizes the predictions which are too far away from the ground truth. This characteristic makes it a better candidate to evaluate the performance of GP related models, since, overall, we want the ground truth to be as close to the predicted mean as possible. It is ideal for the ground truth to lie within the confidence interval, while a value situated several standard deviations away from the mean cannot be tolerated. The fact that MSE imposes penalty to distant points tends to provide more desirable GP models.

- **Certainty of Prediction (CoP)**: In order to quantify the certainty of predictions, we create CoP to help us compare the accuracy of two different models. Please refer to Figure 3.5 for graphical representation. Since we have to standardize each predicted value in order to compare between models, the range of CoP is shown as the following:

$$0 \leq CoP \leq 1$$

CoP equals 1 when the mean exactly matches the ground truth, and it is zero when the predicted mean is infinitely far away from the ground truth.

In our case, there are 1000 testing points, so this gives us a thousand 1-D standard normal distribution graphs just like Figure 3.5, which also provides us 1000 distinct CoP values. We compare the model performance by averaging the CoP across these 1000 testing points.

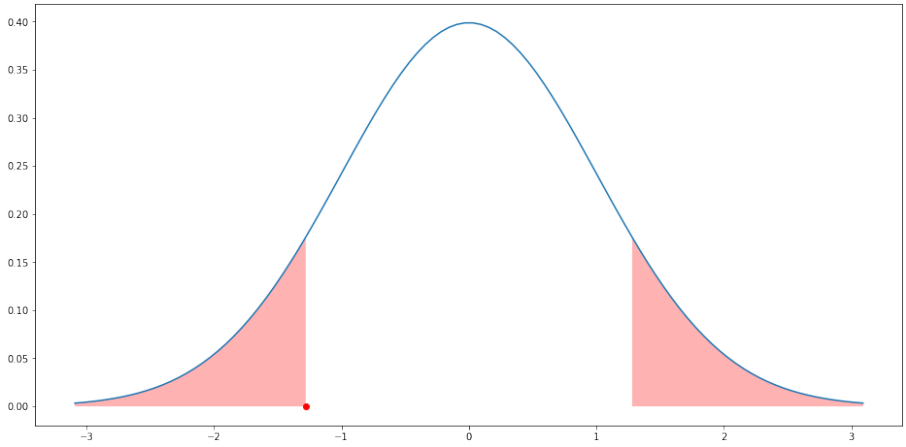


Figure 3.5: Blue line shows a curve of a 1-D standard normal distribution. If we are given a red point, then the red region is the Certainty of Prediction (CoP) and is symmetrical with respect to y-axis. The graph is plotted by connecting 1000 equidistant points between -3 and 3.

3.5 Computational experiments

We are entering the most interesting part of the thesis where a variety of the models are used to best find the relationships between inputs and outputs. First of all, we address the data, architectures and hyperparameters preparation that are shared by all the models. This paves the foundation to all the upcoming computational experiments. The result sections start from a simple multilayered perceptron (MLP) model which is simple, quick to compute and can provide a benchmark to all the other models. Then, we move on to GPR and feed it with medium fidelity data only. As for the cases to fuse multi-fidelity data sources, we first use a linear autoregressive model, followed by a more sophisticated non-linear model. We also tried incorporating MLP into the NARGP framework. In the end, we provide extra surface geometry information and use CNN to capture the pattern, which

is also fused using NARGP. Before the end of the section, we conduct extra experiments where multiple potential kernel candidates are employed to compare against the default RBF kernel.

3.5.1 Experiment preparation

To be able to compare the performance between models and eliminate as many variables as possible, we try to standardize the experiment process as far as possible, and we are going to conclude our approach in this section.

Let us first share how we split the data into the training and testing set. Before revealing the approach, we have to rewind back to the nature of the datasets. The underlying meaning of both TerRA and SCM is a trajectory of the movement of the mobile wheeled robot. This means that the characteristics of the recorded data are continuous. For example, the first thousand samples might represent a robot moving forward, possibly followed by another hundred samples describing the process of acceleration, turning, ascending or descending on the simulated surface.

Therefore, it does not make sense to slice the dataset sequentially with the order train and test, since it is extremely hard for the model to predict the operations that are unseen. With this mindset in mind, for both low and medium fidelity datasets, since they are both simulating exactly the same movement, we first randomly pick 5000 indexes out of all the samples and get the low fidelity training set by matching these indexes to the low fidelity samples. The testing set of 1000 are randomly chosen from the rest of the samples. As for medium fidelity samples, it is selected by randomly picking 1000 indexes out of the 5000 low fidelity indexes and match them to the medium fidelity samples. Our approach most probably ensures that the models can learn from a variety of movements within the scope of the simulation. For the surface geometry information, there are 80901 continuous images depicting the ups and downs of the surface. Among these, we use 80%, 10%, 10% for training, validation and testing a CNN model. In addition, we conduct all the experiments by fixing the random seed in the dataset splitting process, although different set of samples can inevitably lead to slightly different results.

We use Python as the programming language of choice since it is the most dominant and widely used language in the domain of data science. For simplicity purpose, we use scikit-learn when we want to deal with the construction of dense neural networks [38]. When it comes to GP-related models, we use a framework developed by Sheffield machine learning group, GPy [39]. It is extremely useful and versatile, allowing us to create any combination of models. Last but not least, Pytorch is used to build the data pipeline and CNN model [40]. Despite its fame in the support of GPU acceleration, we choose it mainly due to its close integration with native Python.

In the next few subsections, several computational experiments will be introduced with

detailed explanations.

3.5.2 (1) Multilayered perceptron (MLP) model

For the first computational experiment, we intend to adopt a brute force approach by using a dense interconnected neural network. We choose only single layer neural network with 100 nodes, and we train the parameters with SCM, namely medium fidelity data only. Along with the input and output layers, the model has an architecture of 8-100-1 nodes. Table 3.1 shows the hyperparameters we employed regarding the MLP model.

Table 3.1: The hyperparameters used in (1) MLP model

Activation function	Relu
Batch size	32
Learning rate	0.001
Optimizer	Adam
Epoch	500

Since the focus of the thesis is to leverage multi fidelity sources, we did not spend too much effort optimizing the hyperparameters of the neural network, so there is room for improvement if wanted.

Table 3.2 shows the results of the testing set we got from the MLP model.

Table 3.2: The results of (1) MLP model. Format: average $\pm$ standard deviation

MSE	12.872 ± 0.062
MAE	1.127 ± 0.023
Training time (s)	0.52 ± 0.085

The way we calculate the average and the standard deviation here is by fixing the random seed for the dataset and re-initializing the weights of the neural network for five times. Each time we start the optimization with a different set of initial parameters is like randomly choosing a starting point in the space spanned over 8 dimensions, aka. the number of input features. The standard deviation of both MSE and MAE are quite small, suggesting that the neural network is consistently finding local minimums similar in values. In addition, MLP does not have CoP since the way neural works is by conducting basic mathematical operations with trainable weights and biases, so the output is a predicted value of the feature without any uncertainty estimate.

3.5.3 (2) Gaussian Process Regression model

In this section, we are presenting a pure GPR model trained solely by medium fidelity data, as if the low fidelity data does not exist. We do this because it is valuable to compare the differences with the introduction of abundant low fidelity data in multi fidelity setting in future sections. As mentioned, we choose RBF kernel as the estimated shape of the function and enable ARD in order to evaluate each dimension independently. The optimization process of the model is restarted for 20 times to ensure sufficient coverage of the search space. In addition, *bfgs* algorithm is used to find the best set of parameters. It is a qausi-newton method well known for numerical optimization when the number of training data is relatively small. The results are also generally reported performing better than Adam optimizer since it takes curvature into consideration. Table 3.3 shows the results of the testing set we got from the GPR model.

Table 3.3: The results of (2) GPR model. Format: average $\pm$ standard deviation

MSE	14.448 ± 0.363
MAE	1.308 ± 0.015
CoP	0.637 ± 0.004
Training time (s)	178.288 ± 46.781

We can see that the performance is worse than (1) MLP model in any given metrics. This is expected since GP alone is not particular good at multidimensional setting, but it does shine with its ability to provide estimate to the uncertainties. Also, the training time is burdened by the number of restart times that we choose. It can be reduced with less restart times, but it might sacrifice the chances to find better set of parameters. Still, the training time difference is what hinders GP from real-time big data applications.

As a comparison, without ARD, the MSE, MAE and CoP are 15.828, 1.468 and 0.617, respectively. We can clearly notice a distinct decline in all the metrics. However, better performance comes with a cost where the training time gains by approximately 116%. This is a trade-off to be considered before applying ARD to the model. Since training efficiency is not a major concern in this work, we are happy to adapt this technique for the performance gain.

3.5.4 (3) Linear multi-fidelity model

Starting from this section, we are introducing low fidelity data to the models as another source of information. We decide to employ linear multi fidelity model as the first trial. For graphical representation, please refer to Figure 2.11. We can see from the figure as well as the Equation 2.12 that there are two GP models. One is trained using low fidelity data and the other is trained using medium fidelity data. The result of the low fidelity

GP is weighted by a parameter ρ and passed on to the next level. The final prediction is determined by the sum of weighted low fidelity result and the result from medium fidelity GP. However, since their results are linked linearly, the quality of the low fidelity data can greatly influence the final outcomes. Also, models linked linearly might hamper the ability to express complicated patterns.

We choose the package developed by Andrei *et al.*, Emukit [41], to tackle the construction of AR1 model. Emukit is a Python adaptable toolkit aiming to make the prototype of statistical methods on physical process easier. It helps us reduce the amount of work and the lines of code for this model. Due to the increasing computing time, we restart the optimization for five times instead of 20 times in single fidelity GP.

Table 3.4 demonstrates the results of the testing set we got from the AR1 model.

Table 3.4: The results of (3) linear multi-fidelity model. Format: average $\pm$ standard deviation

MSE	12.569 ± 0
MAE	2.002 ± 0
CoP	0.641 ± 0
Training time (s)	12995 ± 2124

The first thing we might notice from the table is the identical result throughout five experiments which lead to zero standard deviation. While we dig into the training log, it shows that for each optimization restart, we do get different losses for each set of parameters. This specific set of parameters gives rise to the minimum loss for all the experiments. The phenomenon indicates that this "local minimum" might be too easy to achieve but hard to escape. Nonetheless, MSE and CoP do perform much better than the one from single level GP despite a poor result in MAE. This shows that the predictions further away from the ground truth are punished significantly. Another fact we can notice is the staggering amount of time it needs for training. An increasing time is expected since we provide additional data for the model. The poor performance might also result from the implementation details inside this toolkit and the incompatibility to the hardware we use.

To sum up, we still experience the superior performance from the first multi fidelity model we tried, and this motivates us to build more sophisticated models shown in the next few sections.

3.5.5 (4) NARGP model

In this section, we introduce a nonlinear multi fidelity model that is linked by a Gaussian process. In Figure 3.1, we lay out a general setting for a nonlinear multi fidelity model. $f(X)$ is chosen to be a Gaussian process, GP_{low}, in this NARGP model, which can be seen in

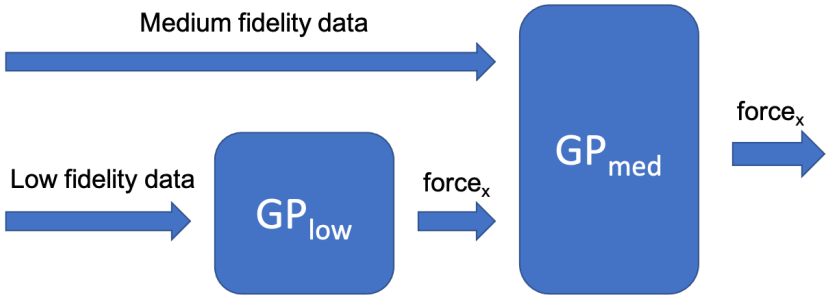


Figure 3.6: Architecture of the data flow for two fidelities NARGP models during training.

Figure 3.6. The output is then passed as input along with medium fidelity data to another GP, GP_{med}, which is where the nonlinearity comes from. For the hyperparameters, we follow the tradition and keep the restart times to be five so that it can be compared to the rest of the models on the same baseline. We choose bfgs optimizer and set the max iteration steps to a thousand. In addition, ARD is also turned on to boost the performance. The inference step is done by sampling 100 points and taking an average, including the predicted mean and variance.

Table 3.5 shows the results of the testing samples we got from the NARGP model.

Table 3.5: The results of (4) NARGP model. Format: average $\pm$ standard deviation

MSE	10.412 ± 2.908
MAE	1.122 ± 0.101
CoP	0.714 ± 0.019
Training time (s)	4683 ± 657

You might notice a rather large standard deviation appeared in both MSE and MAE. This is due to different local minimum that different experiments found. It indicates that there exists multiple regional minimums during parameters optimization which can be relatively easy to enter. Since we only set the restart times to five in each experiment, std might be lower if we restart for more times. MSE can be as low as 8.234 which is much better than all the models earlier, but in some rare occasions the number can jump up to 13.597 which is even worse than linear multi fidelity model. This tells us again that in the future we might have to treat the models with larger iterations or restart times. Overall, all the metrics present us exceptional results as compared to linear MF models, proving its ability to capture more complex relationship between low and medium fidelity data. The average training time is an affordable number of 1.3 hour in local laptop, which can be further reduced if we move the training to a larger computing cluster.

3.5.6 (5) NARGP+MLP model

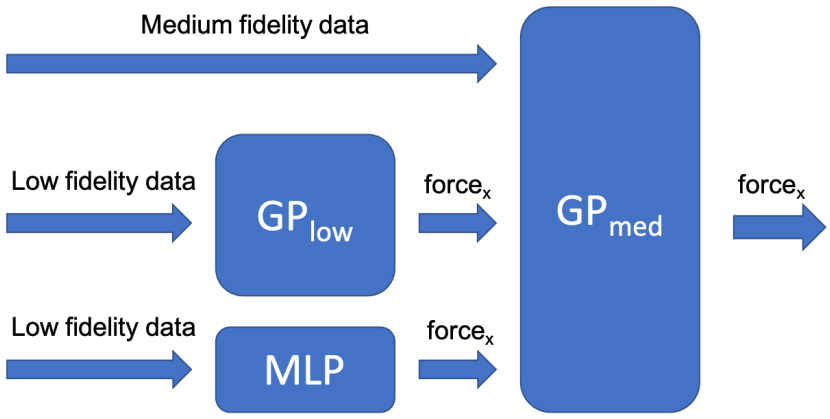


Figure 3.7: Architecture of the data flow for two fidelities NARGP+MLP models during training.

After the success in (4) NARGP model, we decide to enhance the low fidelity evaluation function $f(X)$ by adding another MLP along with the existing GP_{low}. We demonstrate the architecture of this model in Figure 3.7. Both the MLP and GP_{low} models consume and train with the same low fidelity data. There is one more input to the GP_{med} model since we add MLP. This gives us 10 input features to the model, namely the existing 8 medium fidelity features, one force prediction from GP_{low} and one from MLP. Note that this MLP model is different from (1) MLP model since this time we provide it with low fidelity data rather than medium one.

However, we decide to keep using the same architecture and hyperparameters set for this newly added MLP evaluation, since it is proved to be functional in earlier section. Its hyperparameters can be found in Table 3.1. The rest of the setting remain the same as (4) NARGP model.

Table 3.6 shows the results of the testing samples we got from the NARGP+MLP model.

Table 3.6: The results of (5) NARGP+MLP model. Format: average $\pm$ standard deviation

MSE	9.859 ± 2.362
MAE	1.103 ± 0.085
CoP	0.709 ± 0.009
Training time (s)	4834 ± 457

Both the average MSE and MAE are lower than (4) NARGP model, which shows the importance of a good selection of $f(X)$. Although CoP is a bit lower, it is within 0.7% difference to previous NARGP model, so it should be negligible. In addition, training time

is having a 3% increase from the one without MLP, which might even be caused by the fluctuation of laptop performance. It is certainly tolerable to suffer slightly longer training time for noticeable metrics improvement.

Overall, the metrics are better than (4) NARGP model. This is somewhat explainable since from previous computational experiments, MLP is shown to perform better than GP in single fidelity setting. Therefore, we can understand (5) model as (4) model with better $f(X)$ so that GP_{med} gets more reliable input which leads to improved results.

In this section, even though we add one more model and get comparatively better result, the information is still limited since it is the same data being fed to different models. In order to boost the performance even further, we are going to introduce more data so that the model can have different patterns to learn from.

3.5.7 (6) NARGP+CNN model

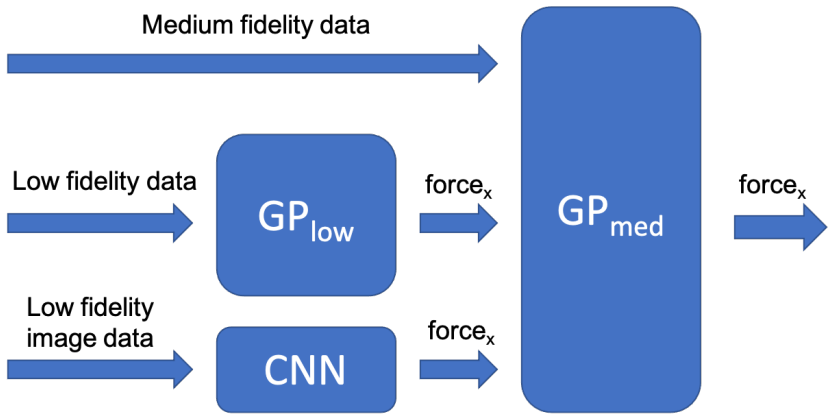


Figure 3.8: Architecture of the data flow for two fidelities NARGP+CNN models during training.

For the last model we developed in this thesis, we decide to employ CNN to be able to include surface geometry information in our multi fidelity model. As shown in Figure 3.8, we replace MLP in previous model with CNN and let the CNN consume only image data from low fidelity source. The rest of the model remain the same, as GP_{med} still has 10 input columns, where 8 of them are medium fidelity features, one from low fidelity feature and the last one from low fidelity 64*64 images.

We have tried a variety of architectures for CNN, and we end up choosing current model, a rather small network. In Figure 3.9, you can see that for the first block we use 16 different kernels to be able to capture different patterns in the images and cut the width and height in half by setting stride = 2. It is then followed by a batch normalization layer, an activation

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 32, 32]             160
       BatchNorm2d-2          [-1, 16, 32, 32]              32
              ReLU-3          [-1, 16, 32, 32]               0
         MaxPool2d-4          [-1, 16, 16, 16]               0
            Conv2d-5            [-1, 16, 8, 8]           2,320
       BatchNorm2d-6            [-1, 16, 8, 8]              32
              ReLU-7            [-1, 16, 8, 8]               0
         MaxPool2d-8            [-1, 16, 4, 4]               0
           Flatten-9                 [-1, 256]               0
           Linear-10                   [-1, 1]             257
================================================================
Total params: 2,801
Trainable params: 2,801
Non-trainable params: 0
```

Figure 3.9: A screenshot of the CNN architecture we used.

layer and a 2d max pooling layer. The second convolution block is really similar. There are still 16 kernels, but this time each one has depth = 16. At the end of the network, there is a dense neural network to collect all the 2d spatial information and make prediction from it. There are only 2801 trainable parameters in this network which is quite tiny in current benchmark. We try to tune the architecture so that the majority of the parameters lies in the convolutional blocks.

Table 3.7: The hyperparameters used in the CNN in (6) model

Activation function	Relu
Batch size	512
Learning rate	0.00001
Optimizer	Adam
Epoch	200
weight decay	0.01

In Table 3.7, we present the hyperparameters used in the CNN. Since there are more than 60k images for training, we increase the batch size so that each batch is representative enough to update the network parameters toward meaningful direction. The size also should not be too large to give the algorithm a chance to break through the local minimum. In addition, weight decay is one of the regularization techniques to avoid overfitting by adding a L2 norm of the weights to the loss function. Normally, it can increase the training loss but reduce the validation loss. Most of the hyperparameters are somewhat empirical,

but this is the reason why we need a validation set. We adjust the hyperparameters with the results from the validation set and test the entire network with the test set.

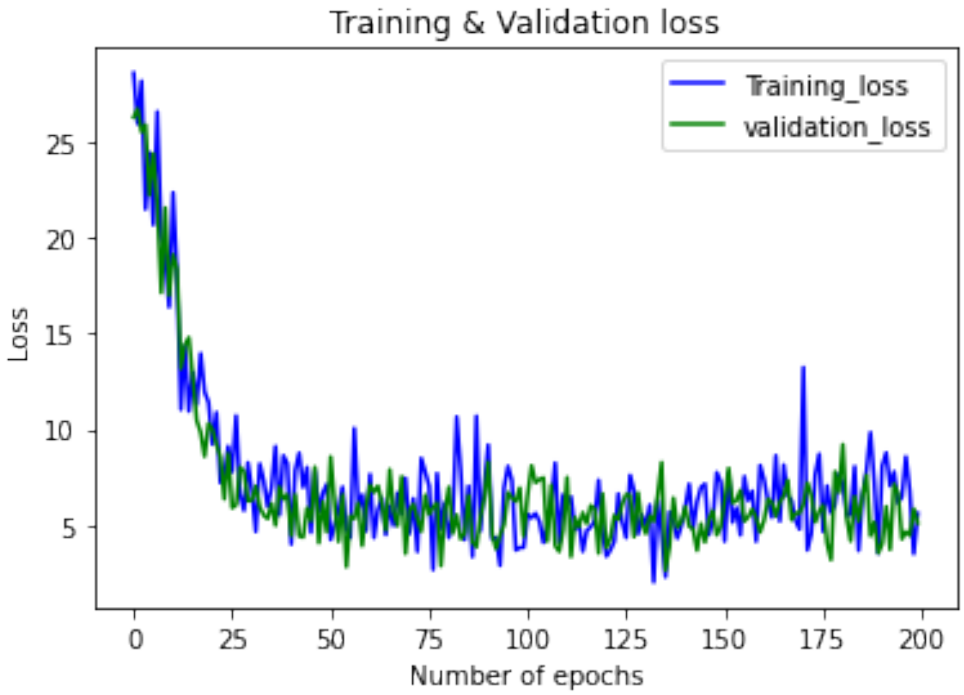


Figure 3.10: Loss curve of the CNN model during training.

The loss curve for the training and validation is shown in Figure 3.10. We did not set the training to enable early stopping; instead, we fix the training epoch to be 200 and only save the model with the best validation loss. The saved model has training, validation and test loss = 2.29, 2.63 and 6.23, respectively, and the training process takes around one and half hour. This model is then used as a component in multi fidelity model during inference.

Table 3.8 shows the results of the testing samples we got from the NARGP+CNN model.

Table 3.8: The results of (6) NARGP+CNN model. Format: average $\pm$ standard deviation

MSE	8.448 ± 0.354
MAE	1.055 ± 0.01
CoP	0.712 ± 0.004
Training time (s)	4400 ± 920

According to the results, both MSE and MAE are less than (4) NARGP and (5) NARGP+MLP

model with less standard deviation as well. CoP is comparable to NARGP model with trivial difference. However, we expect the training time to be more than the training time of (4) model, but it is actually less, possibly due to the fluctuation of laptop performance. A fairer comparison is to add up the training time for CNN to it, which will then give us around 9854 seconds. To conclude, we achieve the best model by first optimizing the best CNN model and merge it in NARGP framework.

3.5.8 Evaluation of all the models

After we present all the models in detail, in this section we summarize all the results in a table and compare them. In addition, the influence of the number of medium fidelity points in a multi fidelity setting will also be experimented and discussed in order to provide intuition for future research.

Table 3.9 compares the results from all the models in this thesis.

Table 3.9: A summary of all the results

Models	MSE	MAE	CoP	Training time (s)
(1) MLP	12.872 ± 0.062	1.127 ± 0.023	N/A	0.52 ± 0.085
(2) GP	14.448 ± 0.363	1.308 ± 0.015	0.637 ± 0.004	178.288 ± 46.781
(3) Linear MF	12.569 ± 0	2.002 ± 0	0.641 ± 0	12995 ± 2124
(4) NARGP	10.412 ± 2.908	1.122 ± 0.101	0.714 ± 0.019	4683 ± 657
(5) NARGP+MLP	9.859 ± 2.362	1.103 ± 0.085	0.709 ± 0.009	4834 ± 457
(6) NARGP+CNN	8.448 ± 0.354	1.055 ± 0.01	0.712 ± 0.004	4400 ± 920

In terms of both MSE and MAE, (6) NARGP+CNN model gives us the best result, which can probably be attributed to the extra surface geometry information being added to the input. Throughout the five trials, the performance of the model remains rather stable by providing a small standard deviation. If surface images are limited, (5) NARGP+MLP model is still a promising candidate given its superior performance to the plain NARGP model. When it comes to CoP, NARGP is actually the best; however, there is no clear difference between the last three models. All of them are good at providing reasonable estimates to $force_x$ as compared to pure GP and liner MF models. MLP model shines at its extremely short training time comparing to all the models, and it also has possibility to be further improved by providing more hyperparameter and architecture optimizations.

Overall, with our experiments, NARGP framework is proven to be able to take advantage of the extra few medium fidelity data and boost the performance. We can clearly notice the huge gap between NARGP and non-NARGP models in all the metrics. Since training efficiency is not a huge concern when it comes to the real-world application of wheeled rovers, we can neglect its poor performance in training time at the moment. Interestingly,

some models might perform better at MSE but worse at CoP. CoP is a metrics that also put variance into consideration, so it is also valuable during the comparison if we care about the uncertainty estimate of the models.

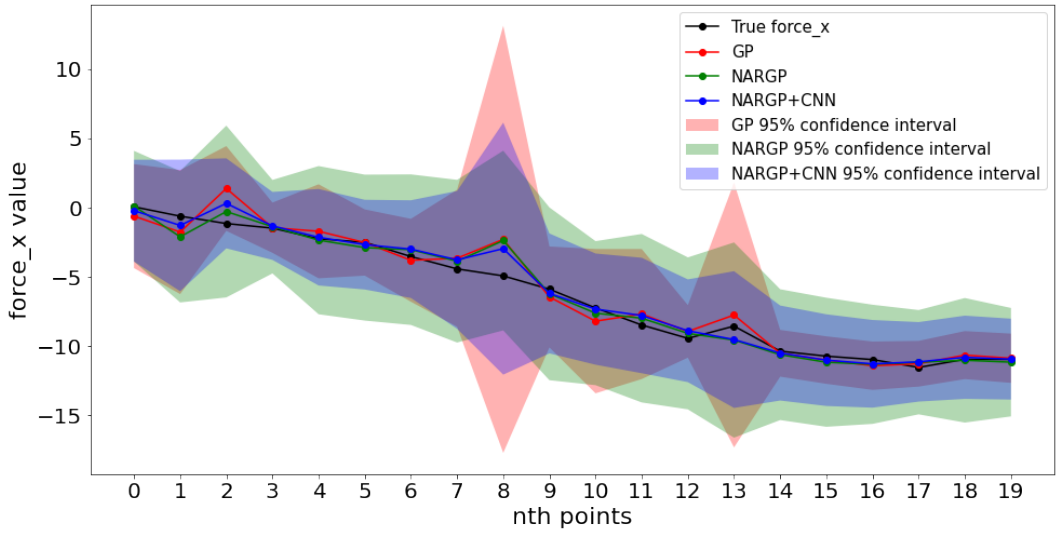


Figure 3.11: Selected 20 sequential points describing the movement of the simulated rover.

Among all the models, we are most interested in basic NARGP model and NARGP with additional CNN to detect the change of the ground. In the following paragraphs, we present some figures to demonstrate the benefit of NARGP. Also, in order to properly spot the marginal improvement when introducing extra abundant low fidelity data, we also include pure GP model as a baseline for comparison.

In Figure 3.11, we choose 20 sequential points within a simulation that are unseen to the models. Model (2), (4) and (6) are compared with each other against the ground truth. We pick the time interval where $force_x$ remains quite steady and observe the predictions from these three models. Although we cannot judge a model's performance simply from 20 selected points, we can still observe some interesting phenomenon from the example. Also, it is a great way to display the properties of GP based models.

For each model in each point, we can plot a 1d Gaussian distribution since we have its mean and variance, and we can compare how close the predicted mean is to the ground truth. We plot the lines with 95% confidence interval, meaning 2 times standard deviation away from the expected mean. Throughout these 20 points, even if the predictions are deviated, the true $force_x$ are all situated within the interval. This is a positive sign indicating that the models give reliable predictions. Moreover, we can clearly notice the trend that NARGP based models perform better than pure GP model, especially when x = 2 and 8. NARGP and NARGP+CNN give similar prediction mean, but NARGP generally has

larger variance, meaning its CoP should be larger.

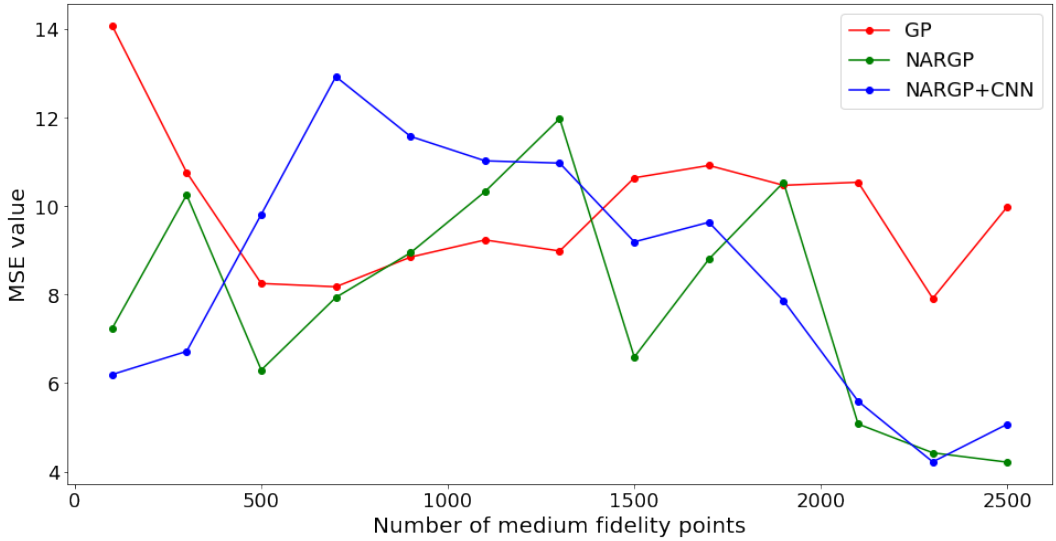


Figure 3.12: The influence of the number of medium fidelity points on MSE.

In Figure 3.12, 3.13 and 3.14, we exhibit the impact of the number of medium fidelity points on models' MSE, MAE and CoP. Experiments are conducted from 100 points to 2500 points with each step being 200. Since the number of low fidelity points is 5000, we designed this so that we can observe the trend where the medium to low fidelity ratio spans from 1:50 to 1:2. In order to design the experiments fairly, we first select a different random seed from before and pick 2500 samples. For each experiment, we slice the first x samples to make sure the models are gradually witnessing more data as x increases from 100 to 2500.

In the case of MSE, unfortunately there is no clear trend being found when the number of medium fidelity points increases. Although NARGP and NARGP+CNN perform the best when x is over 2000, the rest of the curves fluctuate quite heavily. Judged from the graph, there is no guarantee showing that more medium fidelity data can reduce MSE effectively. Some of the newly introduced samples might act as the noise to confuse the models. In Figure 3.13, the clear trend is that both NARGP and NARGP+CNN are consistently better than GP. While NARGP based models are relatively similar in value, there is still no clear trend regarding the influence of the number of medium fidelity points on MAE. For GP model, it seems like there is no benefit of adding more medium fidelity data after the ratio surpasses 1:10.

When it comes to CoP in Figure 3.14, it is obvious that all the models show an upward trend when the number of medium fidelity data increases. In both NARGP and NARGP+CNN, the improvement is significant when the ratio increases from 1:50 to 1:10; however, after that, the value reaches a plateau where an increase in medium fidelity samples does not

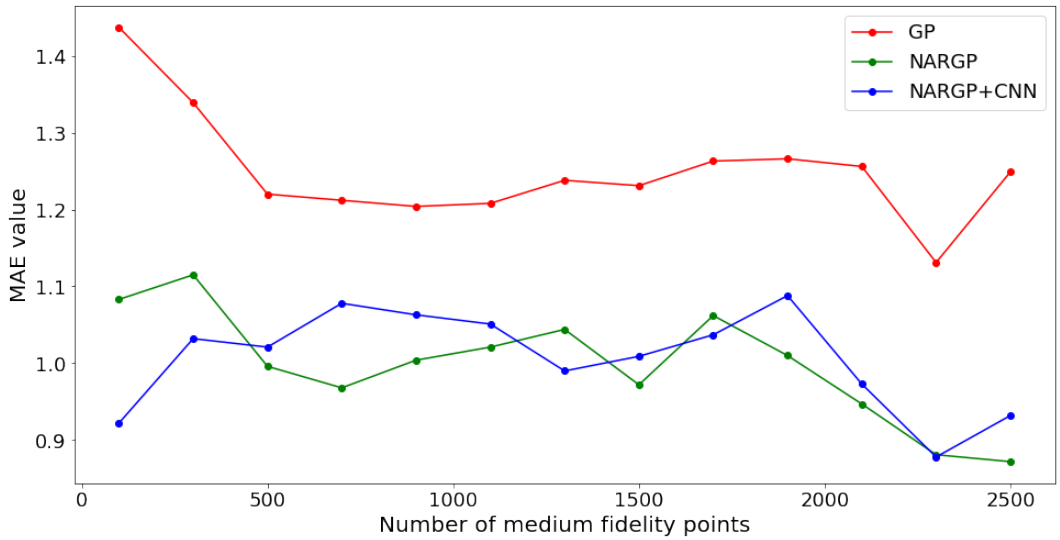


Figure 3.13: The influence of the number of medium fidelity points on MAE.

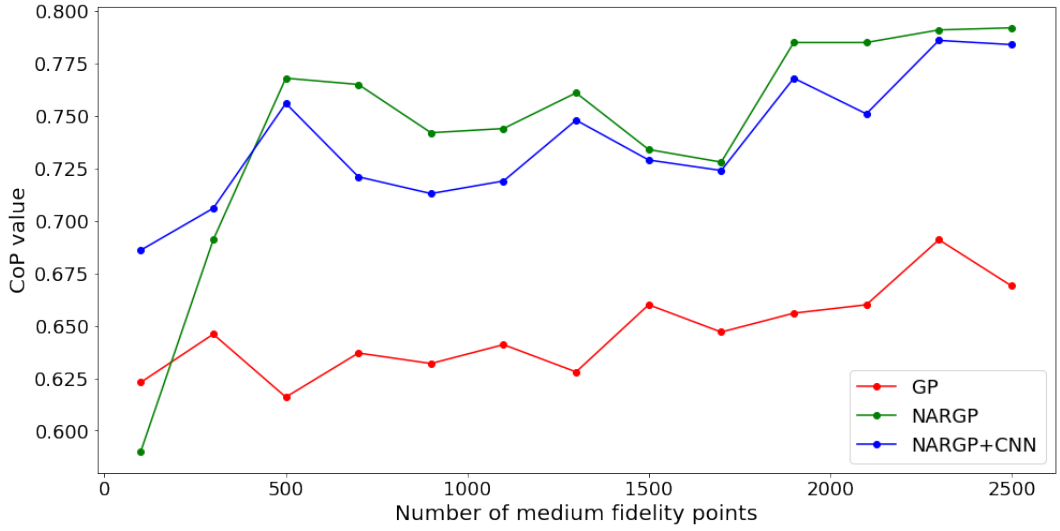


Figure 3.14: The influence of the number of medium fidelity points on CoP.

drastically improve the models' CoP.

Overall, the trend in this CoP figure is expected since the CoP of GP should steadily increase when it is exposed to more data, and NARGP and NARGP+CNN models do require certain number of medium fidelity points to fix its predictions across the entire domain regardless of the ample supply of low fidelity data. Intriguingly, when the number of medium fidelity data is small, it can be explained that NARGP+CNN performs better than NARGP because it has extra surface images to aid its decision.

To combine with the computational experiments in the earlier sections, what we learned from these graphs is that we can achieve similar if not better value in all the metrics if we only use half of the medium fidelity data, i.e., 500 instead of 1000. Also, this is surprisingly true not just for NARGP based models but also for pure GP model as well. We hope that these experiments can provide some insights to the future experiments design.

3.6 Performance differences between kernels

In the last section of the thesis, we decide to explore other kernels other than the default RBF kernel to see if there are more suitable kernels for the properties of wheeled mobile rovers, and we are revealing our computational results in the following tables. In NARGP and NARGP+CNN models, we replace both the kernel of low fidelity GP and the kernel of the GP to fuse medium fidelity data and low fidelity GP's output. We choose 4 different kernels, namely linear, Matérn32, Matérn52 and Rational Quadratic kernels. Except for the linear kernel, the rest of the kernels can be seen as the extension of RBF kernel, with different assumptions.

Table 3.10 compares the results of MSE when employing different kernels. We can see that RBF kernel is almost the worst kernel in all three models. On the contrary, linear kernel stands out from all the kernels to be the one that gives the GP and NARGP+CNN models the lowest MSE. From the perspective of the models, the switch of the kernel can provide the most apparent influence on the GP model, while for NARGP+CNN, the change of kernel is quite trivial.

Table 3.10: The results of MSE in different kernels

Kernels	GP	NARGP	NARGP+CNN
RBF	14.448	10.412	8.448
Linear	9.432	9.708	8.28
Matérn32	12.947	8.522	8.436
Matérn52	13.391	8.519	8.315
Rational Quadratic	10.912	8.825	8.992

Table 3.11 compares the results of MAE when using different kernels. The results show us again that a better MSE does not necessarily lead to a better MAE, despite sharing a positive correlation. For GP, the model achieves the best result when using rational quadratic kernel, while for NARGP and NARGP+CNN, Matérn32 kernel gives us the best result, even though its difference to other kernels is not too large.

Table 3.11: The results of MAE in different kernels

Kernels	GP	NARGP	NARGP+CNN
RBF	1.308	1.122	1.055
Linear	1.192	1.441	1.169
Matérn32	1.159	1.015	1.015
Matérn52	1.207	1.051	1.044
Rational Quadratic	1.11	1.069	1.059

The results of CoP while we apply different kernels on the selected models is shown in Table 3.12. The most interesting result still lies in the row of linear kernel. It achieves the lowest CoP in GP model regardless of the best MSE score. This could indicate small standard deviations. Furthermore, the usage of linear kernel also makes NARGP and NARGP+CNN stand out from the rest of the kernels by a huge margin. The reason behind this is yet to know, but it is exciting to know that the assumed function prior can have tremendous influence on the final results.

Table 3.12: The results of CoP in different kernels

Kernels	GP	NARGP	NARGP+CNN
RBF	0.637	0.714	0.712
Linear	0.483	0.872	0.852
Matérn32	0.673	0.75	0.727
Matérn52	0.658	0.74	0.707
Rational Quadratic	0.665	0.733	0.713

Table 3.13 compares the results of training time when different kernels are used on the three models. In all the models, rational quadratic kernel makes the training time the longest, so it is something to be considered before employing. Due to the O(n) complexity, NARGP as well as NARGP+CNN spend the least amount of time when using linear kernel. Still, the training time is massively impacted by the performance fluctuation of individual laptop.

Overall, there is no clear winner on which kernel is the most suitable for our problem. Some kernels might perform better at certain metrics and on specific models. Linear kernel does perform surprisingly great on certain scenarios, so it could be an interesting direction

Table 3.13: The results of training time in different kernels

Kernels	GP	NARGP	NARGP+CNN
RBF	178	4683	4400
Linear	179	3320	3003
Matérn32	202	4725	3618
Matérn52	157	5936	6191
Rational Quadratic	374	6802	7087

to dive in. In the conclusion section, we will disclose some of the state-of-the-art innovations on the construction of kernels being done by others and their applications on several domains.

4 Conclusions

People use both virtual simulations and real-world experiments to drive the development of the locomotion of rovers in any terrain. These have produced tons of data for data-driven methods. Depending on the data sources, we can divide them into more reliable high fidelity data as well as less accurate low fidelity data.

In this thesis, we construct six models to tackle the force prediction problem of wheeled mobile robot. Neural networks models such as MLP are good at faster training time while Gaussian process models are known for its ability to estimate the prediction uncertainty. Out of all the models used in the thesis, multi-fidelity models show promising potential for terramechanics applications by taking advantage of the fusion of high and low fidelity data. Specifically, with extra CNN model to detect the change of the surface and fuse into the low fidelity part of the model, model (6) NARGP+CNN performs the best in both MSE and MAE. In addition, we also experiment a wide range of medium-low fidelity data ratio and show that 1:10 can be a possible starting ratio for any future endeavor.

Kernel is the key when it comes to constructing a Gaussian process model. We show that the choice of kernel can greatly influence the performance of a model. In addition to RBF kernel, our results indicate that certain kernel can have superior result in certain scenario, depending on the need. Since kernel function is our guess to the characteristics of the data, it becomes natural to incorporate the underlying physical properties of the data. For example, power grid dynamics was incorporated in the prior of PhI-GPR by solving stochastic differential equations [42]. Also, Hanuka, A *et al.* expand the simulation $f(x)$ from the optimum point and construct the RBF covariance function in their Gaussian process model [43]. These methods are called Physics-informed model and have been widely applied in many domains [44, 45]. Thus, one of the main directions in the future would be to build kernels specifically for the rovers in sandy terrain.

Data is the fuel to a machine learning model. In this thesis, we train the model with relatively simple rover trajectory, so the movement that the model can learn from is quite limited. That is to say, we only fill up a tiny portion of the input space spanned by the selected features. Therefore, it will be both a challenge and an opportunity to train the models with a variety of trajectories. Sequential or time dependent models might also be used to understand the locomotion of the rovers. Speaking of data, when we select the data randomly for training, validation and testing, we usually fix a random seed so that we can have identical results if we decide to run it again. During our experiments, we find out

different random seed can significantly impact the result of a model, since some important snapshots of trajectories might not be selected during the learning process.

CoP is one of the metrics that we use to evaluate the models. It refers to the area under the probability density function. In the thesis, we assume that a larger CoP represents a better model, which might only be true to some extent. A larger CoP value means the ground truth is close to the expected mean. When the CoP of the model is too close to one, it means all the predicted means are close to the ground truths. Ideally, we do not want it to happen, since we expect the output of the Gaussian process model to have certain level of variance. It should be probable for the ground truth to be one or two standard deviation away from the expected mean. Therefore, we also need a threshold of CoP to better evaluate and compare the performance.

In conclusion, data-driven methods, especially multi-fidelity models, has shown promising potential to assist the development of wheeled mobile robot in extraterrestrial environment.

List of Figures

List of Tables

Bibliography

[1] C. E. Rasmussen and C. K. I. Williams. Gaussian processes for machine learning. *the MIT Press*, 2006.

[2] Benjamin Peherstorfer, Karen Willcox, and Max Gunzburger. Survey of multifidelity methods in uncertainty propagation, inference, and optimization, 2018.

[3] Andy Keane. Cokriging for robust design optimization. *AIAA Journal*, 50:2351–2364, 11 2012.

[4] David Toal, Andy Keane, Diego Benito, Jeffery Dixon, Jingbin Yang, Matthew Price, Trevor Robinson, Alain Remouchamps, and Norbert Kill. Multifidelity multidisciplinary whole-engine thermomechanical design optimization. *Journal of Propulsion and Power*, 30:1654–1666, 11 2014.

[5] Joëlle Bailly and Didier Bailly. Multifidelity aerodynamic optimization of a helicopter rotor blade. *AIAA Journal*, 57:1–12, 04 2019.

[6] Raymond Arvidson, D. Fuller, M. Heverly, K. Iagnemma, J. Lin, J. Matthews, Ronald Sletten, and Nathan Stein. Mars science laboratory curiosity rover terramechanics initial results. *44th Lunar and Planetary Science Conference*, pages 1193–, 03 2013.

[7] Feng Zhou, Raymond Arvidson, Keith Bennett, Brian Trease, Randel Lindemann, P. Bellutta, Karl Iagnemma, and Carmine Senatore. Simulations of mars rover traverses. *Journal of Field Robotics*, 31, 01 2014.

[8] Zhenzhong Jia, William Smith, and Huei Peng. Terramechanics-based wheel–terrain interaction model and its applications to off-road wheeled mobile robots. *Robotica*, 30, 05 2012.

[9] Haibo Gao, Junlong Guo, Nan li, Zhen Liu, Guangjun Liu, and Zongquan Deng. Longitudinal skid model for wheels of planetary exploration rovers based on terramechanics. *Journal of Terramechanics*, 50:327–343, 10 2013.

[10] Zhenzhong Jia, William Smith, and Huei Peng. Fast analytical models of wheeled locomotion in deformable terrain for mobile robots. *Robotica*, 31, 01 2013.

[11] Fabian Buse. Fully automated single wheel testing with the dlr terramechanics robotics locomotion lab (troll). In *15th Symposium on Advanced Space Technologies in Robotics and Automation*, 05 2019.

[12] R. E. Arvidson et al. Spirit mars rover mission: Overview and selected results from the northern home plate winter haven to the side of scamander crater. *Journal of Geophysical Research: Planets*, 115(E7), 2010.

[13] Stefan Barthelmes. Terra: Terramechanics for real-time application. In *The 5th Joint International Conference on Multibody System Dynamics*, 06 2018.

[14] Fabian Buse. Using superposition of local soil flow fields to improve soil deformation in the dlr soil contact model - scm. In *The 5th Joint International Conference on Multibody System Dynamics*, 06 2018.

[15] Ali Azimi, Jozsef Kovecses, and Jorge Angeles. Wheel–soil interaction model for rover simulation and analysis using elastoplasticity theory. *Robotics, IEEE Transactions on*, 29:1271–1288, 10 2013.

[16] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[17] Ravindra Parmar. Training deep neural networks, 2018. https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964.

[18] Stanford cs231n, 2019. https://cs231n.github.io/neural-networks-1/.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[20] Herbert E. Robbins. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 2007.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[22] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591 vol.1, 1993.

[23] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey, 2018.

[24] Matthias Niessner. Optimization and backpropagation, 2020. https://niessner.github.io/I2DL/slides/4.Optimization_and_Backprop.pdf.

[25] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, pages 319–345. Springer Verlag, 1999.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[29] David Duvenaud. The kernel cookbook: Advice on covariance functions, 2014. https://www.cs.toronto.edu/~duvenaud/cookbook/.

[30] Geoffrey E Hinton and Russ R Salakhutdinov. Using deep belief nets to learn covariance kernels for gaussian processes. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007.

[31] G. E. P. Box and George C. Tiao. Bayesian inference in statistical analysis. *International Statistical Review*, 43:242, 1973.

[32] Jian Wu, Matthias Poloczek, Andrew Gordon Wilson, and Peter I. Frazier. Bayesian optimization with gradients, 2018.

[33] Benjamin Peherstorfer, Karen Willcox, and Max Gunzburger. Survey of multifidelity methods in uncertainty propagation, inference, and optimization. *SIAM Review*, 60(3):550–591, 2018.

[34] Loïc Brevault, Mathieu Balesdent, and Ali Hebbal. Overview of gaussian process based multi-fidelity techniques with variable relationship between fidelities, 2020.

[35] MC Kennedy and A O'Hagan. Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87(1):1–13, 03 2000.

[36] Perdikaris P., Raissi M., Damianou A., Lawrence N. D., and Karniadakis G. E. Nonlinear information fusion algorithms for data-efficient multi-fidelity modelling. *Proc. R. Soc. A.*, 473:20160751, 2017.

[37] R. Lichtenheldt and O. Kromer. Soil modeling for insight's hp3-mole: From highly accurate particle-based towards fast empirical models. *ASCE Earth and Space*, 2016.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[39] GPy. GPy: A gaussian process framework in python. http://github.com/SheffieldML/GPy, since 2012.

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Te-

jani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[41] Andrei Paleyes, Mark Pullin, Maren Mahsereci, Cliff McCollum, Neil D. Lawrence, and Javier Gonzalez. Emulation of physical processes with emukit, 2021.

[42] Tong Ma, David Alonso Barajas-Solano, Ramakrishna Tipireddy, and Alexandre M. Tartakovsky. Physics-informed gaussian process regression for probabilistic states estimation and forecasting in power grids, 2020.

[43] A. Hanuka, J. Duris, J. Shtalenkova, D. Kennedy, A. Edelen, D. Ratner, and X. Huang. Online tuning and light source control using a physics-informed gaussian process. 2019.

[44] Christopher G. Albert. Gaussian processes for data fulfilling linear differential equations. 2019.

[45] Xiu Yang, David Barajas-Solano, Guzel Tartakovsky, and Alexandre M. Tartakovsky. Physics-informed CoKriging: A gaussian-process-regression-based multifidelity method for data-model convergence. *Journal of Computational Physics*, 395:410–431, oct 2019.