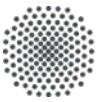


University of Stuttgart

Institute for Artificial Intelligence
Analytic Computing

Universitätsstraße 32
70569 Stuttgart

Master Thesis

Bayesian Symbolic Regression in Structured Latent Spaces

Chenlei Pei

Study program:	Autonomous Systems
1. Examiner:	Prof. Dr. Steffen Staab
2. Examiner:	Prof. Dr. Mathias Niepert
Advisors:	Tim Schneider, M.Sc.
start date:	15.08.2024
end date:	14.02.2025

Abstract

Symbolic regression is an interpretable machine learning method that learns mathematical expressions from given data. It naturally combines with Bayesian Inference which lets experts express their knowledge as prior distributions over equations. However, the infinite search space of mathematical expressions renders exhaustive search impractical, and Bayesian Inference remains costly.

Therefore, we propose to execute the Bayesian Reasoning in the learned latent space of a trained Variational Autoencoder (VAE) and thereby exploit inherent structures in the search space. While latent spaces have been used to structure search spaces, our approach provides the probability of each mathematical expression rather than selecting the best one. We suggest practical approximations to the posterior distribution in latent space and obtain formula examples by sampling from the posterior using the Gaussian Process Hamiltonian Monte Carlo (GP-HMC) method. We have validated our method using various Koza, Nguyen and self-generated datasets and compared it against genetic programming and SInDy concerning the Root Mean Square Error (RMSE).

Keywords: Symbolic Regression, latent space, Variational Autoencoder, Character Variational Autoencoder, Grammar Variational Autoencoder, Bayesian Reasoning, Gaussian Process, Hamiltonian Monte Carlo, Gaussian Process Hamiltonian Monte Carlo

Kurzfassung

Die symbolische Regression ist eine interpretierbare Methode des maschinellen Lernens, die mathematische Ausdrücke aus gegebenen Daten lernt. Sie kombiniert sich natürlich mit der Bayesianischen Inferenz, die es Experten ermöglicht, ihr Wissen als Priorverteilungen über Gleichungen auszudrücken. Jedoch macht der unendliche Suchraum mathematischer Ausdrücke eine erschöpfende Suche unpraktisch, und die Bayesianische Inferenz bleibt kostspielig.

Daher schlagen wir vor, das Bayesianische Schließen im erlernten latenten Raum eines trainierten Variational Autoencoders (VAE) durchzuführen und damit die inhärenten Strukturen im Suchraum zu nutzen. Während latente Räume verwendet wurden, um Suchräume zu strukturieren, liefert unser Ansatz die Wahrscheinlichkeit jedes mathematischen Ausdrucks, anstatt den besten auszuwählen. Wir schlagen praktische Annäherungen an die posteriore Verteilung im latenten Raum vor und erhalten Formelbeispiele durch das Probenahmeverfahren aus dem Posterior mithilfe der Hamiltonian Monte Carlo-Methode des Gaussian-Prozesses. Wir haben unsere Methode mit verschiedenen Koza, Nguyen und selbst erstellten Datensätzen validiert und sie im Hinblick auf den Root Mean Square Error (RMSE) mit genetischer Programmierung und SInDy verglichen.

Schlüsselwörter: Symbolic Regression, latent space, Variational Autoencoder, Character Variational Autoencoder, Grammar Variational Autoencoder, Bayesian Reasoning, Gaussian Process, Hamiltonian Monte Carlo, Gaussian Process Hamiltonian Monte Carlo

Contents

Acronyms

Character VAE Character Variational Autoencoder. 9

CNN Convolutional Neural Network. 13

DNN Dense Neural Network. 12

GP Gaussian Progress. 9

GP-HMC Gaussian Progress Hamiltonian Monte Carlo. 9

GRU Gated Recurrent Unit. 14

GVAE Grammar Variational Autoencoder. 9

HMC Hamiltonian Monte Carlo. 9

KL Divergence Loss Kullback-Leibler Divergence Loss. 16

LSTM Long Short-Term Memory. 13

MCMC Markov Monte Carlo. 20

NN Neural Network. 23

PCFGs Probabilistic context-free grammars. 22

RMSE Root Mean Squared Error. 41

RNN Recurrent Neural Network. 13

SInDy Sparse Identification of Nonlinear Dynamical system. 10

t-SNE t-distributed Stochastic Neighbor Embedding. 49

VAE Variational Autoencoder. 9

1 Introduction

Scientists strive to find expressions of the relationships between observed data. In the machine learning community, this problem is called symbolic regression, which seeks to identify the mathematical expression that best fits the data. This problem can be described as follows: Given some data $D = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i are inputs and y_i are outputs, the task is to find an expression e that best fits the data with $e(x_i) \approx y_i$. We strongly believe that symbolic solutions are better suited for scientific applications and enable humans to understand the relationships within the data more effectively than black-box methods, like deep neural networks.

A commonly used method is Genetic Programming. It generates new expressions by mimicking the process of natural selection, such as selection, crossover, and mutation. The expressions with top performance will be used to generate a new generation. However, a significant disadvantage is that it only chooses the best result without giving uncertain information, and with genetic programming, these expressions tend to be overly complex [MC24].

While many of the challenges in symbolic regression arise from its infinite space of possible solutions, recent work on Grammar Variational Autoencoder (GVAE) and Character Variational Autoencoder (Character VAE) provides a way to reduce complex search space and do exploration in a learned latent space instead [KPH17] [CVG+14]. By using specified grammar, the generated mathematical expressions are ensured to be syntactically valid. Yet, this method is still not Bayesian and only yields a single expression that optimizes the prediction error on the training set throughout the learned latent space.

Our contribution is to do Bayesian reasoning in such a learned structured latent space for equations thus yielding an epistemic uncertainty along with a symbolic solution. Specifically, it first learns a latent space that can capture the inherent structure in mathematical equations by training a Variational Autoencoder (VAE). We estimate the probability distribution of latent space, utilizing a posterior as defined in Bayesian theory. Meanwhile, the trained encoder can map equation candidates into the latent space, which allows us to reason about the posterior probabilities of the expressions. In latent space, the posterior distribution of symbolic expressions is computed with the Gaussian Progress Hamiltonian Monte Carlo (GP-HMC) method introduced in [Ras03]. The basic idea of GP-HMC is to first build a Gaussian Progress (GP) estimate of a yet not normalized posterior distribution and then apply a Hamiltonian Monte Carlo (HMC) algorithm to sample from it [MC24] [Bet18]. Once the model is established, we perform sampling from this model. Finally, the trained decoder allows to map back samples from the latent space to a human-readable symbolic expression.

Overview The rest of this thesis is structured as follows: Chapter 2 introduces the basic concepts in this field, including symbolic regression, neural networks, context-free grammars, Bayesian theory, and sampling methods. Chapter 3 discusses the existing work on these topics, highlighting their

contributions and limitations. Chapter 4 describes the methodology, explaining our method and the entire process in detail. Chapter 5 presents the real experiments, including ablation studies that show the influence of different settings on the results of our method. We also compare our method with two classic symbolic regression methods: Genetic Programming [MC24] and Sparse Identification of Nonlinear Dynamical system (SInDy) [FKK+21]. The final chapter discusses the contributions and shortcomings of our method and suggests how it can be improved in the future.

2 Background

Machine learning, a branch of artificial intelligence, enables computers to learn from experience and improve performance without explicit programming. In machine learning, common tasks include classification and regression. Regression tasks involve predicting the output values for unknown inputs based on known input-output pairs, where the predicted values should be continuous, for example, predicting housing prices and stock market trends. A specialized domain within this field is symbolic regression, which seeks to uncover the mathematical expressions underlying data, providing deep insights into the data generation process.

We can see that both tasks share a core objective — predicting continuous values. However, symbolic regression offers a unique perspective and methodology compared to traditional regression models like linear regression or neural networks, which typically only output numerical predictions. Symbolic regression aims to discover the underlying mathematical expressions that describe the data. Symbolic regression uses genetic programming or other evolutionary algorithms to automatically search for the optimal form of these expressions, thus generating mathematical formulas that describe the relationships between inputs and outputs [MC24]. This method not only predicts future data points but more importantly, it reveals the fundamental mathematical relationships behind the data, which is particularly valuable for scientific discovery and engineering design. For instance, through symbolic regression, researchers can identify which factors significantly influence the behavior of physical or biological systems and may even discover new scientific laws.

In this chapter, we have introduced all the basic concepts and definitions involved in this thesis. We begin by defining symbolic regression, followed by an overview of neural networks, focusing on the Autoencoder and the types of neural units used in this thesis. Finally, we discuss Bayesian theory, Gaussian processes, and the Markov Monte Carlo sampling methods [MC24].

2.1 Symbolic regression

This thesis is related to the work in symbolic regression field. Symbolic regression is an interpretable machine learning method that aims to find mathematical expressions in data [MC24]. More and more attention is focused on this field since it makes the AI model more trustworthy, especially in scientific applications or health care, yielding true scientific hypotheses for the natural sciences or medicine.

Genetic Programming. One of the most classic algorithms in this field is the genetic algorithm and it has been widely used to find the best mathematical expression. First proposed already in the 1980s, it mimics genetic evolutionary behavior in nature, where in each generation equations that fit some training data well are combined. The basic steps include [MC24]:

1. **Create initial population**: Randomly generate a set of solutions, each solution referred to as an individual.
2. **Evaluation**: Assess each individual to determine their capability to solve the problem, known as "fitness".
3. **Selection**: Based on fitness, select a portion of individuals to proceed to the next generation.
4. **Crossover and Mutation**: Generate new individuals through crossover (or mating) and mutation operations to introduce genetic diversity.
5. **Repeat**: Repeat the evaluation, selection, crossover, and mutation steps until a termination condition is met (such as reaching a certain number of iterations or a fitness threshold).
6. **Output Result**: Select the individual with the highest fitness as the optimal solution to the problem.

It has limitations in structuring the search space and it also does not express a (Bayesian) uncertainty of possible solutions.

However, symbolic regression methods also face some challenges. In addition to the endless search space mentioned earlier, overfitting is another problem. Taking genetic algorithms as an example, after many generations, the length of the equations typically increases significantly. At this point, overfitting is likely to occur, meaning the model becomes too complex for the data [MC24].

To address the first issue of transforming an infinite search space into a continuous one, we adopt the approach of learning a structured latent space. The relevant concepts are introduced in the next section. To tackle the second issue, we employ Bayesian theory, taking into account both the observed data points and prior knowledge. The related concepts are discussed in section 2.3.

2.2 Neutral network and representation learning

In this thesis, we utilize neural networks, specifically the method of VAE, to transform the search space into a continuous latent space.

Neural networks are a class of machine learning models inspired by the structure and function of the brain. They are composed of layers of interconnected nodes or neurons, each of which applies a simple calculation to the data. As data passes through each layer, the network transforms the input into a more abstract and composite representation. During the forward process, **active functions** are used to introduce nonlinear properties, the most widely used activation functions include sigmoid, rectified linear units(ReLU) and WeakReLU [YZ20]. **Loss function** is used to measure the different between the output and target, the training goal of Neutral Network is generally minimizes the loss function [YZ20]. Different types of layers serve specific purposes in a neural network, here we introduce the layer types used in this thesis.

A **Dense Neural Network (DNN)** is known as a model with fully connected layers. The fully connected layers connect every input to every output neuron, and also called as dense layers. It is also commonly used to change the dimension of data, to let it fits the classification and regression tasks. The model with multiple layers dense layers has the capacity to learn the complex problem

[STE].

A **Convolutional Neural Network (CNN)** is a type of neural network specifically designed for processing structured data and the data in the form of multiple arrays, such as images or sequences. It uses convolutional layers to apply filters that extract local patterns, and builds hierarchical feature representations as the data moves through the network [LBH15].

Recurrent Neural Network (RNN).

RNN is a type of feedback neural network, which means they not only process input data in a sequence but also use their internal state (memory) to influence the output based on previously received information. RNN emerged as a significant area of research and development throughout the 1990s. These networks are specifically engineered to recognize patterns that unfold over sequences or time. A RNN is a type of neural network that incorporates feedback and connections, enabling it to process data in a sequential manner [MJ+01]. Therefore, RNN is commonly used in fields such as natural language processing, speech recognition, time series analysis, and music generation.

We first introduce the mathematical definition of a RNN, a RNN can be described by the following recurrence relation at each time step:

$$s_t = f_\theta(x_t, s_{t-1}), \tag{2.1}$$

where s_t is the state at time t, x_t is the input at time t, s_{t-1} is the state from the previous time step,

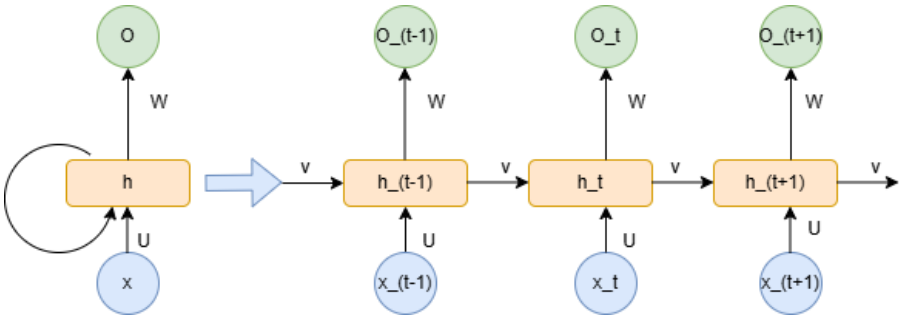


Figure 2.1: Diagram of RNN network. h is the hidden state, x is the input and o is the output while t means the time when the input is preprocessed. During the training, the output considers both input and hidden state, hidden state is updated after each time step.

and θ represents the parameters of the function f, shared across all steps. We can also illustrate this using an unfolded graph to visually represent the process [SM19].

Long Short-Term Memory (LSTM) method uses a LSTM cell replacing a normal recurrent neuron. A LSTM cell contains a memory storing the s_t at time t and three multiplicative gates, the input, forget and output gate, which control the write, reset and read operation of the memory state [SM19].

In an LSTM layer l containing M_l LSTM cells:equations:

$x_{l-1}(t) \in \mathbb{R}^{M_{l-1}}$	Layer input at time n
$x_l(t) \in \mathbb{R}^{M_l}$	Layer output at time n
$s_l(t) \in \mathbb{R}^{M_l}$	Memory state at time n, $s_l(n) \neq x_l(n)$ in contrast to RNNs
$i_l(t) \in \mathbb{R}^{M_l}$	Input gate signal at time n
$f_l(t) \in \mathbb{R}^{M_l}$	Forget gate signal at time n
$o_l(t) \in \mathbb{R}^{M_l}$	Output gate signal at time n

Then we can calculate the state of each gate and control the output by controlling the gates:

Gate Signals

$$i_l(n) = \sigma\big(W_{l,ix}x_{l-1}(n) + W_{l,io}x_l(n-1) + b_{l,i}\big) \tag{2.2}$$
$$f_l(n) = \sigma\big(W_{l,fx}x_{l-1}(n) + W_{l,fo}x_l(n-1) + b_{l,f}\big) \tag{2.3}$$
$$o_l(n) = \sigma\big(W_{l,ox}x_{l-1}(n) + W_{l,oo}x_l(n-1) + b_{l,o}\big) \tag{2.4}$$

Update Memory State

$$s_l(n) = f_l(n) \odot s_l(n-1) + i_l(n) \odot \phi\big(W_{l,sx}x_{l-1}(n) + W_{l,so}x_l(n-1) + b_{l,s}\big) \tag{2.5}$$

Layer Output

$$x_l(n) = o_l(n) \odot \phi(s_l(n)) \tag{2.6}$$

The ability of controlling the three gates makes a LSTM neuron able to selectively remember and forget the information [SM19].

Gated Recurrent Unit (GRU) method uses a GRU cell as an improvement of the traditional recurrent neuron. A GRU cell combines the functionalities of an update gate and a reset gate to control the flow of information, simplifying the structure compared to LSTM while maintaining the ability to capture long-term dependencies. The GRU method uses a GRU cell as an improvement of the traditional recurrent neuron. A GRU cell combines the functionalities of an update gate and a reset gate to control the flow of information, simplifying the structure compared to LSTM while maintaining the ability to capture long-term dependencies [DS17].

In a GRU layer l containing M_l GRU cells, the following notations are used:

$$x_{l-1}(t) \in \mathbb{R}^{M_{l-1}} \quad \text{Layer input at time } t \tag{2.7}$$
$$x_l(t) \in \mathbb{R}^{M_l} \quad \text{Layer output at time } t \tag{2.8}$$
$$s_l(t) \in \mathbb{R}^{M_l} \quad \text{Hidden state at time } t \tag{2.9}$$
$$z_l(t) \in \mathbb{R}^{M_l} \quad \text{Update gate signal at time } t \tag{2.10}$$
$$r_l(t) \in \mathbb{R}^{M_l} \quad \text{Reset gate signal at time } t \tag{2.11}$$

The GRU cell calculates the state of each gate and updates the hidden state by controlling these gates [DS17]:

Gate Signals

$$z_l(t) = \sigma(W_{z,l}x_{l-1}(t) + U_{z,l}s_l(t-1) + b_{z,l}) \tag{2.12}$$
$$r_l(t) = \sigma(W_{r,l}x_{l-1}(t) + U_{r,l}s_l(t-1) + b_{r,l}) \tag{2.13}$$

Update Hidden State

$$\tilde{s}_l(t) = \phi(W_{h,l}x_{l-1}(t) + r_l(t) \odot (U_{h,l}s_l(t-1)) + b_{h,l}) \tag{2.14}$$
$$s_l(t) = (1 - z_l(t)) \odot s_l(t-1) + z_l(t) \odot \tilde{s}_l(t) \tag{2.15}$$

Layer Output

$$x_l(t) = s_l(t) \tag{2.16}$$

The ability to control information flow through the update and reset gates enables a GRU neuron to selectively remember or forget information while maintaining a simpler structure compared to LSTM [CGCB14].

Autoencoder (AE). AE is an unsupervised neural network that learns an efficient representation of input data. It consists of two parts: an encoder and a decoder. The input data is first transferred to a latent space z by the encoder. The latent layer z is a compressed layer with fewer dimensions compared to the input data. The output is reconstructed from z through the decoder. The goal of Autoencoder is not to do self-copy but to learn the most important structures and features hidden within the data [HZ93].

The image above shows the basic structure of an Autoencoder model. The flow of data is marked by

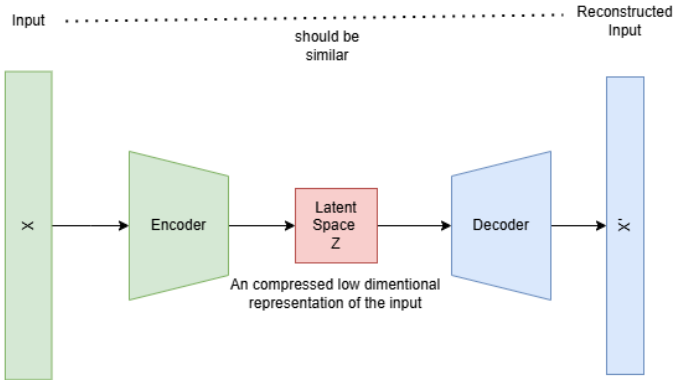


Figure 2.2: An illustration of the Autoencoder model: data propagates in the direction of the arrow, X means the input expressions, X' represents the reconstructed expressions. The latent space Z has a much more lower dimension as the input.

arrows. The latent layer z acts as a compressed representation of the input data, serving as a critical point that holds the reduced essence of the input. It is effective in capturing the key features of the data. The output X' shows the reconstructed version of the original input.

Variational Autoencoder (VAE). Although an AE can learn a latent layer from data, it is not a

generative model in a probabilistic sense. VAE uses reparameterization to impose a distribution constraint on the latent space, the usually used distribution is $\mathcal{N}(0, 1)$. The parameterization trick is a commonly used technique in the VAE method, which represents random variables as a combination of deterministic function and an independent auxiliary variable. This allows the model's gradients to be computed directly via backpropagation, thereby optimizing the model parameters. Once the model has been trained, new outputs can be obtained by randomly sampling data points from the latent space and decoding them through the decoder [KW22].

Reparameter Trick is a crucial technique that allows the model to be trained via stochastic gradient descent. This trick addresses the challenge of performing backpropagation through a network that involves random variables. In the context of VAEs, we typically need to sample a latent variable from a probability distribution, often a Gaussian distribution. Sampling directly from such a distribution poses a problem: the sampling operation is stochastic and thus not amenable to direct gradient descent. The reparameterization trick solves this issue by introducing an auxiliary noise variable, for example, noise ϵ from a standard normal distribution. In this setup, the latent variable z is not directly sampled from its distribution $q(z \mid x)$ but is reconstructed from ϵ using a deterministic function g. This function g depends on the input x and the noise ϵ, and is typically formulated as:

$$z = \mu + \sigma \odot \epsilon, \tag{2.17}$$

where μ and σ are computed by the neural network from the input x and represent the mean and standard deviation of the distribution, respectively. Through this method, the original sampling process is decomposed into two parts: generating the noise ϵ, which is a random process independent of the model parameters, and computing z, which is fully differentiable and thus optimizable via conventional backpropagation techniques. The introduction of the reparameterization trick not only enables the training of VAEs using the backpropagation algorithm but also enhances the stability and efficiency of the training process [KW22].

Loss Function is a mathematical function that quantifies the difference between the real output of a model and the goal value got from the model. The loss function in a VAE is combines two different components: reconstruction loss and Kullback-Leibler Divergence loss [KW22].

Reconstruction Loss: This component of the loss measures how effectively the decoder of the VAE is able to reconstruct the input data after it has been encoded into the latent space and then decoded back to the input space. The goal here is to minimize the difference between the original input and its reconstruction, typically using Mean Squared Error (MSE) for continuous data or Binary Cross-Entropy for binary data [KW22].

Kullback-Leibler Divergence Loss (KL Divergence Loss): This component measures how much the learned latent variable distribution (encoded by the encoder) deviates from the prior distribution, usually assumed to be a standard normal distribution [KW22].

The total loss function for a VAE is the sum of the reconstruction loss and the KL divergence with β as a weighting term that can be used to balance the two components of the loss β is a weighting term that can be used to balance the two components of the loss..

$$\text{Loss}_{\text{VAE}} = \text{Reconstruction Loss} + \beta \times \text{KL Divergence} \tag{2.18}$$

2.3 Bayesian reasoning

Bayes' Theorem. The Bayes' theorem is given by:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \tag{2.19}$$

where A and B represent two different events. The components of this theorem are explained as follows:

$P(A|B)$: This is the conditional probability of event A given that event B has occurred. It is known as the *posterior probability*, which represents the probability of event A being true when event B is already known to be true.

$P(B|A)$: This is also a conditional probability, known as the *likelihood probability*. It represents the probability of event B occurring given that event A is true.

$P(A)$: This represents the *prior probability* of event A. It is the initial degree of belief in event A, before any evidence is taken into account.

$P(B)$: Known as the *marginal probability* of event B. This probability is used to normalize the result and make $P(A|B)$ a true probability measure.

Exploiting known probabilities in this theorem to infer other probabilities is known as **Bayesian Inference**. In our paper, we apply Bayes' Theorem to the following event:

$$P(e|D) = \frac{P(D|e)P(e)}{P(D)}, \tag{2.20}$$

where event D stands for the observed data to regress and event e stands for the expressions. When calculating the posterior distribution using Bayesian theory, we naturally consider both the likelihood and the prior. By combining these two elements, likelihood and prior, rather than relying solely on likelihood, we avoid depending solely on the current observational data. This approach can help mitigate the issue of overfitting to some extent.

2.4 Sampling theory

As discussed in the introduction, our method has transformed the infinite search space into a continuous search space. The final symbolic regression equations are obtained through sampling in this continuous search space. Although the dimensionality of the latent space is significantly reduced compared to the input, it typically remains a high-dimensional space. Therefore, how to efficiently search in a high-dimensional space is also one of the research questions addressed in this thesis. In this section, we introduce some basic concepts of sampling.

The probability distribution of a sampling space is a fundamental concept in statistics that describes how probabilities are assigned to different outcomes in a sample space. It gives a mathematical description of the likelihood of occurrence of different possible outcomes [Ber]. In this thesis, we assume that the probability distribution in the latent space is continuous, namely a continuous probability distribution in a continuous space. Specifically, we assume the probability distribution in

the latent space follows a Gaussian process [SEE04]. This approach allows us to capture nonlinear relationships within the data while also incorporating uncertainty.

2.4.1 Gaussian Progress

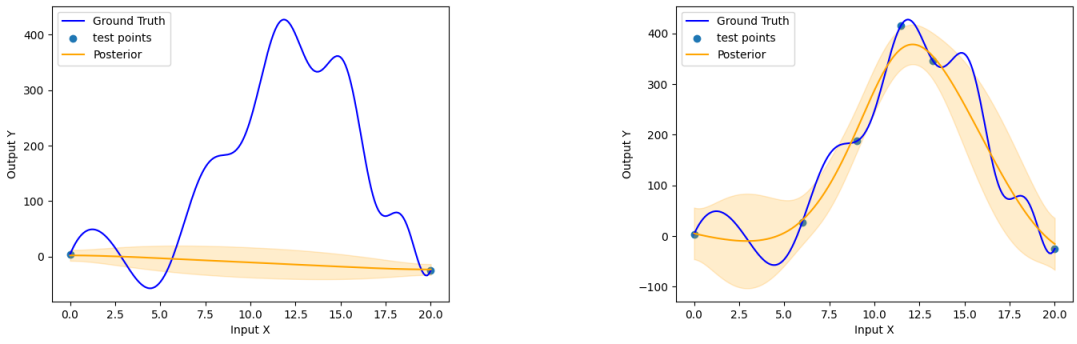
GP is a stochastic process, whose every finite collection of random variables obeys the multiple Gaussian distribution. The distribution of the Gaussian Process is over functions with continuous domain. Kernel functions (covariance functions) are typically used to describe the distance or relationship between data points. By applying a GP model, we estimate a function over a space that allows predicting the value at each point with its mean value and variance [SEE04].

For simplicity, we first consider a noiseless Gaussian model. The outputs $\mathbf{f}$ and the test outputs $\mathbf{f}^*$ follows the joint distribution [RW06].

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right). \tag{2.21}$$

The figures below depict a one-dimensional Gaussian process, the left figure shows the posterior and ground truth with only two test points while the right figure with 8 test points. The blue line represents the ground truth, the blue points indicate the test outputs, and the orange shaded area illustrates a one-sigma confidence interval while the orange line represents the predicted mean. From the diagram, it is evident that by collecting output values at test points, the posterior increasingly converges towards the true values.

During the fitting process of a Gaussian model, the choice of hyperparameters also affects the



(a) Test points = 2

(b) Test points = 8

Figure 2.3: Gaussion Progress Regression: The left figure shows the estimation of Gaussian Progress when there's only 2 sampling points; the right figure shows the same but with 8 sampling points. The posterior distribution becomes closer to the ground truth as more sampling points are added.

model's predictive outcomes. For example, length scale, choose of the default mean value and noise [RW06].

Length Scale. Length scale controls the smoothness of the function. A small length scale leads to a function that changes rapidly with small changes in input values, capturing more fine-grained variations in the data. A large length scale results in a smoother function that varies slowly. The figure 2.4 illustrates an example of the general impact of different length scales [RW06].

From the figure, we can see that a too small length scale may lead to overfitting, while a too large

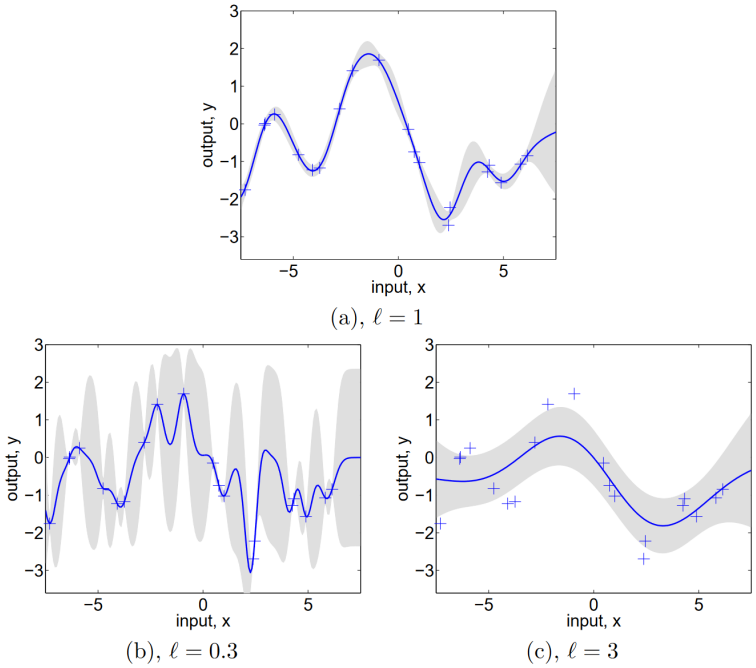


Figure 2.4: Comparison of length scales with length scale = 1, length scale = 0.3 and lengthscale = 1. The points are the sampled points, blue line is the estimation of mean and grey region is the 95% confident region of the GP Model [RW06].

length scale might result in a fit that is overly smooth and thus ineffective. Therefore, choosing an appropriate length scale plays a important role in the fitting.

The choose of lengthscale in our task and how it influences our posterior estimation is discussed in the experiment chapter.

Default Mean. The default value of mean presents when there's no or very sparse information in this region. That means the information from the neighborhood is hard to influence this region. The setting of default mean can influence the estimation of these region and have a influence of the sampling result [RW06].

Noise Level. Noise level means how much noise to expect in the data, allowing the GP model to distinguish between the noise and the underlying function [RW06].

After determining the probability density distribution, the next step is to choose an appropriate sampling strategy. As mentioned above, the searching space in our method is a high-dimensional space. One of the biggest issue is the curse of dimensionality. As the number of dimensions

increases, the volume of the space increases exponentially, making the available data sparse. This sparsity makes it difficult to obtain accurate estimates of the probability distribution and requires exponentially more data to achieve the same level of statistical reliability as in lower dimensions [Bet18].

Markov Monte Carlo (MCMC)). MCMC is a method to estimate posterior probability distribution by sampling, and it is especially suitable for complex distributions. Sampling directly from the posterior distribution, thereby eliminating the need for large sample sizes, addresses the issue of the curse of dimensionality to some extent. The idea is based on Markov Process and Markov Chain. A process is a Markov process if the probability distribution of the next state only relies on the current state. Markov Chain is used to describe the transition probabilities for the Markov process. MCMC approximates a distribution of interest by sampling from a corresponding Markov Chain. Each sampled example could be accepted or rejected and the Markov Chain is updated after each sampling. MCMC algorithm terminates when the Markov Chain converges [Has70].

However, traditional MCMC methods often perform a random walk, which can lead to inefficiencies and slow convergence, especially in high-dimensional spaces. HMC addresses this by using Hamiltonian dynamics to propose new states that are far from the current state, yet still have a high acceptance probability. This reduces such behavior, allowing the chain to move more rapidly across the state space [Bet18].

Hamiltonian Monte Carlo (HMC). HMC is a type of Markov Chain Monte Carlo sampler that mimics the kinetic behaviors of a physical system. Physical systems adhere to the theorem of conservation of kinetic and potential energy. When using an HMC sampler, we assume the probability of the posterior as potential energy. HMC randomly generates a trajectory thatis either accepted or rejected based on whether it adheres to the theorem. Compared to the classic random walk method, HMC is more efficient in sampling points that are far apart and can avoid getting stuck in local optima to a certain extent [Bet18].

HMC is an advanced Markov-chain Monte Carlo method for sampling from complex probability distributions. It leverages the concepts from physics to more efficiently explore high-dimensional spaces than traditional sampling techniques like Metropolis-Hastings or Gibbs sampling. Given these advantages, we employ the HMC method for sampling in the latent space in this thesis, which represents one of our contributions. HMC sampling follows these steps [Bet18]:

Initialization

Start by initializing the position (parameters) and momentum randomly. The momentum is typically drawn from a normal distribution. In this thesis, the initialized position is the position in the latent space.

Leapfrog Integration

HMC uses the leapfrog method to numerically integrate the Hamiltonian equations. This method updates the position and momentum variables in a way that approximately conserves the Hamiltonian, helping to ensure detailed balance and reversibility. The leapfrog integrator alternates between updating the momentum and the position variables, taking small steps at a time. In this step, the step size and the number of steps are hyperparameters that also affect the outcome of the sampling.

Update the velocity by a half-step

$$v\left(t + \frac{\Delta t}{2}\right) = v(t) + \frac{\Delta t}{2} \cdot a(t) \tag{2.22}$$

Update the position by a full step

$$x(t + \Delta t) = x(t) + \Delta t \cdot v\left(t + \frac{\Delta t}{2}\right) \tag{2.23}$$

Complete the velocity update with another half-step

$$v(t + \Delta t) = v\left(t + \frac{\Delta t}{2}\right) + \frac{\Delta t}{2} \cdot a(t + \Delta t) \tag{2.24}$$

Metropolis Update

After a Leapfrog Integration step, we follow the trajectory to arrive at a new position. The new state is accepted with a probability that depends on the Hamiltonian value of the previous and new position, which ensures that the sampling targets the correct distribution. In this thesis, the Hamiltonian value is the posterior distribution in the latent space. We accept the new state with the probability $A(x, x')$:

$$A(x, x') = \min\left(1, \exp(E(x) - E(x'))\right), \tag{2.25}$$

where E(x) and E(x') are the energies of the current and proposed states.

Iteration

This process is repeated many times to generate a chain of samples. The momentum is typically resampled at the beginning of each iteration to avoid undesirable random walk behavior and to ensure exploration of the space.

Burn-in Strategy. The burn-in strategy is a common technique in MCMC methods, used to ensure that the final samples are representative of the target distribution. During the initial phase of sampling, the algorithm may start from an arbitrary point in the parameter space, leading to biased samples. To address this, the first n iterations, known as the burn-in period, are discarded. This allows the Markov chain to converge to the target distribution before collecting samples for analysis [GP15].

2.5 Equation grammars

In this thesis, we have used grammar rules to generate equation datasets and calculate the prior of the equations. While using GVAE, the input equations are also decomposed into grammar rules, and the output equations are generated based on the same grammar rules. We have employed probabilistic context-free grammars for all grammars mentioned.

Probabilistic context-free grammars. A context-free grammar is a formal grammar. It consists of

a given set of rules to generate a set of words (called language) patterns. It is called context-free because its rules are regardless of context. Probabilistic context-free grammars (PCFGs) are an expansion of context-free grammar. They assign a probability to each production rule. The probability of a final derivation is the production of the probability of each production rule involved [Chi].

Consider a simple PCFGs designed to generate basic arithmetic expressions involving the addition and multiplication of integers. The grammar rules and their respective probabilities are defined as follows:

$$
\begin{aligned}
&S \rightarrow T\text{'} + \text{'}T \quad 30\% \\
&S \rightarrow T\text{'} \times T\text{'} \quad 30\% \\
&S \rightarrow T\text{'} / T\text{'} \quad 40\% \\
&T \rightarrow \text{'1'} \quad 10\% \\
&T \rightarrow \text{'2'} \quad 20\% \\
&T \rightarrow \text{'3'} \quad 70\%
\end{aligned}
$$

The probability P of generating the expression 1 + 3 is computed as:

$$P = P(S \rightarrow T + T) \times P(T \rightarrow \text{'1'}) \times P(T \rightarrow \text{'3'}) = 0.30 \times 0.10 \times 0.70 = 0.021 \quad (\text{or } 2.1\%) \quad (2.26)$$

3 Related Work

This chapter will provide an overview of the current research in the fields related to this thesis, namely symbolic regression, representation learning, and Bayesian reasoning.

3.1 Methods applied in the symbolic regression field

We can categorize symbolic regression methods in the following manner: regression-based methods, expression tree-based methods, physics-inspired and mathematics-inspired methods [MC24]. We have already introduced the classic genetic program method in the last chapter, which belongs to the expression tree-based method. Since that our method is more closed to the expression tree-based direction, we will introduce other two advanced expression tree-based methods in the following.

Reinforcement Learning (RL). RL is based on the idea of making optimal decisions. After making optimal decisions, a reward will be given to the agent from the environment. The agent will choose the next action based on the rewards in the past. Peterson et al introduced a deep reinforcement learning method used to find the best mathematical expression. A RNN model will be trained to represent the distribution of arithmetic expression. Then expressions will be sampled and evaluated based on dataset. The results of the evaluation will be seen as a reward and used to update the RNNs using the policy gradient algorithm [PLM+21].

Neural Network (NN). Neural Network (NN) method and especially Deep Learning methods are more and more used in the symbolic regression field. The Transformer Neural Network is a novel NN architecture in natural language processing to deal with sequential data, which was first introduced by Vaswani et al. (2017) [VSP+23]. A recent research from Kamienny et al combined transformer model and symbolic regression. An end-to-end method was introduced, which can directly predict the full mathematical expression and the included constants [KALC22].

Some examples of other types of methods are shown in the Figure 3.1 below. Among them, linear symbolic regression assumes the existence of linear relationships between data points, while nonlinear symbolic regression assumes that both linear and nonlinear relationships may exist, typically leveraging neural networks to learn the relationships within the data. The AI Feynman method used the neural network to find the simplicity in the data, like symmetry and separability. It also showed that by using physic units in dimension analysis, the discovery rate of equations is improved [UT20].

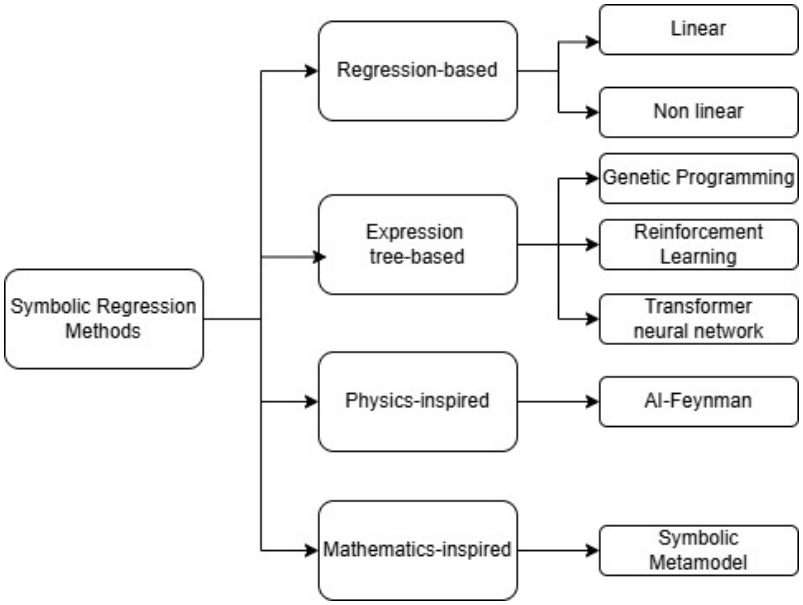


Figure 3.1: Type of symbolic regression methods introduced in [MC24]. The methods are categorized into Regression-based, Expression tree-based, Physics-inspired and Mathematics-inspired classes. For each class, at least one example is given.

3.2 Deep Learning architectures for advanced symbolic regression

A major challenge in symbolic regression is exploring the infinite search space. An approach discussed in the literature for other data types is to utilize a structured latent space. In this thesis, we plan to apply this strategy to mathematical equations, searching within the structured latent space. This approach will enable us to estimate the probability distribution across this structured search space.

The idea and concepts of Autoencoder and VAE are already introduced in chapter 2. On this basis, some VAE models with specific structures designed for symbolic regression have also been proposed. Using these structural features, some specific problems can be solved, for example, RNN, which is discussed in chapter 2, can be used to time series analyse like sentence prediction. Grammar Variational Autoencoder, which includes a mask function to choose the syntactically correct rules [KPH17].

Variational autoencoders can also be used to learn the latent structure of the mathematical expression [PLB+23]. S. Popov et al. introduced a method that refines a set of equations by progressively filtering out points with higher Mean Squared Error rankings between actual and predicted values. This refined set of equations is then used to train a VAE. First, some formulas will be generated randomly to pre-train a VAE model; then the formula set will be optimized by deleting formulas that don't meet predicate conditions, the formula set will then be optimized by deleting formulas that do not meet specific conditions, and the refined formula set is used to train the VAE again until it terminates. Although the method takes into account the data in the dataset and improves the

recovery rate, it does not incorporate prior knowledge in the form of probabilities [PLB+23].

Character Variational Autoencoder Cho et al. proposed the use of RNNs within VAE for predicting the next output character. RNNs gather information from previous inputs to influence the current input and output. Unlike traditional deep neural networks, the outputs of RNNs depend on prior elements in the sequence. Applying this to symbolic regression problems allows for the natural consideration of previous inputs, making the next output character more plausible. As shown in the figure below, when making predictions for the output, not only the current input but also previous inputs and outputs are considered [CVG+14].

Although the Character VAE can simultaneously consider past and current inputs, its character-by-character output at each step may result in syntactically invalid outputs. To address this issue, M. J. Kusner and colleagues proposed a grammar variational autoencoder, which ensures that the output at each step of the variational autoencoder is not a character but a valid grammatical rule. The construction of outputs will based on the grammatical rules produced [KPH17].

Grammar Variational Autoencoder (GVAE). GVAE is a VAE for data whose syntactical structure can be constrained by grammar. For instance, equations have an inherent syntax that needs to be respected. Arbitrary strings of mathematical symbols are not necessarily valid equations, the high-level idea is to exploit a grammar in both the encoder and decoder to ensure syntactically valid outputs after training. Unlike a character-based VAE, a GVAE does not predict the next character, but instead predicts the next rule based on the current character. During the encoding process, each rule will be converted to a 1-hot vector and then mapped to the latent space. When generating new outputs, a sampled latent code will be converted to logits for all rules of the grammar and the production, i.e. a sequence of matching rules, will be sampled according to the associated probability [KPH17].

M.J.Knuser et al. demonstrate the operation of this method using a set of chemical formulas, SMILES, as an example in their article. The encoder first parses the input SMILES molecular formula into a parse tree and then extracts the grammatical rules from the parse tree. Subsequently, these grammatical rules are embedded as one-hot vectors, which are used for training the VAE. The figure below illustrates this process. During decoding, based on the current symbol, invalid

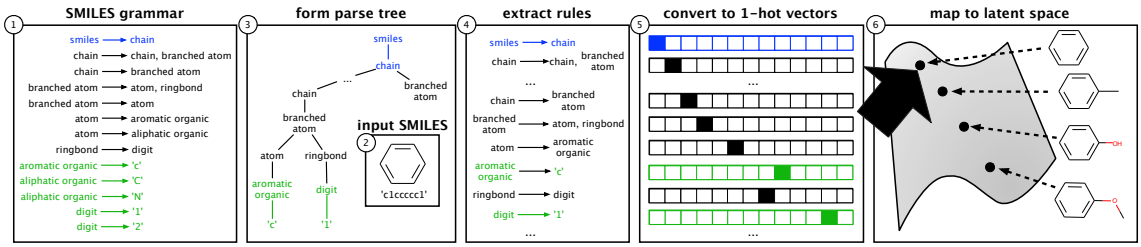


Figure 3.2: The encoder process of the GVAE: 1. generate form parse tree with the given grammars 2.extract the used rules from the expressions 3.convert them to 1-hot vectors 4. process the vectors through encoder to the latent space [KPH17].

rules are excluded. This entails initially selecting rules that start with the current symbol, and then choosing the next rule from these. This practice ensures that the results are syntactically valid. The figure below demonstrates this specific approach [KPH17].

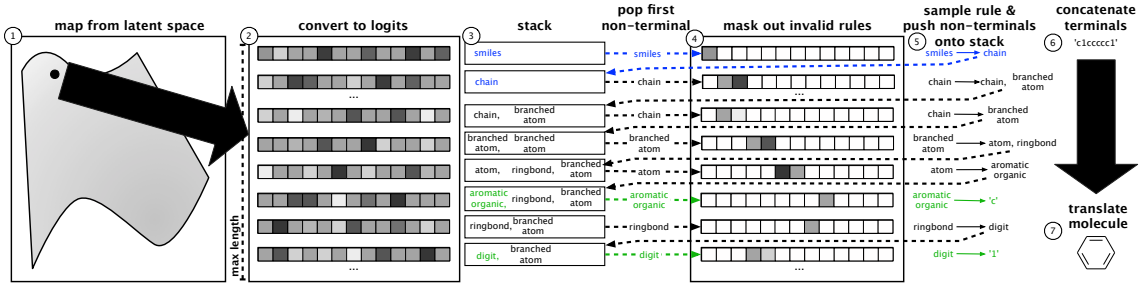


Figure 3.3: The decoder process of the GVAE: 1. convert the data point in the latent space to logits 2.from start to end, choose the valid rules which are not masked out by the masks 3. build the expression based on the chosen rules [KPH17].

In this thesis, the proposed model will learn the inherent structure present in mathematical equations and will be able to generate new syntactically valid equations.

3.3 Efficient sampling

As discussed in the background part, deal to the curse of dimension, sampling in high-dimensional space often faced with the problem of how to sample efficiently. Gaussian Process can describe the probability distribution flexibly with uncertainty and Hamilton Monte Carlo sampling algorithm can move more rapidly and accelerate the sampling space [RW06]. Based on these characteristics, a recent study proposes to combine the two for fast and efficient sampling.

Gaussian Process - Hamilton Monte Carlo (GP-HMC). GP-HMC was first presented by Rasmussen et al. which applies HMC to do sampling and assume the distribution of energy is a Gaussian Process [RW06]. That makes it combine the advantages of sampling in Gaussian Process and HMC Sampler. It consists of two phases, the exploratory phase and the sampling phase [Ras03].

Exploratory Phase. In this phase, enough information will be gathered using the HMC sampler, under the assumption that the distribution of energy function is a GP. The HMC method will follow the steps introduced in chapter 2. In each iteration, a sampling is performed, and if the new sample point is accepted, the Gaussian process model will be updated based on the new set of sample points [Ras03].

Sampling Phase. After the exploratory phase, a GP Model will be built based on the gathered information and will not be updated during the sampling phase. Then, the desired samples will be generated using the GP Model from the first phase. Since the sampling is based on the GP Model, it will be more cost-effective and easier to evaluate. By using the HMC sampler, the sampling process will be sped up [Ras03].

4 Methodology

4.1 Problem definition and datasets

This Chapter will introduce the method and the process of our method. First, we will define the problem, our goal and datasets that we used in this thesis.

Problem. Given a data points dataset with observed data, $D = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i are inputs and y_i are the observed outputs, the task is to find an expression e that best fits the data with $e(x_i) \approx y_i$. However, the infinite search space of mathematic symbols makes it impractical to apply exhaustive search methods [MC24]. At the same time, only considering the observed data points makes it face the problem of overfitting. Our method aims to first learn a structured latent space, which represents the structure of expressions but a continuous space. In the latent space, our method will consider both observed data points and prior knowledge by applying Bayesian Reasoning. Our method is able to solve this problem while avoiding the exhaustive search and overfitting.

Datasets. In this thesis, we have two kinds of datasets, one is the **expressions dataset**, which includes different expressions, and the other is the **data points dataset**, which includes the observed data. We will introduce the specific expressions dataset and data points dataset in the next chapter, and Appendix A contains all the expressions datasets used in this thesis. Figure 4.1 shows an example of a data points dataset. The data points are the observed data $D = \{(x_i, y_i)\}_{i=1}^{N}$, our goal is to find the target function, which describes the relationship between the inputs and the outputs. In this thesis, we will use part of the data points as a training dataset, which are the green points in Figure 4.1. The other part of data points will be used as test set and used for evaluation.

The rest of this chapter is as follows: Section 2 introduced the process overview of our method. Section 3 discusses how Bayesian theory is applied in this thesis. Section 4 discusses how to learn a structured latent space. It introduces the deep learning methods used in this thesis, namely Character VAE and GVAE. Section 5 discusses how sampling is conducted in the latent space. For the methods used in this thesis, this section briefly introduces the parameters and hyperparameters involved, with a detailed analysis to be conducted in the ablation study section of the next chapter.

4.2 Process overview

This section introduces the overview of the the process of this thesis. As shown in Figure 4.2, our method includes three parts: learn a structured latent space, apply Bayesian Reasoning in the latent space and sampling in the latent space. Beside them, our work also includes the evaluation part which compares our method with the classic approach in the symbolic regression field.

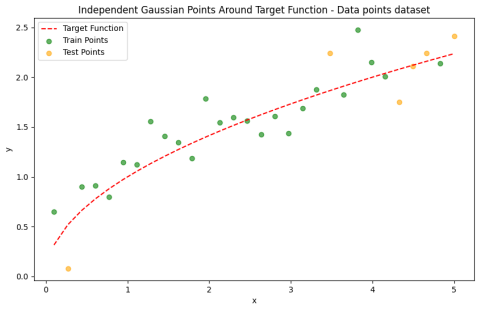


Figure 4.1: An example of data points dataset. The red dashed line represents data points that follow the objective function. The green and yellow dots represent the actual observed data, where the green dots are used for the training set and the yellow dots are used for the test set.

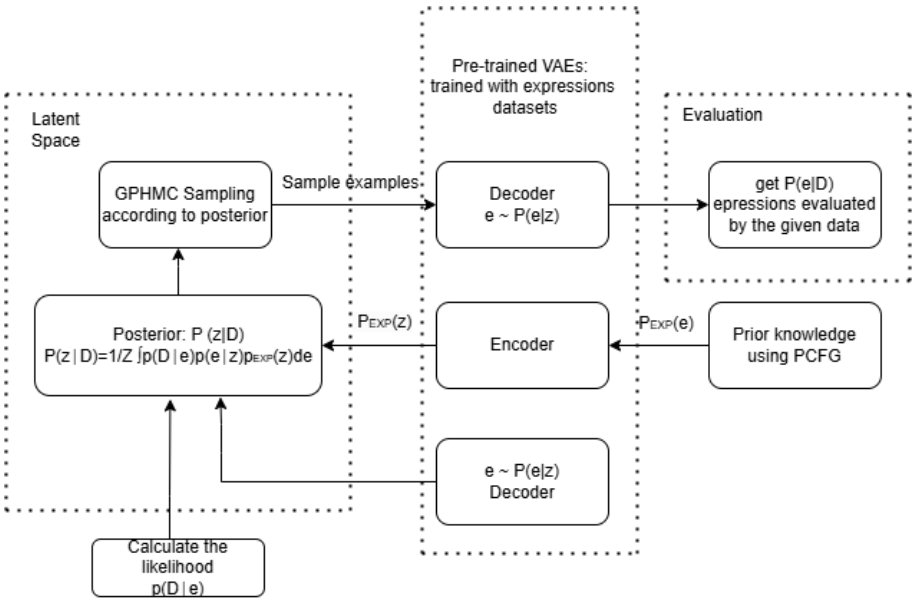


Figure 4.2: Process Design of Bayesian Symbolic Regression using VAE and GP-HMC. The trained VAE part means the decoder and encoder are from the pre-trained VAE . The latent space part means the process is happened in the latent space.

As introduced in Section 4.1, our goal is to find the expression that best describes the observed data points. While some widely used symbolic regression methods, like Genetic Programming, search in a discrete space, we first transform the search space into a structured continuous space, which is a learned latent space in our method. To obtain a learned latent space, we first define the dataset and the training model. We use PCFGs [Chi] to generate a mathematical expressions dataset, since it allows us to parse equations into parse trees and use the production rules as input. Then, we use the expressions dataset for model training. The choice of model type, architecture, and other hyperparameters will be discussed in the next chapter. Once the model training is completed, the model and latent space are saved and used as the trained VAE s and latent space shown in Figure

4.2. To choose a position in the latent space, we first need to build the distribution of the posterior in the latent space, which refers to how likely the expressions decoded from this position best fit the observed data. To avoid the problem of overfitting, we have used Bayesian Reasoning in the latent space. We have obtained the likelihood by calculating the joint likelihood that the observed data are from the equation decoded from this position in the latent space. Prior knowledge has been transferred from the expression level to the latent space using the trained encoder. Sampling in the latent space is based on the posterior. We have applied the GP-HMC method to first build the posterior distribution in the latent space and then perform the sampling.

To evaluate our method, we have compared the result with other two methods, Genetic Programming [MC24] and SInDy [FKK+21]. The result and discussion are in the Chatper 5.

4.3 Bayesian Reasoning in a learned structured latent space

The specific problem addressed in this research is: given some observed data points $D = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i are inputs and y_i are outputs, how to find an expression e such that $e(x_i) \approx y_i$ in a probabilistic way:

$$P(D|e) := \prod_i \mathcal{N}(y_i|e(x_i), \sigma), \tag{4.1}$$

yields a reasoning problem using Bayesian theorem

$$P(e|D) = \frac{1}{Z} P(D|e)P(e), \tag{4.2}$$

where Z is a normalization coefficient. The explanation of symbols in this thesis can be found in Table 4.1.

4.3.1 Learned Latent Space

However, the intrinsic structure of the mathematical formulas has not been adequately comprehended by using 4.2. Additionally, the search for equations is discrete and non-differentiable. Therefore, this thesis proposes using Bayesian reasoning on potential symbolic solution candidates for a regression problem in learned, structured latent space. The latent space is learned by training a Variational Autoencoder with generated mathematical expressions. In this thesis, we employ two types of models discussed in the previous section: the Character VAE and the GVAE. In each step, the Character VAE outputs the prediction of the symbol at the position. The GVAE method expands upon the VAE by incorporating an additional step that uses masks to filter out invalid grammar rules. By probabilistically mapping to grammatical rules, enabling the generation of data that adheres to specific grammatical structures [KPH17]. The training processes of the two VAEs are both introduced through pseudocode in the next section. After training, the latent space, the model itself,

Symbol	Explanation
D	The observed data to regress, $D = \{(x_i, y_i)\}_{i=1}^{N}$
e	Expressions
z	Latent variable in latent space
$P(e)$	Prior probability of an expression, here is domain expert prior
$P(z)$	Prior probability of specific positions within the latent space
$P(D\|z)$	Probability of generating the observed data from the expressions decoded at a given position in the latent space
$P(z\|D)$	Posterior probability over latent codes given the observation data
$P(e\|z)$	Probability of generating an expression from a given position in the latent space
$P(D\|e)$	Probability of generating the observed data from a given expression
P_θ	Parameters of decoder
q_ϕ	Parameters of encoder
$g(z)$	Posterior of latent space estimated by Gaussian Process

Table 4.1: Explanation of Symbols used in this thesis.

and the related parameters are all saved. At this point, each point in the latent space can be decoded into equation expressions. In other words, the discrete search space of all possible equations is now transformed into a continuous latent space. Then we can transfer this problem mentioned before to the latent space by solving:

$$P(z|\mathcal{D}) = \frac{1}{Z} \int p(\mathcal{D}|e)\, p(e|z)\, p_{\text{EXP}}(z)\, de, \tag{4.3}$$

where $p(e|z)$ is obtained by the decoder and $p(z)_{\text{EXP}}$ is mapped through the pre-trained encoder from the prior knowledge. $p(\mathcal{D}|e)$ can be calculated by 4.1. A depiction of the process can be found in Figure 4.3

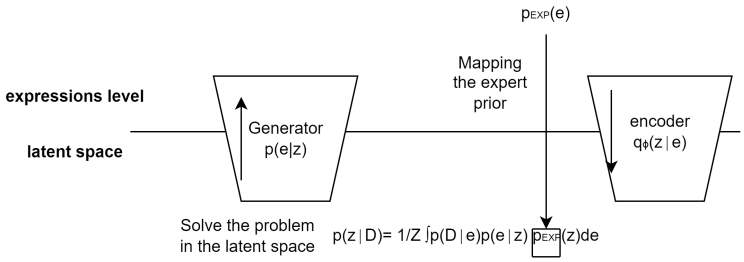


Figure 4.3: The upper area of the image represents the expressions level, while the lower area represents the latent space level. It illustrates the problems that need to be solved at the expressions level and the latent space level, respectively. The direction of the arrows indicates the direction of information transfer.

The expressions of prior knowledge are generated by using probabilistic context-free grammar introduced in the related work. To approximate the expert prior $P(e)$ in latent space $P(z)$, the following generation model is used:

$$\begin{aligned} P_{\text{EXP}}(z) &= \int p(z|e)\, p(e)_{\text{EXP}}\, de && \text{- Reasoning} \\ &\approx \int q_\phi(z|e)\, p(e)_{\text{EXP}}\, de && \text{- approximate with trained encoder} \\ &= \mathbb{E}_{e\sim p_{\text{EXP}}(e)}\left[q_\phi(z|e)\right] \\ &\approx \frac{1}{M}\sum_{i=1}^{M} q_\phi(z|e_i), \quad e_i \sim p_{\text{EXP}}(e) && \text{- approximate by sampling} \end{aligned} \tag{4.4}$$

Once we obtain the prior in the latent space, we can utilize it for Bayesian reasoning within the latent space.

4.3.2 Bayesian Reasoning

Then we estimate the posterior in the latent space with a GP model, which is part of the GP-HMC method. Posterior in one position in this space can be derived as follows:

$$\begin{aligned} g(z) &:= \int p(D|e)\, p(e|z)\, P_{\text{EXP}}(z)\, de \\ &= \mathbb{E}_{e\sim p_\theta(e|z)}\left[p(D|e)\right] P_{\text{EXP}}(z) \\ &\approx \frac{1}{M}\sum_{i=1}^{M} p(D|e_i) P_{\text{EXP}}(z), \quad e_i \sim p_\theta(e|z) \end{aligned} \tag{4.5}$$

To estimate the probability distribution in the latent space, we need to first sample some points in the latent space and get their function values g(z) using Equation 4.5. We use HMC to sample from

$$z \sim \frac{1}{Z} g(z), \tag{4.6}$$

where Z is the coefficient constant. After enough examples are sampled, we fix the GP model based on the sampled points $(z, g(z))$, then do HMC sampling again to get some sampled latent space positions z. The sampled latent space positions z can be transferred from latent space to the level of the equations as shown in Figure 4.2.

More details about the sampling algorithm and reconstruct points in the latent space to expression level is discussed in the following sections.

4.4 Learn a structured latent space

Constructing a high-quality structured latent space is a critical step in this process. In this section, we discuss the methodologies and significance of structuring latent spaces within neural network architectures. The concept of a latent space refers to an abstract, multi-dimensional representation

of the data, engineered to capture its inherent patterns in a compressed format.

We employ advanced machine learning techniques such as VAE to model these spaces. During the training process of a VAE, a continuous latent space is formed. Furthermore, the use of the reparameterization trick in VAE introduces noise, making the distribution in the latent space differentiable and naturally incorporating uncertainty. In this section, we introduce various types of VAEs in the model structure level also discuss different types of neural network layers.

4.4.1 Character Variational Autoencoder

Character VAE is a kind of VAE that selects an appropriate model structure based on the characteristics of the symbols it processes. The concept of VAE has been already introduced in the background [KW22]. Here we focus on the characteristics of Character VAE.

As the name suggests, the input of Character VAE is characters, such as letters or other symbols. This kind of task has a specific characteristic: the current output depends on not only the current input but also the previous input and output. This perfectly aligns with the characteristics of RNN, which can remember the previous state and take it into consideration for the current output. By incorporating RNN layers into the VAE architecture, we can create a Character VAE that handles symbols with sequentiality [CVG+14].

In this thesis, we employed LSTM units within the RNN layers of both the encoder and decoder. The concept of RNN and LSTM is introduced in Chapter 2.

Since our input consists of symbols, which cannot be used directly for training, we need to convert them into vectors that can be used for training. Typically, a dictionary is employed to map each symbol to an index [CVG+14]. In this thesis, all the symbols in the input expressions are stored in a dictionary, and an index is assigned to each of them. The vocab dictionary looks like the following:

After conversion, all equations are split by symbols, with each symbol converted into an index for storage. All equations are padded with a padding symbol to match the length of the longest equation in the dataset, facilitating batch processing by the model.

Model Architecture

A distinctive feature of the Character VAE is the use of RNN layers within both the encoder and decoder. As mentioned previously, each symbol of the input mathematical expression is transformed into the index which is suitable for learning and training purposes. In practical input scenarios, all symbols of an expression are inputted simultaneously, and their sequence is processed as the temporal order in the RNN layers. Given that RNN layers simultaneously consider the characteristics of both previous and current inputs, the output of each symbol in the Character VAE is generated based on the prior inputs, aligning with our fundamental requirements for symbolic learning [CVG+14].

Figure 4.4 shows a general structure of applying RNN Layers in the encoder and decoder. The X

Table 4.2: Vocabulary Index Table for Character VAE

Index	Token
0	'<pad>'
1	'sin'
2	'1'
3	')'
4	'*'
5	'2'
6	'+'
7	'x'
8	'exp'
9	'/'
10	'('
11	'cos'
12	'3'
13	'<start>'
14	'<end>'

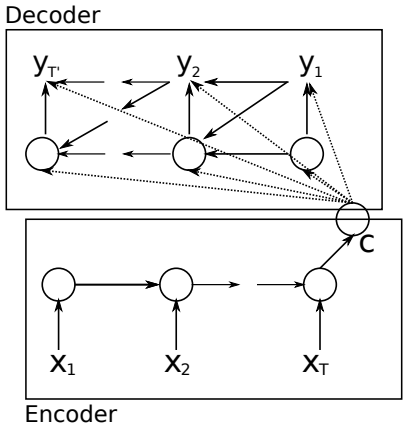


Figure 4.4: An illustration of the proposed RNN Encoder-Decoder, X as inputs and Y as outputs. C represents the information in the latent space. The arrows show the influence and sequence between the data [CVG+14].

represent the input symbols, while X_T is the current input. The Y are the outputs symbols and Y_T means the current output. The circles are the hidden states.

For example, in our problem, when our input is '$x + x \times x + \sin(x)$', the model first learn it in the encoder character by character. The first symbol x as the X_1, a hidden state is created according to the current input. In the next step, the both symbol + and the previous hidden state are used to update the new hidden state until the whole expression is processed. When generating the reconstructed output, a hidden state is first generated from the current point in the latent space, which is the c in the Figure 4.4 and produces the initial symbol '⟨start⟩'. Combining the current hidden state and the

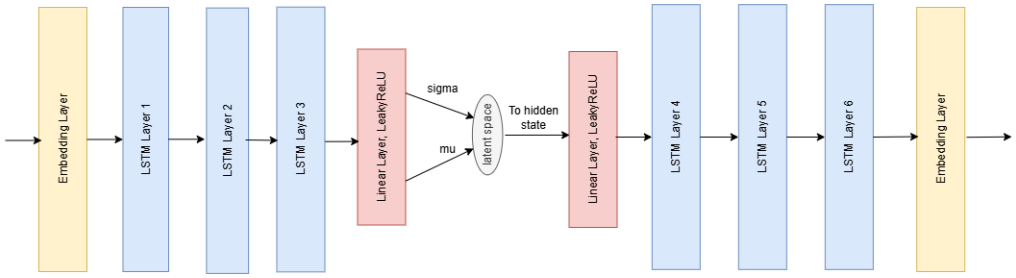


Figure 4.5: Model Architecture of Character VAE, the input is processed along the direction of arrows through each layer. μ and σ are the parameters while applying reparameter trick.

'⟨start⟩' symbol, the model will generate the next symbol 'x' as y_1, and update the hidden state containing all current output information. Then, based on the new hidden state and output, the model generates the next output symbol '+' as y_2.

Figure 4.5 shows the structure of our model (take three LSTM layers as an example), it includes an **Encoder**, latent space and a **Decoder**. The encoder includes an embedding layer, three LSTM layers, and a linear layer sequentially. The decoder includes a linear layer, three LSTM layers, and a linear layer sequentially. The input is forwarded through the encoder and compressed into the latent space after the parameterization process. In our method, each equation is compressed to a data point in the latent space.

The functions and roles of each layer are different. In the encoder, the embedding layer primarily transforms large discrete inputs, here is symbolic indices, into fixed-size continuous dense vectors, providing richer feature representations for subsequent processing in deep learning models. The LSTM layers are to handle sequential data by maintaining a memory of past inputs which allowing it to capture long-term dependencies and context within the data. The multi-layer LSTM architecture can help the model capture more complex patterns and relationships. For instance, different layers can learn different levels of sequential information, thus enabling a deeper representation of data compared to a single-layer LSTM [SM19]. The linear layer is for the reparameterization process, which is introduced in Chapter 2. This layer generates the mean and log variance of the parameter distribution in the latent space.

The construction of the decoder generally mirrors the encoder in reverse order, aiming to enable the points in the latent space to inversely generate equations similar to the input equations. Directly after the latent space is an embedding layer. It is used to embed the input token, which is the start symbol in our task. After the embedding layer, an linear layer transforms the state from the latent space into a hidden state, which acts as the input in the LSTM layers. As the core component of the decoder, the LSTM layers are responsible for generating the output sequence. It starts with the initial representation provided by the latent space or an input start vector and progressively generates outputs at each time step, ultimately producing the entire output sequence. The final linear layer that converts the output of the LSTM into the dimensionality of the vocabulary size. It helps by calculating the loss function and getting the predicted tokens from the output of Character VAE. Outputs from this layer are the logits, which present information about the predicted symbol at each position in the sequence.

The output logits are used for the calculation of the loss function, and when generating actual mathematical equations, we typically select the symbol with the highest logits value at each position.

As introduced in the Chapter 2, loss Function is a differentiable function used to measure the performance of a model and guide the optimization process. During the training of the Character VAE, we define the loss function following the general choice of VAE.

Reconstruction Loss measures the difference between the input sequence and the reconstructed sequence.

$$\mathcal{L}_{\text{recon}} = -\sum_{i=1}^{N}\sum_{j=1}^{V} x_{i,j} \log \hat{x}_{i,j}, \tag{4.7}$$

where N is the number of samples, V is the vocabulary size, $\hat{x}_{i,j}$ represents the predicted probability distribution, and $x_{i,j}$ is the target distribution.

KL Divergence Loss regularizes the latent space to follow a normal distribution, encouraging a meaningful latent representation.

$$\mathcal{L}_{\text{KL}} = -\frac{1}{2}\sum_{i=1}^{N}\left(1 + \log \sigma_i^2 - \mu_i^2 - \sigma_i^2\right), \tag{4.8}$$

where μ_i and σ_i are the mean and standard deviation of the latent space.

Total Loss combines the reconstruction loss and KL divergence, with β as a weighting factor for tuning the relative importance of the KL term.

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{recon}} + \beta \cdot \mathcal{L}_{\text{KL}}, \tag{4.9}$$

where β is a hyperparameter used to balance the weight of the KL divergence term. The influence of β is discussed in the next chapter.

The training process follows the Algorithm 4.1 [CVG+14]. Upon completion of the training, each input is mapped to a point in the latent space, and each point in the latent space can be decoded into an output through the decoder. The training process is finished once the parameter converges.

The figure 4.6 visualizes the mapping of a dataset containing five mathematic equations in a two-dimensional latent space. From the visualization, we can observe that similar mathematic equations are mapped to nearby regions. The subsequent Bayesian inference is conducted within the structured latent space learned during this step.

4.4.2 Grammar Variational Autoencoder (GVAE)

In our method, we can apply different kind of VAEs to obtain a learned latent space. GVAE introduced another way to process the input and generate outputs. Especially, GVAE parses equations based on the context-free grammar used to generate them and maps the resulting rules to a dictionary rather than mapping the characters. An example of the dictionary is shown in the Table

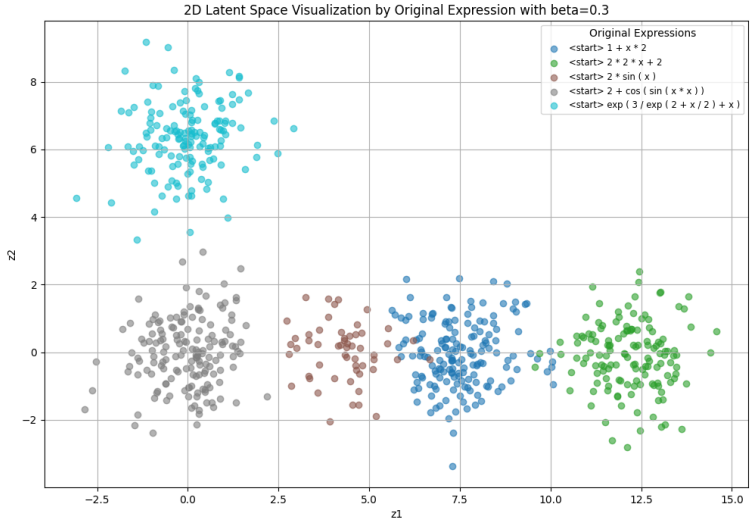


Figure 4.6: Learn latent space with dataset including 5 expressions. The right upper shows the 5 expressions. $z1$ and $z2$ are the two dimensions in the latent space.

Algorithm 4.1 Training the Character VAE

Input: Equation dataset $\{E^{(i)}\}_{i=1}^{N}$, acceptable symbols // in this thesis like exp, sin, cos...
Output: Trained Character VAE model $p_\theta(E|z)$, $q_\phi(z|E)$ and latent space z // parameters for encoder and decoder
while Character VAE not converged **do**
    Select element: $E \in \{E^{(i)}\}_{i=1}^{N}$ (or minibatch)
    Encode: $z \sim q_\phi(z|E)$ // do it by using parameters from last step
    Decode: given z, compute logits $F \in \mathbb{R}^{T_{\max} \times K}$ // $T_{\max}$is the largest length of the sequences, K is the size of character dictionary
    for $t = 1$ to $T_{\max}$ **do**
        Compute $p_\theta(E_t|z)$ via RNN Layers in the decoder
    end for
    Update θ, ϕ using estimates $p_\theta(E|z)$, $q_\phi(z|E)$, via gradient descent of loss function
end while

4.3. Take the rules in the Table 4.3 as an example, when the input expression is $'x' +' 1'$, the data prepossessing part parses the expression to the rules $S \rightarrow S' +' T$, $T \rightarrow' x'$ and $T \rightarrow' 3'$. Then it will find the corresponding index and use the index to train the model.

Another characteristic of GVAE is the use of a mask at each step when generating new outputs with the decoder. Mask is a filter that only choose the rules for the next step based on the current output. For example, if the output of the current step is $'S'$, the mask will mask out all rule that not start with $'S'$ for the next output. The rules which are masked out are not considered in the step and masks are generated at each step. This rule-based generation ensures that all outputs comply with semantic rules [KPH17]. In this thesis, this feature should ensure that all generated equations are

Table 4.3: Grammar Rules Table for GVAE

Index	Rule
0	S → S '+' T
1	S → S '*' T
2	S → S '/' T
3	S → T
4	T → '(' S ')'
5	T → 'sin' '(' S ')'
6	T → 'cos' '(' S ')'
7	T → 'exp' '(' S ')'
8	T → 'x'
9	T → '1'
10	T → '2'
11	T → '3'

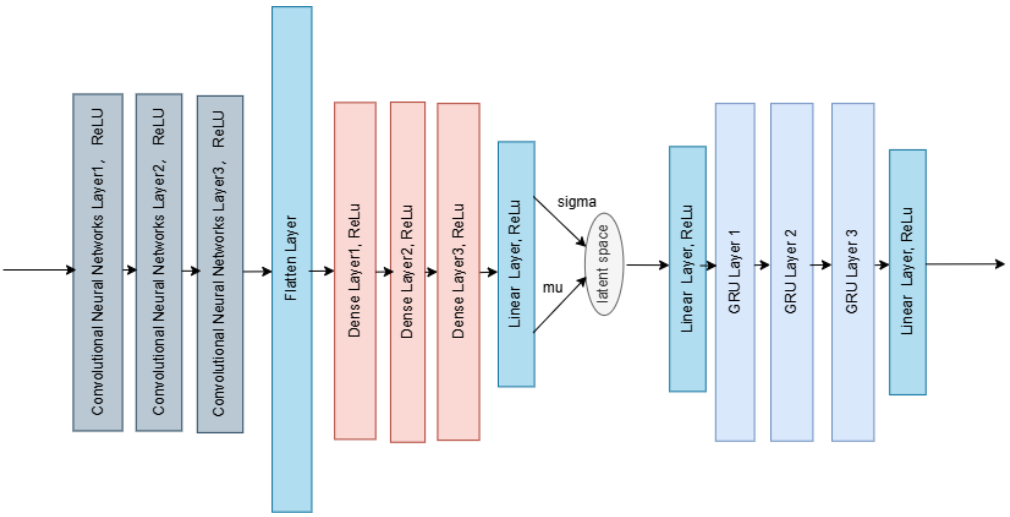


Figure 4.7: Model Architecture of GVAE, the input is processed along the direction of arrows through each layer. μ and σ are the parameters while applying reparameter trick.

semantically valid, significantly enhancing the efficiency of sample generation from the latent space.

Model Architecture

The basic model architecture of GVAE is consistent with the previously introduced Character VAE, consisting of an encoder and a decoder in sequential order. Figure 4.7 shows the model structure of GVAE in this thesis. The configuration of the neural network layers follows the settings described in [KPH17]. The overview of GVAE model architecture is shown in the Figure 4.7.

Encoder includes three convolutional layers, a flattening layer, and two dense layers. The input first forward through the CNN layer. The CNN layers can automatically extract local patterns and

relationships from the input equations. This helps the model identify meaningful structures, such as subexpressions.After the CNN layers, the flattening layer converts the multi-dimensional output of the convolutional layers into a one-dimensional vector, facilitating its subsequent processing by the dense layers. At the end, the dense layer integrates all the features from the flattening layer through fully connected operations and ultimately maps them to the dimension of the size of the rules dictionary for output prediction.

Decoder includes a linear layer, three GRU layers, and a linear layer sequentially. The two linear layers have the same purpose as in the Character VAE while the GRU layers have the same consideration and use as in the Character VAE.

Loss Function In this thesis, the loss function of GVAE contains the KL Divergence Loss and reconstruction loss, the same form as in the Character VAE.

The following Algorithm 4.2 summarizes the training procedure for the proposed model, highlighting the core steps and their interactions. The explanation considering our method is included [KPH17].

Algorithm 4.2 Training the Grammar VAE

Input: Equation dataset $\{E^{(i)}\}_{i=1}^{N}$, Context-Free Grammars // here we use mathematical expressions generated by these PCFGs
Output: Trained GVAE model $p_\theta(E|z)$, $q_\phi(z|E)$ and latent space z // parameters for encoder and decoder
while GVAE not converged **do**
    Select element: $E \in \{E^{(i)}\}_{i=1}^{N}$ (or minibatch)
    Encode: $z \sim q_\phi(z|E)$ // do it by using parameters from last step
    Decode: given z, compute logits $F \in \mathbb{R}^{T_{\max} \times K}$ // $T_{\max}$is the largest length of the sequences, K is the size of rules dictionary
    for $t = 1$ to $T_{\max}$ **do**
        Compute $p_\theta(E_t|z)$ via
$p(E_t = k \mid \alpha, z) = \frac{m_{\alpha,k} \exp(f_{tk})}{\sum_{j=1}^{K} m_{\alpha,k} \exp(f_{tj})}$ // mask present if the rule possible to be used in this position, f_{tk} is the (t, k)-element of the logit matrix F
    end for
    Update θ, ϕ using estimates $p_\theta(E|z)$, $q_\phi(z|E)$, via gradient descent of loss function
end while

By applying masks in the decoder, GVAE only generates the syntactically correct mathematic expressions. However, the output of Character VAE is not guaranteed to be syntactically correct [KPH17].

4.5 Sampling the posterior in latent Space

To sample in the latent space, we must estimate the probability distribution within it. Due to the complexity of integrating distributions in high-dimensional spaces, calculating the posterior distribution of points in the latent space using integration becomes impractical. However, we can

compute the posterior distributions of some sampled points in the latent space and use these values to estimate the overall posterior distribution of the entire latent space. According to the Bayesian theorem and 4.5 mentioned in the previous section, the estimation of the posterior distribution is composed of two parts: the likelihood $p(D|e_i)$ and the prior $P_{\text{EXP}}(z)$.

The calculation of the likelihood is based on our existing dataset, which includes true observational values. For each data point, we assume that its distribution follows a Gaussian distribution with the predicted mean $f(x_i)$, denoted as $N(f(x_i), \sigma)$. The probability of the actual observer y_i belonging to this distribution is represented by the probability density of y_i within this distribution.

$$p(y \mid x) := \prod_{i=1}^{n} p(y_i \mid x_i), \quad y_i \sim \mathcal{N}(f(x_i), \sigma) \tag{4.10}$$

Calculating this for all data points in the dataset yields the likelihood of the mathematical expression for that dataset.

The calculation of the prior at the expression level is based on the probabilistic context-free grammars, an example of generating an expression is shown in the background and equation 4.1. Then, we need to transfer it to the latent space level. The process follows the following steps:

1. Generate expressions dataset based on the PCFGs. The PCFGs are defined according to the scientific knowledge in the specific domain.
2. Use the expressions dataset as input, feeding it to the pre-trained encoder. For each expression, we can get two parameters μ and σ, which are used for the reparameter trick for the latent space.
3. Since we assume the latent space is normally distributes, we can obtain the probability in the latent space given a

By applying Bayesian theorem, we can multiply the likelihood and prior and get the posterior.

4.5.1 Build the Posterior distribution

In this thesis, we assume that the posterior distribution in the latent space conforms to a Gaussian process. The basic definition of a Gaussian process has been discussed in the previous section. According to this definition, a Gaussian process is a non-parametric process, meaning it does not require a fixed structure and can adapt to the complexity of the data autonomously. Additionally, the stochastic nature of Gaussian processes introduces uncertainty, which helps to prevent overfitting to some extent. In practical experiments, assuming that the data conforms to a Gaussian process can also accelerate the convergence process [RW06].

In practice, we first need to define the mean function and the kernel function of the Gaussian Process. The mean function represents the default value of a point when there is no other information available about its neighborhoods. The kernel function defines the covariance between two points based on the similarity of their inputs. Common kernels include the squared exponential kernel, the rational quadratic kernel, and the Matérn kernel [RW06]. Additionally, the Gaussian Process also involves other hyperparameters such as length scale, default mean and noise level. The general influence of the have been introduced in the Chapter 2, the influence of our problem and how they

impact the sampling results are discussed in the Chapter 5 [RW06].

By first sampling some points in the latent space, do the Bayesian Reasoning discussed before, we can obtain the posterior distributions corresponding to these points. Use the information of these points as input, the distribution of GP Model can be estimated. When new data points are added, the GP Model will be updated, the prediction for other points in the GP are also updated accordingly. Furthermore, when a sufficient number of points are used to fit the GP, the distribution of the predicted values often closely approximates the true distribution [RW06].

4.5.2 Sampling Algorithm

Latent spaces are often high-dimensional. Sampling in high-dimensional spaces frequently encounters many challenges, such as low sampling efficiency when the dimensionality is too high and the sampling space becomes excessively large [Bet18].

4.5.3 GP-HMC Algorithm

Sampling in the latent space requires us to quickly and accurately construct the distribution of the posterior probability in the latent space. Therefore, in this thesis, we use the GP-HMC algorithm proposed by C. E. Rasmussen et al., which performs the construction of the GP distribution and HMC sampling iteratively. Once the estimated GP Model is stable, we no longer update the GP model but continue sampling on the established GP model. It includes two phases, an exploratory phase and a sampling phase [RW06].

In our method, we use latent space as the searching space, posterior as energy function in HMC. The workflow of and setting details of GP-HMC algorithm in our task is illustrated by the pseudocode Algorithm 4.3.

Exploratory Phase

During the exploration phase, our primary task is to rapidly construct the posterior distribution in the latent space. In this thesis, our potential energy during the exploration phase is given by:

$$E_{\text{pot of exp}} = \mu - \sigma, \tag{4.11}$$

where E_{pot} is the potential energy, μ and σ are the mean and variation of posterior in the latent space, respectively. This will encourage the algorithm to explore areas with greater uncertainty more extensively.

The HMC sampling process follows the procedure mentioned in the previous section, specifically using the leapfrog method to update positions and employing the Metropolis rule to decide whether to accept new sample points. However, during the exploration phase, each time a new sampling point is accepted, the GP Model of the posterior distribution is updated, meaning that the estimated values for each point are revised. As the number of sampling points increases, the GP Model will

increasingly approximate the true values.

Sampling Phase

Once the exploration phase is complete, we consider the GP Model to be stable and close to the true distribution, and then we enter the sampling phase. Therefore, during the sampling phase, the GP Model is not updated.

During the sampling phase, our potential energy is given by:

$$E_{\text{pot of samp}} = \mu \tag{4.12}$$

Since the GP Model has stabilized by this stage, we only consider the mean of the predicted values, which will make the sampling results more closely approximate those from the true distribution.

In the initial stages of sampling, the result of sampling is influenced to some extent by the starting points of the samples. To mitigate this influence, we employ a burn-in strategy during the sampling phase. Using the Burn-in strategy, the samples generated during the first certain number of iterations are discarded [GP15]. The samples after the burn-in period are collected and used for analysis. The definition of burn-in strategy is introduced in the background part, the details of applying it in our task will be discussed in the next chapter.

4.5.4 From Latent Points to Decoded Outputs

Until now, we have saved the trained VAE models and sampled points in the latent space. The next step is to use the decoder part of the VAE to transform these points back into mathematical expressions. The reconstruction follows the steps below.

Character VAE

Feeding the point in the latent space and start symbol to the trained decoder of Character VAE, get the logits outputs. For each step, choose the symbol with the maximum logit value until it reaches the maximum length.

GVAE

Feeding the point in the latent space and the start rule to the trained decoder of GVAE, get the logits outputs. For each step, choose the rule with the maximum logit value until it reaches the maximum length. Build the equations according to the chosen rules.

Then we complete the symbolic regression task by obtaining the symbolic outputs. In this thesis, we evaluate the outputs using real datasets generated from the ground truth. In addition, we compare our method with two classic approaches, namely Genetic Programming [MC24] and SInDy [FKK+21]. The evaluation metrics include Root Mean Squared Error (RMSE) [Hod22], the length of the generated equations, and other criteria.

Algorithm 4.3 GP-HMC: Gaussian Process with Hamiltonian Monte Carlo

Input: Training data X, Y, in out task X is the location in latent space while Y is the posterior value;
Covariance function of Gaussian Process and its hyperparameters σ_n^2, ω_0,ω_d;
Potential energy setting E_{pot}, Step size ϵ, Trajectory length L

Initialize:
Set GP model's mean function $m(x)$ and covariance function $k(x, x')$
Initialize GP model's hyperparameters

Exploratory Phase:
Select initial position q_0 for HMC
Compute dynamics using $E_{\text{pot}} = \mu - \sigma$ // set the potential energy, μ and σ are got from GP model
for $t = 1$ to L **do**
    Draw random momentum $p \sim \mathcal{N}(0, I)$
    $\theta_{t-1} \leftarrow \theta$
    $p_{t-1} \leftarrow p$
    $(\theta^*, p^*) \leftarrow \text{Leapfrog}(\theta, p, \epsilon, L)$ // Explaination of Leapfrog can be found in chapter 2
    Compute acceptance probability:

$$\alpha = \min(1, \exp(\text{Hamiltonian}(\theta_{t-1}, p_{t-1}) - \text{Hamiltonian}(\theta^*, p^*)))$$

    // Metropolis acceptance criterion
    Draw $u \sim \text{Uniform}(0, 1)$
    if $u < \alpha$, accept **then**
        $\theta \leftarrow \theta^*$
        Update design points
    else reject
        $\theta \leftarrow \theta_{t-1}$
    end if
    if the posterior is stable, the GP model converges **then**
        Break the trajectory
    end if
end for

Sampling Phase:
GP model is fixed after the exploratory phase
$E_{\text{pot}} = \mu$ // set the potential energy
for $t = 1$ to L **do**
    Simulate particle dynamics, applying Leapfrog and Metropolis acceptance criterion same as in the exploratory phase
    if accepted **then**
        Add the new point to the effective samples
    end if
end for

Output: Effective sample set

5 Experiments

As introduced in Chapter 4, two kinds of datasets are used in this thesis: the expressions dataset and data points datasets. This section presents the details of generating them and the details of the datasets that we used.

5.1 Expressions and data points generation

Datasets

As introduced in the Chapter 4, we have two kinds of datasets, expressions dataset and data points dataset. Four expressions datasets are used in this Chapter. In the first four sections of this chapter, we used **Dataset1** and **Dataset2**. In the fifth section, we used **Dataset3** and the in the last section we used **Dataset4** to do the comparison experiments. The **Dataset1** includes 10,000 equations, among which there are five different equations generated by the aforementioned syntactic rules. The **Dataset2** comprises 10,000 equations, including 20 equations generated by the same rules. Both data sets are generated using the grammars in the Table 5.1 with the maximum production rules of 32. In the first five sections, figures 5.4 to 5.8 were trained and sampled from the first dataset. Due to its simplicity, this allowed for clear and concise visualizations. Figures 5.1 to 5.3 were generated from the second dataset, demonstrating our model's ability to learn from more complex datasets. In the fifth section, we used the **Dataset3** that includes the sorption isotherm equations, which contains relevant prior knowledge in this field. This dataset will be detailed in the fifth section. At the comparison section, we have used the **Dataset4** which includes the commonly used dataset Nyugen [UHO+11] and Koza [Koz94]. All the datasets can be found in the Appendix A.

The data points datasets are generated according to various ground truths and defined domain. The generation rules are introduced in the subsection 5.1.2.

5.1.1 Expressions datasets generation

We defined a context-free grammar for generating mathematical expressions, as shown in Table 5.1. These production rules are used to generate **Dataset1** and **Dataset2**. In this thesis, we assume that all generation probabilities from the same non-terminal symbol are uniformly distributed. These rules include basic mathematical operations (addition, subtraction, multiplication, division), trigonometric functions, exponential functions, constants, and a single variable.

Non-terminal	Production Rules	Probability
S	$S \rightarrow S + T$	0.25
	$S \rightarrow S * T$	0.25
	$S \rightarrow S/T$	0.25
	$S \rightarrow T$	0.25
T	$T \rightarrow (S)$	0.125
	$T \rightarrow \sin(S)$	0.125
	$T \rightarrow \cos(S)$	0.125
	$T \rightarrow \exp(S)$	0.125
	$T \rightarrow x$	0.125
	$T \rightarrow 1$	0.125
	$T \rightarrow 2$	0.125
	$T \rightarrow 3$	0.125

Table 5.1: Grammar rules and their probabilities for generating mathematical expressions in the Dataset1 and Dataset2.

The sampling process is the recursive generation of expressions using grammar rules. The specific steps are as follows:

Initialization: Sampling begins with the start symbol S, and grammar rules are applied sequentially.

Random Rule Selection: At each step, a suitable grammar rule is selected randomly based on its probability and is used to replace the current non-terminal symbol.

Stopping Criteria: The generation process terminates when all non-terminal symbols have been replaced with terminal symbols. To avoid generating excessively long or overly complex expressions, a maximum production step count is set. If this limit is exceeded and the expression is incomplete, the process restarts.

Retry Mechanism: If a valid expression is not successfully generated during a sampling attempt, a new random seed is used to retry the process.

5.1.2 Data Points datasets generation

In the experiments conducted in this thesis, to ensure the applicability of the method, we utilized different equations as ground truth. Here, we take one of these equations as an example to demonstrate the dataset generation process.

Generate uniformly distributed x values: n evenly spaced points are generated within the specified range.

Compute the mean values of the target function: For each x value, the mean is calculated using the given ground truth, which provides the expected output for the corresponding input x.

Add independent Gaussian noise: Independent noise is added to each mean value to simulate real-world data variability. The noisy output is computed as:

$$y = \mu + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma), \tag{5.1}$$

where μ is the mean value calculated from the target function, and ϵ represents Gaussian noise with a mean of 0 and a standard deviation specified by the parameter σ.

5.2 Processioning of expressions

The processing of expression is different with Character VAE and GVAE, since Character VAE uses single Characters as input while the GVAE used context-free grammars as input.

5.2.1 Data processing of Character VAE

Get Equations

Read equations from the generated dataset. Each line in the file represents a single equation.

Build Vocabulary

Build a vocabulary by extracting all symbols (e.g., variables, constants, operators, etc.) from the list of equations. Construct a vocabulary that maps each symbol to a unique index. Assign indices to the symbols in order and include the following special symbols:

- `<start>`: Represents the start of a sequence.
- `<end>`: Represents the end of a sequence.
- `<pad>`: Used to pad sequences to ensure uniform length.

Generate input sequences

Each equation is processed as follows: First, the `'<start>'` symbol is added at the beginning of the equation. Regular expressions are then used to extract symbols and map them to their corresponding indices, generating the input sequence. The `'<end>'` symbol is appended to the end of the equation. The length of each sequence is calculated, and the longest sequence length is recorded. Finally, all sequences are padded to ensure they have the same length by using the `'<pad>'` symbol to pad them to the length of the longest sequence. During the processing of equations, generating input sequences of the same length is crucial, as the input for training a VAE must have consistent dimensions.

5.2.2 Data processing of GVAE

While training the GVAE, the preprocessing of equations follows the same guidelines as Character VAE. Different is that the dictionary is build by the grammar rules.

Get Equations

Same as training the Character VAE that each line is an equation.

Build the Rules Vocabulary

Parse trees will be generated from equations according to the rules of context-free grammars. Each rule will be mapped to a unique index and used to build the vocabulary of rules. Assign the indice for the pad symbol:

- <pad>: Used to pad sequences to ensure uniform length.

Generate input sequences

Each equation will be parsed into a parse tree, and the extracted rules will be mapped to their corresponding indices in the dictionary.

Build the input sequence of each equation base on the corresponding indices. Pad them to the same length for the same reason as Character VAE. Here we still use the length of the longest sequence.

5.3 Comparison of variational autoencoders

5.3.1 Character VAE

Our model architecture is already explained in the method Chapter. However, the choosing of hyperparameters could also influence the quality of the result. Since this thesis requires the construction of a structured latent space, we first focus on the dimensionality of the latent space.

Influence of dimension of Latent Space

The latent space is a representation space, used by the model to capture the core features of the input data. It is typically generated by the model's encoder and is designed to encapsulate the essential characteristics of the data. Latent spaces are often low-dimensional, aiming to compress the redundant information in the input data. However, when the dimensionality is too low, the data may lose too much information, leading to insufficient representation capacity, which degrades the quality of the model's generated results. Conversely, when the dimensionality is too high, the data representation may become overly sparse, increasing the risk of overfitting. Beside that, the computational complexity also increases, making the model less efficient.

Influence of dimension of number of layers

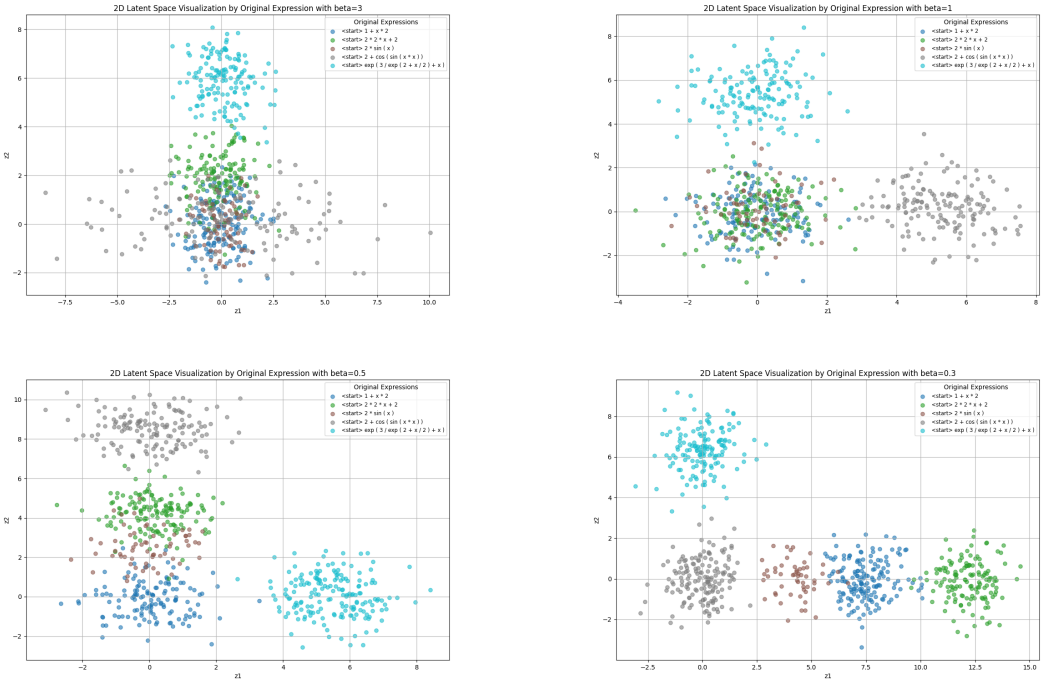


Figure 5.1: Influence of the KL Divergence Loss coefficient in the latent space. The coefficients from the top to the bottom are 3, 1, 0.5 and 0.3. The higher the coefficient, the distribution in the latent space will be more like normal distribution; the lower the coefficient, the same expressions are more clustered together.

Secondly, the complexity of the model determine its learning capacity. When the model is too simple, it may not have sufficient learning ability, potentially leading to underfitting as it might not learn all the features. Conversely, when the model is overly complex, training time increases, and the model might memorize information rather than generalize, causing the loss on the validation set to be higher than on the training set. Therefore, we also focus on the number of LSTM layers in the model, experimenting with different layer counts and observing the loss functions in both the training and validation sets to select the most suitable model parameters.

Trade-off in Loss Function

In the theoretical foundation introduced in the previous chapter, we defined the loss function in VAE, where the coefficient β controls the relative weight between the KL Divergence Loss and the reconstruction error. Variations in this relative weight result in changes in the sample distribution within the latent space.

Here we use a dataset with five identical equations as an example, setting the coefficient of KL Divergence Loss respectively as 3, 1, 0.5, and 0.3. The visualisation of the latent space according to the settings are shown in the Figures 5.1.

From the Figure 5.1 we can see: as β increases, the weight of the KL Divergence Loss grows, causing the sample distribution to align more closely with a normal distribution. Conversely, as β

decreases, the sample distribution becomes more dispersed, but samples of the same class become more tightly clustered, and the reconstruction error is reduced.

Based on the above characteristics, during the actual training process, we adopted an annealing strategy to control the value of β. Specifically, β is not a fixed value but grows dynamically as training progresses. It can allow the VAE first focus on the reconstruction function, and let the output be similar to the input. During the training process, the weight of KL Divergence Loss increases and the distribution in the latent space is more like a normal distribution. The formula for **linear annealing** is given as:

$$\beta_t = \min\left(1, \frac{t}{T}\right), \tag{5.2}$$

where $\beta_t$is the value of β at the t-th training iteration. $t$means the current training iteration and T is the total number of iterations after which annealing is complete [Rut89].

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{recon}} + \beta_t \cdot \beta \cdot \mathcal{L}_{\text{KL}} \tag{5.3}$$

This approach encourages the model to focus more on the reconstruction error during the initial stages of training, aiming to enable the model to generate samples similar to the input. As training progresses and the reconstruction error gradually decreases, more attention is shifted to the KL Divergence Loss, allowing the model to generate a latent space that aligns with the desired distribution.

Hyperparameter Tuning of Model Architecture

Based on the above analysis, in this paper, we set the dimensions of the latent space to 2, 4, 8, 16, and the number of LSTM neural network layers to 2, 3, 5, 8 and apply the linear annealing with $\beta = 0.3$ for experimentation. The dataset contains 10,000 equations generated by the context-independent grammar rule 5.1, of which eighty percent is used as the training set and twenty percent as the test set. The model is trained for 120 epochs at each hyperparameter combination and the reconstruction loss, KL Divergence Loss and total loss on the training and test sets are recorded. The model was trained using the Adam optimizer to minimize classification error. For each combination of hyperparameters, we have recorded all three loss value for both training set and validation set during the training process and plotted the corresponding curve.

Figure 5.2 shows the variation of the KL Divergence Loss during the training. It can be observed that the value of the KL Divergence Loss initially increases and then decreases. This is because, during the initial stages of model training, the model focuses more on minimizing the reconstruction error, while the coefficient corresponding to the KL Divergence Loss relatively small. 5.3 shows the variation of the reconstruction loss during the training. It can be observed that the reconstruction loss of the training set and validation set have a similar trend and do not appeared big difference. That means the model has learn the features from the input, not only remembered them. 5.4 shows the total loss with the definition 2.18. The total loss of both set have a slight uptick because of the coefficient change by using linear annealing. Since both KL Divergence Loss and Reconstruction loss stabilize after about 80 epochs, we can assume that the model has been trained at this time. The Table 5.2 shows the total value at 120 epoch:

From the Table 5.2, it can be seen that when the latent space dimension is 16 and the number of LSTM

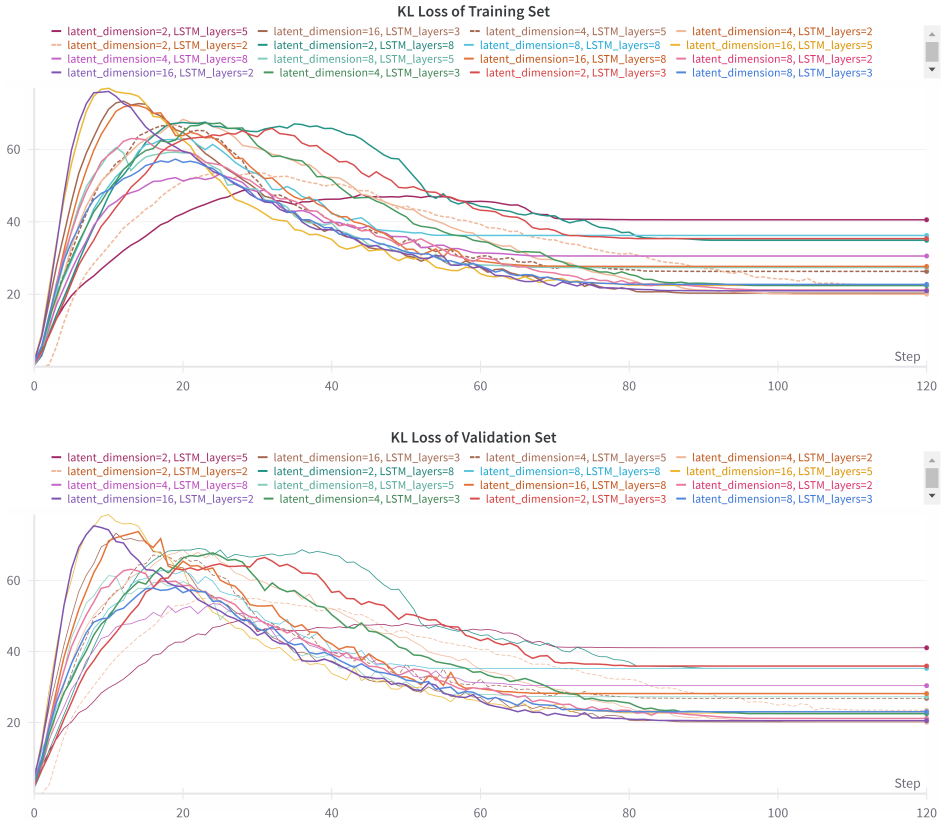


Figure 5.2: KL Convergence loss value of training set and validation set during the training. Each step means an epoch. The latent space dimension and number of LSTM layers are shown in the labels.

layers is 3, the value of the loss function reaches its lowest in both the training and validation sets. To verify its clustering effect in high-dimensional latent spaces, we use the t-distributed Stochastic Neighbor Embedding (t-SNE) method to perform dimensionality reduction and visualization. In the figure below, we use the t-SNE method to reduce the high-dimensional latent space to a two-dimensional space and visualize it. The goal of t-SNE is to embed high-dimensional data into a lower-dimensional space while preserving the local neighborhood relationships of data points in the high-dimensional space as much as possible [VH08].

From the Figure 5.5 It is shown that each type of expression in the dataset has clustered in the latent space. Notably, the shorter equations are concentrated in the bottom left corner, while the longer expressions are predominantly in the right part. Additionally, expressions containing exp and sin are mostly gathered in the upper part, whereas those including cos are primarily found at the bottom. This indicates that similar expressions are closer together in the latent space, suggesting that the distance in the latent space can represent the similarity at the expression level.

Visualisation of the latent space to the expressions

As discussed in the method chapter, we need to transfer the points in the latent space to the expression

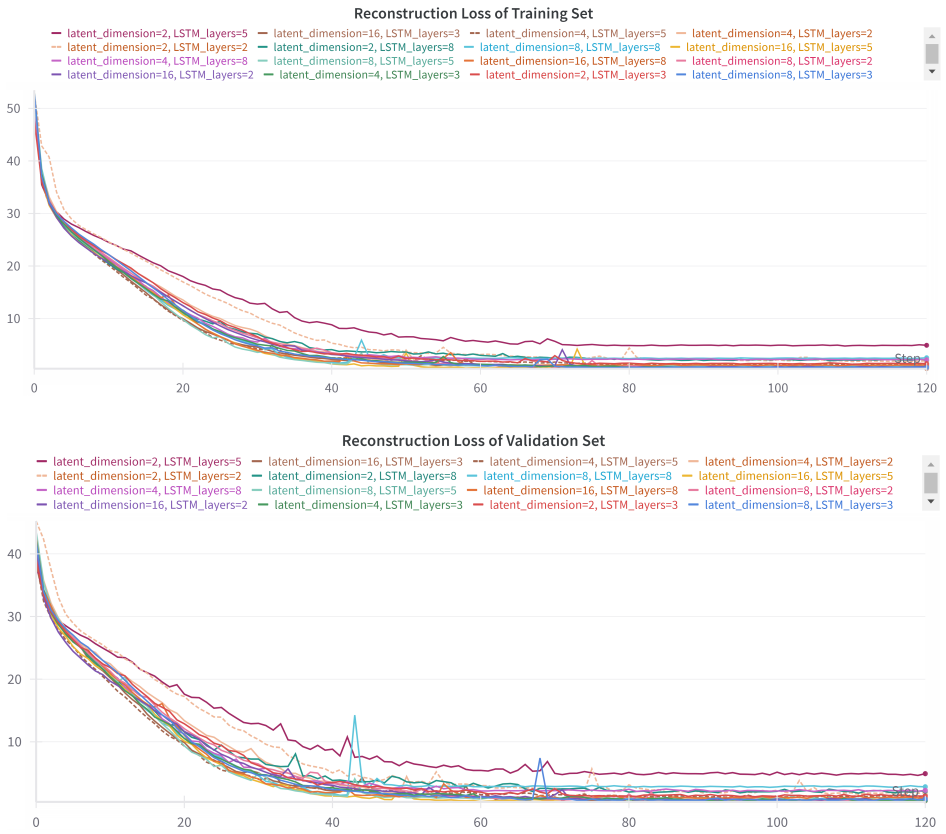


Figure 5.3: Reconstruction loss value of training set and validation set during the training. Each step means an epoch. The latent space dimension and number of LSTM layers are shown in the labels.

level. To visualize the distribution of expressions in the latent space, we show it by doing uniformly sampling in the latent space. We have sampled in the latent space with the region of $z1$ is from -5 to 5, the region of $z2$ is from -5 to 8. We do the visualization with the same setting as used in this section, namely use the five equations dataset and $'2 \cdot 2 \cdot x + 2'$ as ground truth, latent space dimension is 2. For the higher dimensions, the visualization can be down with the same method.

From the figure 5.6 we have observed that the expressions are clustered. The figure shows that the identical expressions are generated from the same region, which means the location of latent space could provide the information, which could be decoded to the information of mathematic expressions. The second point is, new expressions are generated. If we look at the orange squares in the top right part, we can see that a new express which does not exist in the dataset, $' < start > 2 + x * 2'$ is generated between the expression $' < start > 1 + x * 2'$ and $' < start > 2 * x * 2'$. It represents that our model is able the generate new expressions which is from the structure level similar to its neighbors. However, not all output expressions are valid. Since the output of Character VAE are based on the characters, that means the output could be some invalid expressions, for example $' < start > 2 + x*'$.

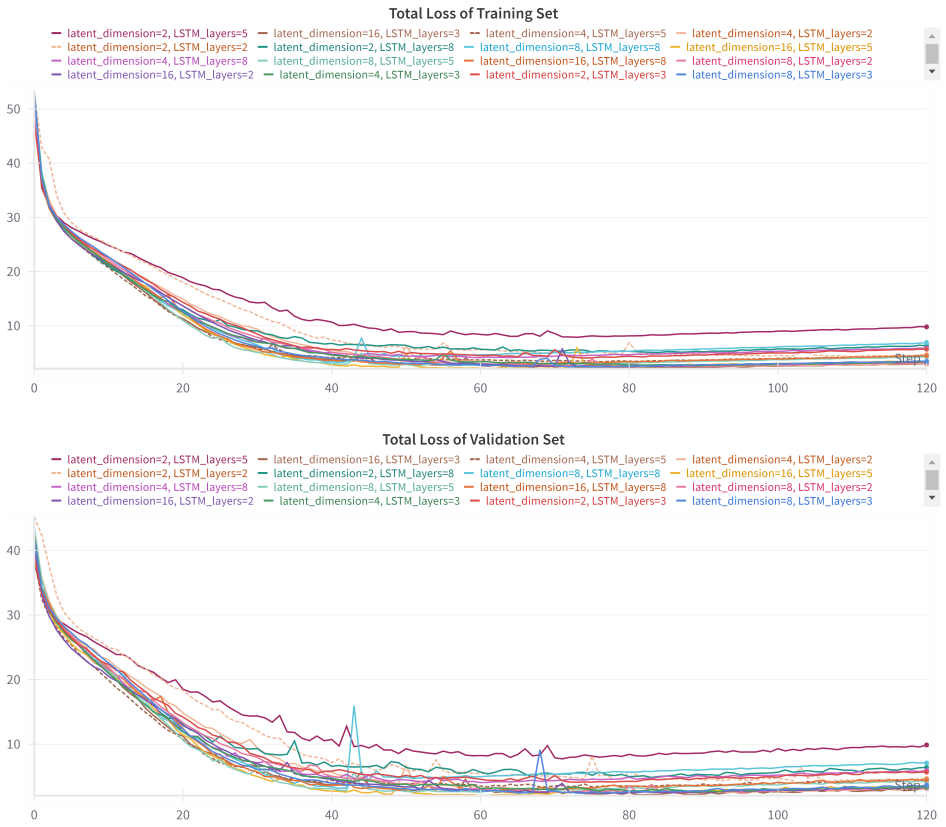


Figure 5.4: Total loss value of training set and validation set during the training. Each step means an epoch. The latent space dimension and number of LSTM layers are shown in the labels.

5.3.2 Grammar VAE

Model Architecture and Dimension of Latent Space

The model architecture of GVAE is already introduced in Chapter 4. Our also chose the layer of RNN layers, and the dimension of latent space of our hyperparameter for the GVAE model. However, we used the Maximum Likelihood Loss as the reconstruction loss function to train the GVAE [ZC17]. The KL Divergence Loss keeps same as Character VAE.It is designed to maximize the likelihood of the correct outputs given the input data [ZC17]. During the training of GVAE, we used the same trade-off strategy as Character VAE in the loss function. We have chosen the latent space dimension = 2, 4, 8, 16 and GRU layers = 3, 5, 8. After training for the corresponding combinations, the result shows that the model with latent space dimension = 16 and GRU layers = 3 had the best performance. The training process is shown in the Figure 4.7

However, through the training and validation loss keeps go down during the training, and the clustering result shows that the expressions are not well clustered and the ability of generating new examples is very limited.

latent space dimension	LSTM Layers	loss of training set	loss of validation set
2	2	4.69668	4.71507
2	3	5.68639	5.68391
2	5	9.80157	9.88126
2	8	6.31556	6.40449
4	2	2.99058	3.14087
4	3	3.40197	3.35304
4	5	4.54866	4.68133
4	8	5.84823	5.9303
8	2	3.21482	3.14563
8	3	3.42204	3.64039
8	5	4.22467	4.13903
8	8	6.91087	7.08414
16	2	3.34329	3.37662
16	3	2.94189	3.12592
16	5	3.44158	3.71329
16	8	4.50674	4.49892

Table 5.2: Comparision of total loss

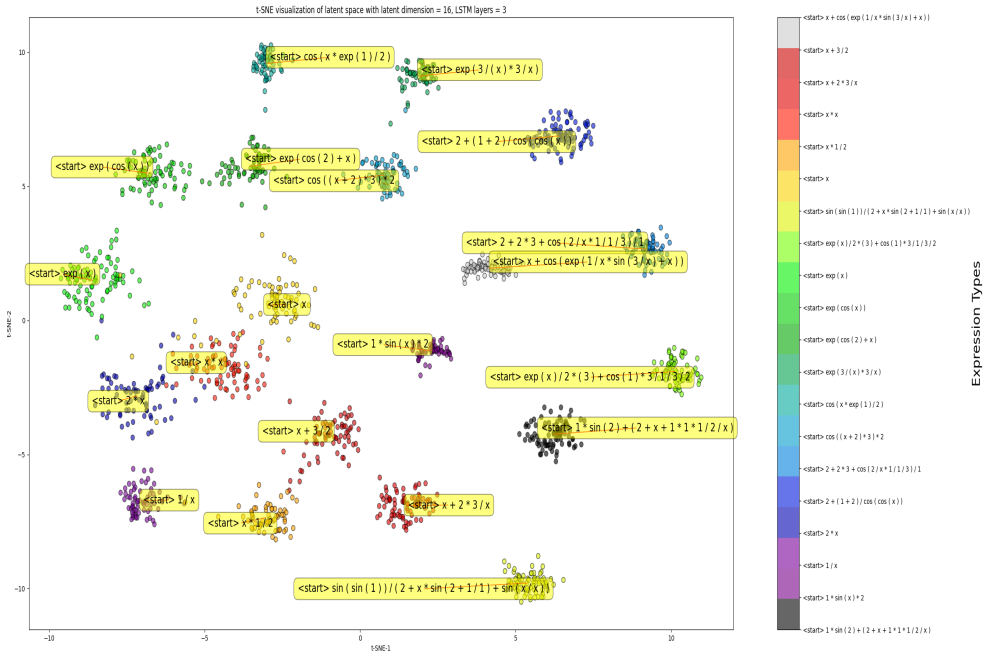


Figure 5.5: Latent space visualization with latent space dimension = 16, LSTM Layers = 3. *Dataset2* is used. Each color represent an identical equation, which are shown at the right part.

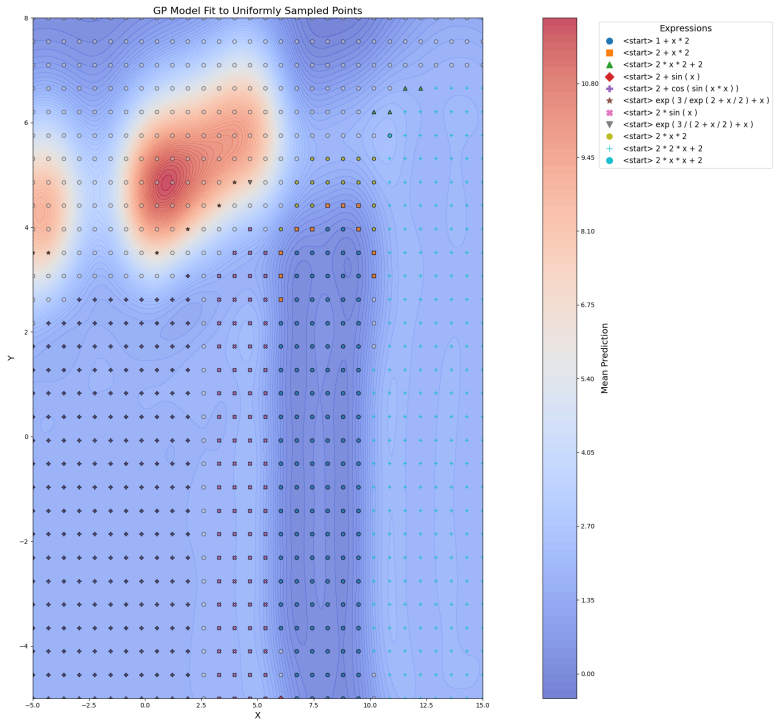


Figure 5.6: In this figure, each kind of symbol represents one kind of expression. X=z1, Y=z2 in the latent space. Since our output is based on the expressions level, we didn't use the automatically simplification here. The grey circle which is not shown in the expressions means the output is not a valid expression. The color represents the mean prediction of likelihood.

Due to the better performance of Character VAE, we have used the latent space learned by Character VAE for the Bayesian Reasoning and sampling steps.

5.4 Comparison of hyperparameters in Gaussian Progress Hamiltonian Monte Carlo

After the structured latent space is learned from the VAE, we can used the pre-trained model and latent space to do the sampling. To make the visualization part clear and easy to understand, we used the **Dataset1** and latent space dimension = 2 in this section. We have already shown that our model is able to learn the more complex datasets can have a good performance with higher latent space dimension, this sampling method can be expanded to the more complex problems too.

In this section, the sampling is based on the latent space learned by Character VAE with using coefficient $\beta = 0.3$, with linear annealing. We set the expression *'<Start> 2 * 2 * x + 2'* as **ground truth** in this section and assume the $P_{exp}(z)$ is uniformly distributed in the latent space. The likelihood is calculated according to 4.10. The process of GP-HMC follows the 36. In this section, we will discuss the influence of hyperparameters in the GP-HMC method and improve the sampling performance by adjusting the choose of mean function, the those of lengthscale and apply burn-in

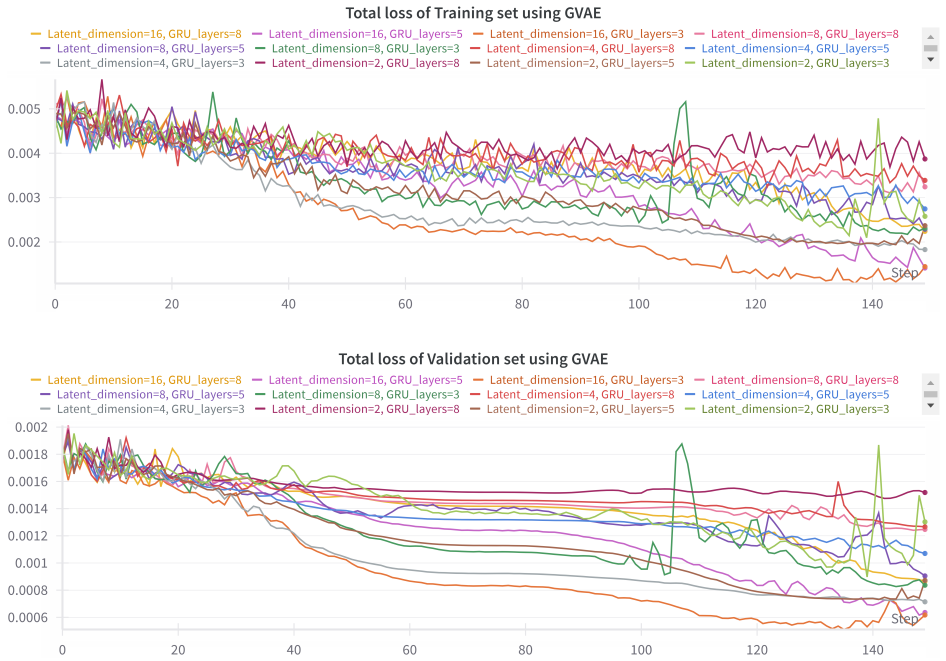


Figure 5.7: The training result of GVAE during 150 epochs. The Loss is the total loss includes KL Divergence Loss and reconstruction loss using Maximum Likelihood Loss. Loss function of training dataset and validation dataset of each epoch is shown.

strategy [RW06] [GP15].

Choose of Mean

In the previous chapter, we introduced the process of fitting a GP Model, which requires first defining the mean function, also known as the default value. Setting the default value to zero is a common choice in practice [RW06]. However, in the context of the problem studied in this thesis, we need to perform sampling based on this model. When the values of the sampling points are greater than zero, especially significantly greater than zero, the default mean function value forms a local minimum among these points. During subsequent sampling, this local minimum may block the sampling process, causing it to remain stuck in this minimal region.

We aim to achieve smooth transitions between sampling points rather than forming local minima. To address the issue, we propose using the mean of the posterior probabilities of all current sampling points as the value of the mean function each time the GP Model is refitted. Using this approach can help avoid the problem to some extent.

The figure 5.8 shows a comparison of GP Models in the latent space with a dimensionality of 2, trained on five equations, using two different mean functions.

From the figure, we can see that in the left image, when we use 0 as the default value for the mean, the predicted mean hovers around 0 in areas lacking information. This creates a regional minimum, causing sampling points to be trapped within this area, unable to move towards the

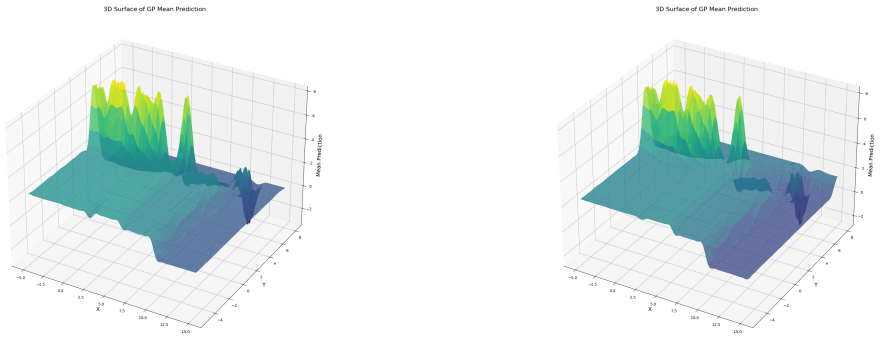


Figure 5.8: Comparision of different mean value setting. The left figure shows the mean prediction of GP Model when choosing the default mean value as 0. The right figure shows the same prediction while setting the default mean as the actual average mean of all samples. For the both figures, X=z1, Y=z2 in the latent space.

actual ground truth. In the right image, by using the current average mean value as the default, areas without information are able to borrow from the average of all current sampling points. Even though the predicted mean values in areas lacking information might be slightly lower than those in surrounding areas, the smaller difference allows sampling points to more easily escape from these information-deprived areas.

The Figures in 5.9 show us the trajectory of the sampling phase. The trajectory starts from white and becomes black with the iteration growing. It is clear to see that when setting mean = 0, it is hard to jump out from the local minimum since it's a big jump from the local minimum to its neighbours. When the mean is set as the actual average mean, the jump becomes easilier and the sampling locations moved to the region of ground truth fester.

To show the impact more directly, we numerically compared the impact of the two different mean function formulations on the final sampling results. For this comparison, we choose the equation $' < start > 2 \cdot 2 \cdot x + 2'$ as the ground truth, initial sampling point as: (5, 8), length scale = 0.5, step size = 0.015, steps in each sampling = 10, sampling iterations = 1000. We repeated the experiment for 10 times and counted the mean and standard deviation of the following results. The definition of the following criteria are:

First Ground Truth Iteration: In which iteration it sampled the ground truth for the first time.

Valid Expressions Ratio: The ratio of valid mathematical expressions in the samples to the total number of samplings.

Ground Truth Ratio: The ratio of ground truth expressions to the total valid expressions in the samples.

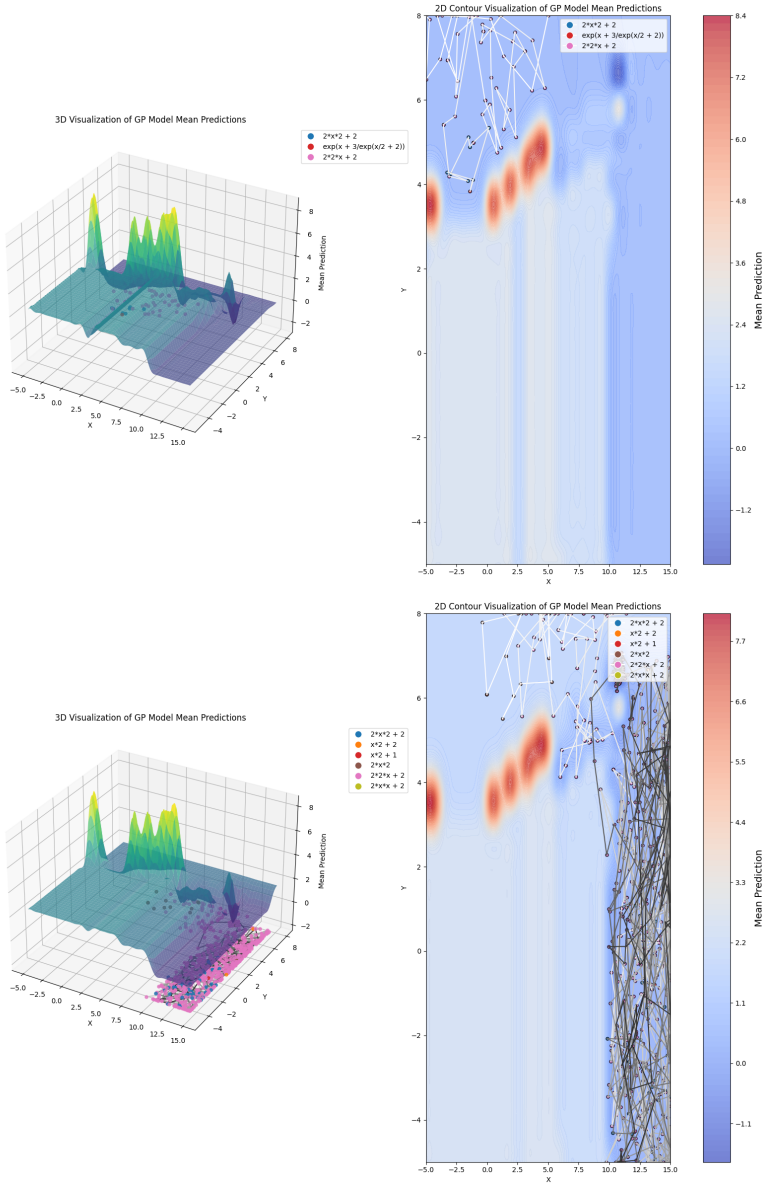


Figure 5.9: The figure above shows the sampling trajectory in the latent space when setting the default mean as 0; the figure below shows the sampling trajectory in the latent space when setting the default mean as the actual average mean. For both figures, X=z1, Y=z2. The color of trajectory shows the order of sampling, it starts from white and becomes to black during the sampling phase.

Mean Function	**First Ground Truth Iteration**	**Ground Truth Ratio (in valid expressions)**	**Valid Expressions Ratio**
Mean = 0	130.30± 111.45	**98.75%±9.98%**	27.14%± 19.16%
Mean = actual average value	**70.5± 72.56**	94.20%±3.10%	**78.36%±8.49%**

Table 5.3: Comparison of different mean functions and their impact on sampling results. **Dataset1** is used for the experiment and the sampling experiments have been repeated for 10 times. In the Table, the average values and standard deviations of each criterion for the 10 experiments are shown.

The results are presented in Table 5.3. From these results, we can see that by applying the default mean value as the actual average value when sampling, both the stability of the initial iterations and the ratio of valid expressions have significantly improved. Although the ground truth ratio is higher when the mean is set to 0, the higher ratio of valid expressions achieved by setting the mean as the actual average value allows for more effective sampling of the ground truth.

Choose of Lengthscale

By adjusting the mean function, we can partially address the issues of the GP Model being insufficiently smooth and forming local minima. However, as the dimensionality of the latent space increases, the sampling points become sparse. This results in many points in the space being unable to obtain estimates based on the information from surrounding points.

As mentioned in the previous chapter, the length scale controls the smoothness of GP model. That is to say, it allows controlling the spatial range of influence of the sampling points. For example, in the case of the RBF kernel, its mathematical expression is:

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\ell^2}\right), \tag{5.4}$$

where $k(x, x')$ computes the similarity between two points x and x', $\|x - x'\|^2$ is the squared Euclidean distance between x and x' and ℓ is the length scale hyperparameter, which controls how quickly the similarity decreases as the distance between x and x' increases. When the length scale increases, the value of the RBF kernel also increases, meaning the variance becomes larger and the range of influence of the sampling points expands. Conversely, the opposite is true [RW06].

In this thesis, we experimented with different values of the length scale and compared their effects. The Figure 5.10 below illustrates the impact of varying the length scale in a two-dimensional latent space .

In addition, we analyzed the impact of different length scales on the final sampling results, the definition of the criterion are same as used when analyzing the impact of setting of mean value. The final sampling results corresponding to different length scales are shown in the Table 5.4.

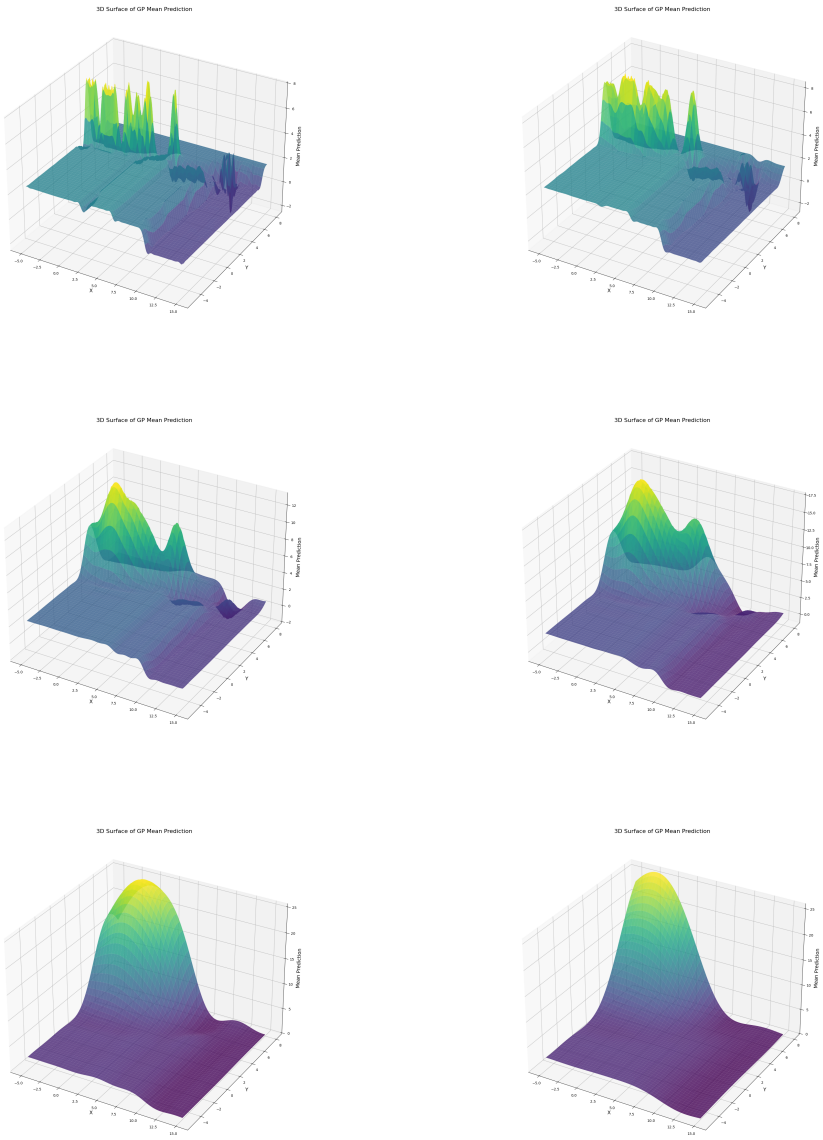


Figure 5.10: The six figures show the prediction of the mean value of the GP Model in the latent space. The lengthscales are separately 0.3, 0.5, 1.0, 1.5, 3.0, 5.0. For the all figures, X=z1, Y=z2.

Lengthscale	First Ground Truth Iteration	Ground Truth Ratio (in valid expressions)	Valid Expressions Ratio
0.3	131.1±97.44	86.50%±9.26%	35.68%±13.23%
0.5	166.2±170.94	83.27%±14.21%	37.22%±15.86%
1.0	121.5±152.12	85.15%±13.78%	34.49%±16.21%
1.5	150.7±231.36	**72.94%±23.80%**	36.12%±18.63%
3.0	**84.2±95.48**	84.63%±12.52%	69.81%±12.40%
5.0	135.3±141.21	77.66%±17.32%	**77.55%±5.30%**

Table 5.4: Comparison of sampling results under different length scales. **Dataset1** is used for the experiment and the sampling experiments have been repeated for 10 times. In the Table, the average values and standard deviations of each criterion for the 10 experiments are shown.

From the result above, it can be seen that as the lengthscale increases, although the values of the three indicators fluctuate, the overall trend is that the *First Ground Truth Iteration* gradually decreases, while the *Ground Truth Ratio* and *Valid Expressions Ratio* gradually increase. The bolded part represents the optimal indicators according to the revised standard.

Influence of burn-in strategie

The burn-in strategy is a common technique in MCMC methods, used to ensure that the final samples are representative of the target distribution. During the initial phase of sampling, the algorithm may start from an arbitrary point in the parameter space, leading to biased samples. To address this, the first n iterations, known as the *burn-in period*, are discarded. This allows the Markov chain to converge to the target distribution before collecting samples for analysis [GP15].

During the experimental process of this thesis, we have observed that, in the initial stages of sampling, the positions of the sampling points are largely influenced by the starting point of the sampling. This is because in HMC sampling each subsequent sample depends on the current state. When using different latent spaces, the same points in the space represent different information. Therefore, it is essential to eliminate the influence of the starting point.

In our approach, we implement a burn-in strategy to discard early-stage samples that may not adequately reflect the posterior distribution. This ensures that only samples from the stationary phase are included, improving the quality and reliability of the results.

5.5 Transfer prior knowledge to latent space

As introduced in the method chapter, we need to combine the prior knowledge with the likelihood in the latent space according to the equation 4.5. This is one of our contributions: transferring the prior knowledge to the latent space and applying Bayesian reasoning in the latent space.

In this thesis, we applied our method to soil and material science, especially the sorption isotherms equations. It describes the model between the equilibrium sorptive concentration c $\frac{\text{mg}}{L}$ in the solution phase and the sorbate concentration s $\frac{\text{mg}}{Kg}$ [TG12]. We are focused on the sorption isotherm equations which have similar constructions, they are shown in the Table 5.5.

We generated the **Dataset3** with 10000 equations, which includes 30 different expressions. The dataset generated using these production rules are used as the prior knowledge at the expressions level. To applying Bayesian Reasoning in the latent space, we need to transfer the dataset to the latent space. Remember that we already trained and saved our Character VAE with the tuned hyperparamters, we can use the prior knowledge dataset as input and spread forward to get latent space. Turn the model to the evaluate model that the parameters will not be updated. To get the prior knowledge, we have used the probability context-free grammar the generate the equation dataset. The grammars used are shown in the Table 5.6. The probability of each rule corresponding to every non-terminal symbol is equal and shown in the Table 5.6.

Model	**Equation**
Langmuir	$S_T * \frac{kc}{1+kc}$
Modified Langmuir	$S_T * \frac{k_1 c}{1+k_1 c} * \frac{1}{1+k_2 c}$
Two Site Langmuir	$S_T * \frac{f_1 k_1 c}{1+k_1 c} + \frac{f_2 k_2 c}{1+k_2 c}$

Table 5.5: The Langmuir equations which we used for our experiment

Non-terminal	**Production Rules**	**Probability**
S	$S \rightarrow A * B$	1
A	$A \rightarrow$ 'S_T'	1
B	$B \rightarrow C * D$ $B \rightarrow E + F$	$\frac{1}{2}$ $\frac{1}{2}$
C	$C \rightarrow$ '1' $C \rightarrow$ '1'/('1' + $k2 * x$)	$\frac{1}{2}$ $\frac{1}{2}$
D	$D \rightarrow k1 * x/(\text{'1'} + k1 * x)$	1
E	$E \rightarrow f1 * k1 * x/(\text{'1'} + k1 * x)$	1
F	$F \rightarrow f2 * k2 * x/(\text{'1'} + k2 * x)$	1

Table 5.6: The context-free grammar used to generate the equations. The left column shows the nonterminal symbols; right column shows the grammar, namely production rules. x is the independent variable and S_t, $f1$, $f2$, $k1$, $k2$ are present constant here.

Our pre-trained encoder will process each equation in the dataset to the latent space. Due to the parameter trick in the VAE, for every equation, after the encoding part, we will get the mean of the location of the input in the latent space μ and the standard deviation of it σ. Each pair of μ and σ represents a Gaussian distribution of the given expression, all pairs of them represent the Gaussian Mixture Model in the latent space. Then we have got the expectation of latent space, which is $P_{\text{EXP}}(z)$ in the equation 4.5.

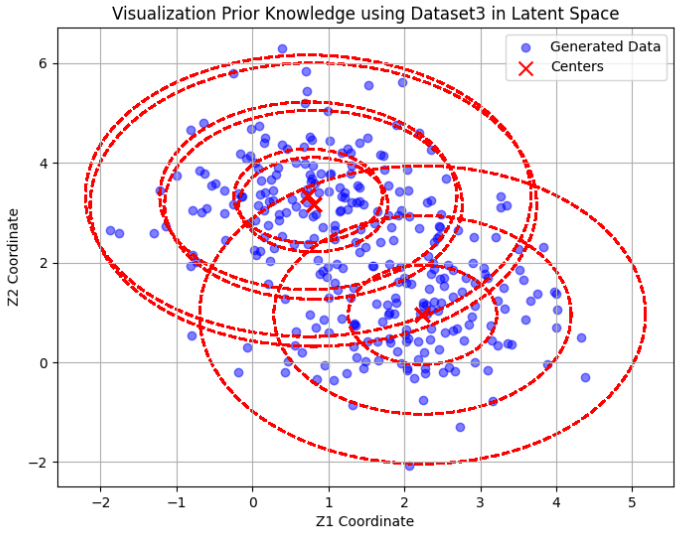


Figure 5.11: Prior Knowledge in the latent space using **Dataset3**. The blue points show the mean and standard deviation of each pair of μ and σ. The red dashed lines are the contour lines of the GMM Model. Z1 and Z2 are coordinates in the latent space.

From the Figure 5.11 and the Figure 5.12, it can be observed that our model is able to transfer the prior knowledge from the expressions level to the latent space level. The $P_{exp}(z)$ can be used to calculate the posterior in the latent space by combining the likelihood talked in the section 5.4. Due to the time limitation of this thesis, we keep the assumption that the prior knowledge follows the uniform distribution in the latent space. This section shows that our method is also able to be used with the prior knowledge in the latent space.

5.6 Evaluation - comparision with other methods

In the end, we will evaluate our method and compare our method with two classic symbolic regression methods, Genetic Programming [MC24] and SInDy [FKK+21].

Dataset. In this section, we have used two datasets to evaluation our method. The **Dataset1** is the dataset used for tuning the hyperparameters in this Chapter. The complete dataset is in Appendix A. Another **Dataset4** is based on Koza [Koz94] and Nyugen [UHO+11], which is commonly used in the symbolic regression field.. For the complete dataset, see Appendix A. The data points generation follows the steps in the section 5.1 with the equation 5.1. All data points are randomly split into 2 sets, the training set with 80% data points and the test set with 20% data points. The training dataset is used to train the Character VAE and calculate the likelihood in our method, as well as to find the expression for the GP and SInDy methods. The test set is for the evaluation based on the result of the training phase.

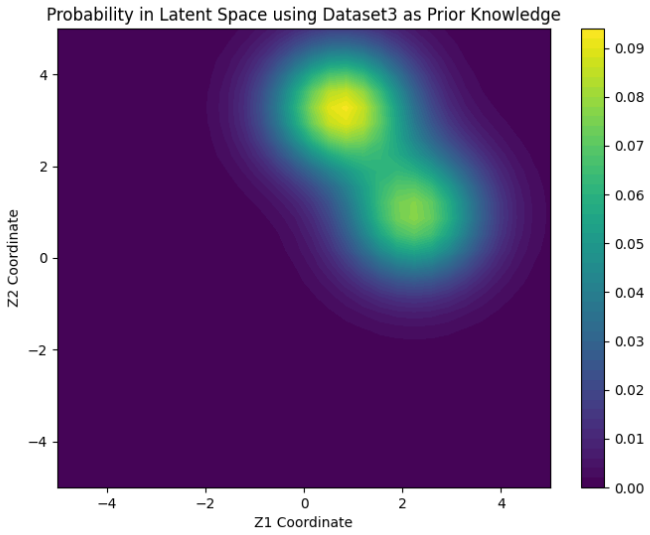


Figure 5.12: $P_{exp}(z)$, the expectation of each location in the latent space using **Dataset3**. The color bar shows the value of probability according to the color bar at the right, the model is the same GMM introduced in this section.

Evaluation. The Evaluation is based on the test set and uses the RMSE as a criterion. We got the RMSE for both the training and test datasets. RMSE is a metric that measures the average magnitude of prediction errors by taking the square root of the mean of the squared differences between predicted and actual values [Hod22]. For the Genetic Programming method, we repeated the experiment 10 times and reported the median of RMSE. The SInDy method has the same result during the experiment. For our method, we also reported the median of RMSE which is more robust to outliers.

Our Method. Based on the result and fine-tuning in this chapter, we choose the following model and parameters for our method: use Character VAE with latent space dimension = 2, LSTM layers = 3 for the pretrained VAE model. Set the default mean value of the GP model as the average mean of current samples, length scale = 3, sampling iterations = 1000, burn-in steps with 300. Steps for each sampling = 100 and step size = 0.015.

Genetic Programming. Genetic Programming is a classic and widely used algorithm in the symbolic regression field; the basic idea and the steps of it are already introduced in Chapter 2 [MC24]. It is implemented by *gplearn*.

SInDy. SInDy is a powerful method for automatically discovering mathematical models of dynamic systems from data. By combining sparse regression techniques with an appropriate library of functions, SInDy accurately identifies the key functions and their coefficients that describe the dynamics of a system [FKK+21]. The implementation is based on the *pysindy*.

Result. The results of the experiment for each algorithm and the data set are shown in the Table 5.9 and 5.10. Through the baseline method, GP have good performance in the training set, it tends to overfit the observed data since the RMSE results in the test set are more higher than them in

Median of RMSE for Dataset1					
Ground Truth Expression	**Genetic Programming**	**SInDy**	**our method**	**Expression**	**Data Range**
1	**0.35**	3.97	0.37	$1 + x * 2$	U[0,5]
2	**0.24**	8.13	0.26	$2 * 2 * x + 2$	U[0,5]
3	**0.27**	1.82	0.29	$2 * \sin(x)$	U[0,5]
4	0.33	0.33	**0.26**	$2 + \cos(\sin(x * x))$	U[0,5]
5	2.41	57.09	**0.37**	$\exp(3/\exp(2 + x/2) + x)$	U[0,5]

Table 5.7: Median of RMSE of the **training dataset** of expressions **Dataset1**. For the GP method, we repeated the experiment 10 times. The SInDy method always gives the same result back. For our method, we chose all samples. Best performances are bolded.

Median of RMSE for Dataset1					
Ground Truth Expression	**Genetic Programming**	**SInDy**	**our method**	**Expression**	**Data Range**
1	**0.27**	4.16	0.28	$1 + x * 2$	U[0,5]
2	**0.15**	8.13	0.19	$2 * 2 * x + 2$	U[0,5]
3	0.30	2.37	**0.27**	$2 * \sin(x)$	U[0,5]
4	0.40	0.40	**0.32**	$2 + \cos(\sin(x * x))$	U[0,5]
5	3.13	56.98	**0.44**	$\exp(3/\exp(2 + x/2) + x)$	U[0,5]

Table 5.8: Median of RMSE of the **test dataset** of expressions **Dataset1**. For the GP method, we repeated the experiment 10 times. The SInDy method always gives the same result back. For our method, we chose all samples. Best performances are bolded.

Median of RMSE for Dataset4					
Ground Truth Expression	**Genetic Programming**	**SInDy**	**our method**	**Expression**	**Data Range**
Koza-1	**35.23**	278.63	43.29	$x^4 + x^3 + x^2 + x$	U[-5,5]
Koza-2	**209.50**	1578.00	377.64	$x^5 - 2x^3 + x$	U[-5,5]
Koza-3	**486.29**	5771.84	5057.37	$x^6 - 2x^4 + x^2$	U[-5,5]
Nguyen-1	**9.11**	81.35	10.86	$x^3 + x^2 + x$	U[-5,5]
Nguyen-2	**8.02**	293.61	9.56	$x^4 + x^3 + x^2 + x$	U[-5,5]
Nguyen-3	107.78	1477.42	**97.07**	$x^5 + x^4 + x^3 + x^2 + x$	U[-5,5]
Nguyen-4	**159.07**	6936.58	None	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	U[-5,5]
Nguyen-5	**0.34**	0.38	54.76	$\sin(x^2)\cos(x) - 1$	U[-5,5]
Nguyen-6	**0.67**	1.05	47.99	$\sin(x) + \sin(x + x^2)$	U[-5,5]
Nguyen-7	**573.75**	1030.18	918.68	$\log(x + 1) + \log(x^2 + 1)$	U[0,5]
Nguyen-8	**1111.23**	1487.57	1506.42	$\sqrt{x}$	U[0,5]

Table 5.9: Median of RMSE of the **training dataset** of expressions **Dataset4**. For the GP method, we repeated the experiment 10 times. The SInDy method always gives the same result back. For our method, we chose all samples. Best performances are bolded.

Median of RMSE for Dataset4					
Ground Truth Expression	**Genetic Programming**	**SInDy**	**our method**	**Expression**	**Data Range**
Koza-1	41.89	304.41	**40.37**	$x^4 + x^3 + x^2 + x$	U[-5,5]
Koza-2	**231.18**	937.44	304.79	$x^5 - 2x^3 + x$	U[-5,5]
Koza-3	**688.30**	4502.59	2312.24	$x^6 - 2x^4 + x^2$	U[-5,5]
Nguyen-1	11.73	67.68	**10.85**	$x^3 + x^2 + x$	U[-5,5]
Nguyen-2	**11.91**	240.47	12.34	$x^4 + x^3 + x^2 + x$	U[-5,5]
Nguyen-3	79.81	1684.79	**53.23**	$x^5 + x^4 + x^3 + x^2 + x$	U[-5,5]
Nguyen-4	**163.91**	3651.94	None	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	U[-5,5]
Nguyen-5	0.61	**0.60**	62.17	$\sin(x^2)\cos(x) - 1$	U[-5,5]
Nguyen-6	**0.55**	1.02	82.48	$\sin(x) + \sin(x + x^2)$	U[-5,5]
Nguyen-7	1082.81	1466.20	**1098.79**	$\log(x + 1) + \log(x^2 + 1)$	U[0,5]
Nguyen-8	495.35	**433.78**	501.91	$\sqrt{x}$	U[0,5]

Table 5.10: Median of RMSE of the **test dataset** of expressions **Dataset4**. For the GP method, we repeated the experiment 10 times. The SInDy method always gives the same result back. For our method, we chose all samples. Best performances are bolded.

Median of the length of expressions outputs		
Ground truth expression	**Genetic Programming**	**Our Method**
$1 + x * 2$	6	5
$2 * 2 * x + 2$	13	7
$2 * \sin(x)$	8	4
$2 + \cos(\sin(x * x))$	5	5
$\exp(3/\exp(2 + x/2) + x)$	43	10
$x^4 + x^3 + x^2 + x$	94	19
$x^5 - 2x^3 + x$	129	29
$x^6 - 2x^4 + x^2$	127	11
$x^3 + x^2 + x$	93	11
$x^4 + x^3 + x^2 + x$	102	19
$x^5 + x^4 + x^3 + x^2 + x$	85	29
$x^6 + x^5 + x^4 + x^3 + x^2 + x$	42	None
$\sin(x^2)\cos(x) - 1$	7	11
$\sin(x) + \sin(x + x^2)$	25	11
$\log(x + 1) + \log(x^2 + 1)$	146	11
$\sqrt{x}$	115	11

Table 5.11: Comparision of the median length of output expressions with **Dataset1** and **Dataset4**. In *gplearn*, the equation length refers to the total number of nodes in the expression tree, including both operators and variables. In our method, we use the same calculation approach.

the training set. Our method shows the best performance for 4 out of 11 ground truths without the overfitting trend. On the other hand, our method failed to generate a valid expression when the ground truth is Nguyen-4.

Table 5.11 shows the length of the output expressions of Genetic Programming and our method. The equation length refers to the total number of nodes in the expression tree, for example, $x * x * x$ has the length of 5, include three variables and two operators. including both operators and variables In our method, we use the same calculation approach. From the result we can see that through the Genetic Programming method the best performance is obtained for some data sets, but the length of expressions indicates that it tends to overfitting the data. At the same time, our method shows a good performance without the trend of overfitting.

6 Conclusion and future work

6.1 Conclusion

This thesis proposes the application of Bayesian reasoning in the learned latent space, enabling the posterior distribution in the latent space to simultaneously consider both the prior and the likelihood. Additionally, this paper demonstrates the feasibility of mapping prior knowledge, especially domain-specific knowledge, into the latent space as priors through the encoder. When sampling from the posterior distribution in the latent space, we employed the GP-HMC sampling method, constructing a Gaussian Process model of the posterior distribution in the latent space and enhancing the efficiency of sampling.

In this thesis, we explored various VAE approaches to obtain a structured latent space and experimented with and compared different combinations of hyperparameters. We implemented the GP-HMC sampling method and analyzed the impact of various parameters during the actual experimental process, ultimately selecting the parameters and settings that best suited this study. In the final experimental section, we compared our method with Genetic Programming and the SInDy method using self-generated datasets as well as commonly used datasets in the field, such as the Nguyen and Koza datasets. This comparison demonstrated that our approach outperforms the other two methods on some datasets and effectively prevents overfitting.

Other work in this thesis includes constructing the entire method's pipeline, using PCFGs to generate an expressions dataset, and creating an observational dataset under the assumption of a Gaussian process based on the ground truth, which includes observed data. We trained the Character VAE and GVAE using the expressions dataset as input and fine-tuned to select appropriate hyperparameters. We also calculated the likelihood of points in the latent space based on the observational data. The GP-HMC sampling algorithm was implemented and applied to the latent space, and the sampled points were used to generate corresponding mathematical expressions.

6.2 Limitation and future work

However, our research still has areas that could be further improved. First, our method have not considering the coefficient in the mathematic expressions. When using formulas from the field of physics that include coefficients, our method is unable to automatically learn these coefficients. At the end of the project, we made preliminary attempts to output equations with coefficients and select the most appropriate coefficients based on observational data. However, time constraints prevented us from integrating this into the entire method and conducting comprehensive experiments.

Additionally, although we validated the method of mapping prior knowledge into the latent space, due to time constraints and computational limitations in the final experimental phase, we assumed a uniform distribution of priors in the latent space.

Further research directions could include adding functionality to automatically compute appropriate constant terms, and based on this feature, selecting larger sets of equations with more practical physical or disciplinary significance for training.

Bibliography

[Ber] A. E. Berndt. “Sampling Methods”. In: () (cit. on p. 17).

[Bet18] M. Betancourt. *A Conceptual Introduction to Hamiltonian Monte Carlo*. 2018. arXiv: 1701.02434 (cit. on pp. 9, 20, 40).

[CGCB14] J. Chung, C. Gulcehre, K. Cho, Y. Bengio. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. Dec. 11, 2014. DOI: 10.48550/arXiv.1412.3555. arXiv: 1412.3555[cs]. URL: http://arxiv.org/abs/1412.3555 (visited on 01/29/2025) (cit. on p. 15).

[Chi] Z. Chi. “Statistical Properties of Probabilistic Context-Free Grammars”. In: *Computational Linguistics* 25.1 () (cit. on pp. 22, 28).

[CVG+14] K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio. “Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179. URL: http://aclweb.org/anthology/D14-1179 (visited on 10/16/2024) (cit. on pp. 9, 25, 32, 33, 35).

[DS17] R. Dey, F. M. Salem. “Gate-variants of Gated Recurrent Unit (GRU) neural networks”. In: *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS). Boston, MA: IEEE, Aug. 2017, pp. 1597–1600. ISBN: 978-1-5090-6389-5. DOI: 10.1109/MWSCAS.2017.8053243. URL: https://ieeexplore.ieee.org/document/8053243/ (visited on 01/29/2025) (cit. on p. 14).

[FKK+21] U. Fasel, E. Kaiser, J. N. Kutz, B. W. Brunton, S. L. Brunton. “SINDy with Control: A Tutorial”. In: *2021 60th IEEE Conference on Decision and Control (CDC)*. 2021 60th IEEE Conference on Decision and Control (CDC). Austin, TX, USA: IEEE, Dec. 14, 2021, pp. 16–21. ISBN: 978-1-66543-659-5. DOI: 10.1109/CDC45484.2021.9683120. URL: https://ieeexplore.ieee.org/document/9683120/ (visited on 01/29/2025) (cit. on pp. 10, 29, 41, 61, 62).

[GP15] B. M. Gyori, D. Paulin. *Non-asymptotic confidence intervals for MCMC in practice*. Sept. 28, 2015. DOI: 10.48550/arXiv.1212.2016. arXiv: 1212.2016[math]. URL: http://arxiv.org/abs/1212.2016 (visited on 01/29/2025) (cit. on pp. 21, 41, 54, 59).

[Has70] W. K. Hastings. “Monte Carlo sampling methods using Markov chains and their applications”. In: *Biometrika* 57.1 (Apr. 1970). _eprint: https://academic.oup.com/biomet/article-pdf/57/1/97/23940249/57-1-97.pdf, pp. 97–109. ISSN: 0006-3444. DOI: 10.1093/biomet/57.1.97. URL: https://doi.org/10.1093/biomet/57.1.97 (cit. on p. 20).

[Hod22] T. O. Hodson. "Root-mean-square error (RMSE) or mean absolute error (MAE): when to use them or not". In: *Geoscientific Model Development* 15.14 (July 19, 2022), pp. 5481–5487. ISSN: 1991-9603. DOI: 10.5194/gmd-15-5481-2022. URL: https://gmd.copernicus.org/articles/15/5481/2022/ (visited on 01/29/2025) (cit. on pp. 41, 62).

[HZ93] G. E. Hinton, R. Zemel. "Autoencoders, Minimum Description Length and Helmholtz Free Energy". In: *Advances in Neural Information Processing Systems*. Ed. by J. Cowan, G. Tesauro, J. Alspector. Vol. 6. Morgan-Kaufmann, 1993. URL: https://proceedings.neurips.cc/paper_files/paper/1993/file/9e3cfc48eccf81a0d57663e129aef3cb-Paper.pdf (cit. on p. 15).

[KALC22] P.-a. Kamienny, S. d'Ascoli, G. Lample, F. Charton. "End-to-end Symbolic Regression with Transformers". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh. Vol. 35. Curran Associates, Inc., 2022, pp. 10269–10281. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/42eb37cdbefd7abae0835f4b67548c39-Paper-Conference.pdf (cit. on p. 23).

[Koz94] J. R. Koza. *Genetic programming II: automatic discovery of reusable programs*. MIT press, 1994 (cit. on pp. 43, 61).

[KPH17] M. J. Kusner, B. Paige, J. M. Hernández-Lobato. *Grammar Variational Autoencoder*. en. arXiv:1703.01925 [stat]. Mar. 2017. URL: http://arxiv.org/abs/1703.01925 (visited on 04/08/2024) (cit. on pp. 9, 24–26, 29, 36–38).

[KW22] D. P. Kingma, M. Welling. *Auto-Encoding Variational Bayes*. en. arXiv:1312.6114 [cs, stat]. Dec. 2022. URL: http://arxiv.org/abs/1312.6114 (visited on 04/08/2024) (cit. on pp. 16, 32).

[LBH15] Y. LeCun, Y. Bengio, G. Hinton. "Deep learning". In: *Nature* 521.7553 (May 28, 2015), pp. 436–444. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14539. URL: https://www.nature.com/articles/nature14539 (visited on 01/29/2025) (cit. on p. 13).

[MC24] N. Makke, S. Chawla. "Interpretable scientific discovery with symbolic regression: a review". In: *Artificial Intelligence Review* 57.1 (Jan. 2024), p. 2. ISSN: 0269-2821, 1573-7462. DOI: 10.1007/s10462-023-10622-0. URL: https://link.springer.com/10.1007/s10462-023-10622-0 (visited on 04/29/2024) (cit. on pp. 9–12, 23, 24, 27, 29, 41, 61, 62).

[MJ+01] L. R. Medsker, L. Jain, et al. "Recurrent neural networks". In: *Design and Applications* 5.64-67 (2001), p. 2 (cit. on p. 13).

[PLB+23] S. Popov, M. Lazarev, V. Belavin, D. Derkach, A. Ustyuzhanin. *Symbolic expression generation via Variational Auto-Encoder*. en. arXiv:2301.06064 [cs]. Jan. 2023. URL: http://arxiv.org/abs/2301.06064 (visited on 04/22/2024) (cit. on pp. 24, 25).

[PLM+21] B. K. Petersen, M. Landajuela, T. N. Mundhenk, C. P. Santiago, S. K. Kim, J. T. Kim. *Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients*. en. arXiv:1912.04871 [cs, stat]. Apr. 2021. URL: http://arxiv.org/abs/1912.04871 (visited on 04/30/2024) (cit. on p. 23).

[Ras03] C. E. Rasmussen. "Gaussian Processes to Speed up Hybrid Monte Carlo for Expensive Bayesian Integrals". en. In: *Bayesian Statistics 7*. Ed. by V. Lindley, J. M. Bernardo, M. J. Bayarri, J. O. Berger, A. P. Dawid, D. Heckerman, A. F. Smith, M. West. Oxford University PressOxford, July 2003, pp. 651–660. ISBN: 978-0-19-852615-5 978-1-383-02433-3. DOI: 10.1093/oso/9780198526155.003.0045. URL: https://academic.oup.com/book/54032/chapter/422208374 (visited on 04/08/2024) (cit. on pp. 9, 26).

[Rut89] R. Rutenbar. "Simulated annealing algorithms: an overview". In: *IEEE Circuits and Devices Magazine* 5.1 (Jan. 1989), pp. 19–26. ISSN: 8755-3996. DOI: 10.1109/101.17235. URL: http://ieeexplore.ieee.org/document/17235/ (visited on 02/10/2025) (cit. on p. 48).

[RW06] C. E. Rasmussen, C. K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. OCLC: ocm61285753. Cambridge, Mass: MIT Press, 2006. 248 pp. ISBN: 978-0-262-18253-9 (cit. on pp. 18, 19, 26, 39, 40, 54, 57).

[SEE04] M. SEEGER. "GAUSSIAN PROCESSES FOR MACHINE LEARNING". In: *International Journal of Neural Systems* 14.02 (2004). PMID: 15112367, pp. 69–106. DOI: 10.1142/S0129065704001899. eprint: https://doi.org/10.1142/S0129065704001899. URL: https://doi.org/10.1142/S0129065704001899 (cit. on p. 18).

[SM19] R. C. Staudemeyer, E. R. Morris. *Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks*. Sept. 12, 2019. DOI: 10.48550/arXiv.1909.09586. arXiv: 1909.09586[cs]. URL: http://arxiv.org/abs/1909.09586 (visited on 01/29/2025) (cit. on pp. 13, 14, 34).

[STE] C. Szegedy, A. Toshev, D. Erhan. "Deep Neural Networks for Object Detection". In: () (cit. on p. 13).

[TG12] A. Thompson, K. Goyne. "Introduction to the sorption of chemical constituents in soils". In: *Nature Education Knowledge* 4.4 (2012), p. 7 (cit. on p. 60).

[UHO+11] N. Q. Uy, N. X. Hoai, M. O'Neill, R. I. McKay, E. Galván-López. "Semantically-based crossover in genetic programming: application to real-valued symbolic regression". In: *Genetic Programming and Evolvable Machines* 12.2 (June 2011), pp. 91–119. ISSN: 1389-2576, 1573-7632. DOI: 10.1007/s10710-010-9121-2. URL: http://link.springer.com/10.1007/s10710-010-9121-2 (visited on 02/04/2025) (cit. on pp. 43, 61).

[UT20] S.-M. Udrescu, M. Tegmark. *AI Feynman: a Physics-Inspired Method for Symbolic Regression*. Apr. 15, 2020. DOI: 10.48550/arXiv.1905.11481. arXiv: 1905.11481[physics]. URL: http://arxiv.org/abs/1905.11481 (visited on 01/13/2025) (cit. on p. 23).

[VH08] L. Van der Maaten, G. Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008) (cit. on p. 49).

[VSP+23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin. *Attention Is All You Need*. Aug. 2, 2023. DOI: 10.48550/arXiv.1706.03762. arXiv: 1706.03762[cs]. URL: http://arxiv.org/abs/1706.03762 (visited on 02/11/2025) (cit. on p. 23).

[YZ20] T. Yu, H. Zhu. *Hyper-Parameter Optimization: A Review of Algorithms and Applications*. Mar. 12, 2020. DOI: 10.48550/arXiv.2003.05689. arXiv: 2003.05689[cs]. URL: http://arxiv.org/abs/2003.05689 (visited on 01/29/2025) (cit. on p. 12).

[ZC17] N. Zhang, P. Chandrasekar. “Sparse learning of maximum likelihood model for optimization of complex loss function”. In: *Neural Computing and Applications* 28.5 (May 2017), pp. 1057–1067. ISSN: 0941-0643, 1433-3058. DOI: 10.1007/s00521-015-2118-2. URL: http://link.springer.com/10.1007/s00521-015-2118-2 (visited on 02/10/2025) (cit. on p. 51).

Appendix

A Datasets

Dataset1

Expression	Data range
$1 + x * 2$	$U[0, 5]$
$2 * 2 * x + 2$	$U[0, 5]$
$2 * \sin(x)$	$U[0, 5]$
$2 + \cos(\sin(x * x))$	$U[0, 5]$
$\exp(3/\exp(2 + x/2) + x)$	$U[0, 5]$

Table 1: Expressions of Dataset1 and their data range

Dataset2

Expression	Data range
$\cos(x * \exp(1)/2)$	$U[0, 5]$
$x + \cos(exp(1/x * \sin(3/x) + x))$	$U[0, 5]$
$\exp(\cos(x))$	$U[0, 5]$
$2 + (1 + 2)/\cos(\cos(x))$	$U[0, 5]$
$x * 1/2$	$U[0, 5]$
x	$U[0, 5]$
$\exp(x)$	$U[0, 5]$
$\sin(\sin(1))/(2 + x * \sin(2 + 1/1) + \sin(x/x))$	$U[0, 5]$
$x + 3/2$	$U[0, 5]$
$\exp(\cos(2) + x)$	$U[0, 5]$
$x * x$	$U[0, 5]$
$\exp(x)/2 * (3) + \cos(1) * 3/1/3/2$	$U[0, 5]$
$\exp(3/(x) * 3/x)$	$U[0, 5]$
$2 * x$	$U[0, 5]$
$1/x$	$U[0, 5]$
$2 + 2 * 3 + \cos(2/x * 1/1/3)/1$	$U[0, 5]$
$x + 2 * 3/x$	$U[0, 5]$
$\cos((x + 2) * 3) * 2$	$U[0, 5]$
$1 * \sin(x) * 2$	$U[0, 5]$
$1 * \sin(2) + (2 + x + 1 * 1 * 1/2/x)$	$U[0, 5]$

Table 2: Expressions of Dataset2 and their data range

Dataset3

Expressions name	Expression	Data range
Two Site Langmuir	$St * f1k1 * x/(1 + k1 * x) + f2k2 * x/(1 + k2 * x)$	$U[-5, 5]$
Langmuir	$St * 1 * k1 * x/(1 + k1 * x)$	$U[-5, 5]$
Modified Langmuir	$St * 1/(1 + k2 * x) * k1 * x/(1 + k1 * x)$	$U[-5, 5]$

Table 3: Expressions of Dataset3 and their data range

Dataset4

Expressions name	Expression	Data range
Koza-1	$x^4 + x^3 + x^2 + x$	$U[-5, 5]$
Koza-2	$x^5 - 2x^3 + x$	$U[-5, 5]$
Koza-3	$x^6 - 2x^4 + x^2$	$U[-5, 5]$
Nguyen-1	$x^3 + x^2 + x$	$U[-5, 5]$
Nguyen-2	$x^4 + x^3 + x^2 + x$	$U[-5, 5]$
Nguyen-3	$x^5 + x^4 + x^3 + x^2 + x$	$U[-5, 5]$
Nguyen-4	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	$U[-5, 5]$
Nguyen-5	$\sin(x^2)\cos(x) - 1$	$U[-5, 5]$
Nguyen-6	$\sin(x) + \sin(x + x^2)$	$U[-5, 5]$
Nguyen-7	$\log(x + 1) + \log(x^2 + 1)$	$U[0, 5]$
Nguyen-8	$\sqrt{x}$	$U[0, 5]$

Table 4: Expressions of Dataset4 and their data range

B Implementation

All the implementations are in Github: https://github.com/ChenleiPei/Master-Thesis. The repository includes four branches; the branch master includes the most important functions, which include the whole pipeline and the implementation of Genetic Programming and the SInDy method. The branch Character-VAE-with-GPHMC includes the Character VAE training and sampling, while the branch Grammar-VAE-GPHMC includes the training of Grammar VAE. The Toy-Project-of-GPHMC-and-VAE includes the implementation at the beginning of the Master Thesis to understand the Gaussian Progress and Hamilton Monte Carlo Sampling.