

Bayesian Deep Learning for Distilling Physical Laws from Videos

Zixuan Wan

Master of Science Thesis

Bayesian Deep Learning for Distilling Physical Laws from Videos

MASTER OF SCIENCE THESIS

For the degree of Master of Science in Mechanical Engineering at Delft
University of Technology

Zixuan Wan

September 19, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of
Technology

Abstract

An end-to-end framework is developed to discover physical laws directly from videos, which can help facilitate the study on robust prediction, system stability analysis and gain the physical insight of a dynamic process. In this work, a video information extraction module is proposed to detect and collect the pixel position of moving objects, which would be further transformed into physical states we care about. A physical law discovery module is developed to learn closed-form expressions based on the extracted physical information. The video information extraction module takes advantage of contour detection and Hough transformation to extract position information. The physical law discovery module includes a deep neural network-like hierarchical structure Mathematical Operation Network (MathONet) which is consisted of basic mathematical operations. We develop a sparse Bayesian learning algorithm to learn both the topology and parameters of dynamic systems. Several simulated videos were generated to illustrate Newton's law of motion, the state space of a Duffing oscillator, and the pendulum motion equation. By demonstrating on these examples, our method can discover the corresponding governing function without requiring much prior information.

Keywords: Physical law discovery, Video detection, Sparse Bayesian learning

Table of Contents

Acknowledgements	xi
1 Introduction	1
1-1 Background	2
1-2 Motivation and Contribution	2
1-3 Pipeline	4
2 Preliminary	5
2-1 Deep Neural Network	5
2-1-1 Overfitting and Regularization	6
2-1-2 Efficiency and Model Compression	7
2-2 Bayesian Learning	8
2-2-1 Markov Chain Monte Carlo	9
2-2-2 Variational Inference	10
2-2-3 Laplace Approximation	10
2-2-4 Conclusion	11
3 Video Information Extraction	13
3-1 Video Preprocessing	14
3-2 Feature Extraction	16
3-2-1 Contour Detection	16
3-2-2 Hough Transform	17
3-2-3 Comparison between Contour Detection and Hough Transform	17
3-3 Conclusion	19

4	Physical Law Discovery	21
4-1	Multilayer Perceptrons	21
4-2	Model Design	24
4-2-1	Polynomial-Network	24
4-2-2	Operation-Network	25
4-2-3	Model Compression	26
4-3	Dependency between connections	27
4-4	Specific Example	29
4-5	Conclusion	30
5	Sparse Bayesian Learning Algorithm	33
5-1	Laplace Approximation	33
5-2	Evidence Maximization	34
5-3	Regularization Update Rules	35
5-3-1	Conclusion	37
6	Experiment Result	39
6-1	Free Falling Ball	40
6-1-1	Synthetic Dataset	40
6-1-2	Real Dataset	46
6-2	Duffing Oscillator	48
6-2-1	Synthetic Video Generation	48
6-2-2	Physical Information Extraction	49
6-2-3	Physical Law Discovery Experiment Setup	50
6-2-4	Result Analysis	50
6-3	Swinging Pendulum	53
6-3-1	Synthetic Video Generation	53
6-3-2	Physical Information Extraction	54
6-3-3	Experiment Setup	56
6-3-4	Result Analysis	57
7	Conclusion	61
7-1	Discussion	61
7-2	Further Works	62
A	Supplement Experiment result	63
A-1	Free Fall	63
A-1-1	Distilled results of L1 regularization	63
A-1-2	Distilled results of no regularization	64
A-2	Duffing Oscillator	65
A-2-1	Distilled results of L1 regularization	66
A-2-2	Distilled results of no regularization	67
A-3	Swinging Pendulum	68
A-3-1	Experiment 1	68
A-3-2	Experiment 2	68

B Real Dataset for Pendulum	71
B-1 Data Collection	71
B-1-1 ROS framework	71
B-1-2 Python code	72
B-2 Video Information Extraction	75
B-3 Experiment Result	76
Bibliography	77

List of Figures

1-1	Schematic architecture of the end-to-end framework	4
2-1	Two kinds of layers in deep neural networks[21]	5
2-2	Underfitting and overfitting[20]	6
2-3	Comparison between L1 regularization and L2 regularization	7
2-4	Conversion from a dense network to a sparse network	8
2-5	Comparison between a traditional neural network and a Bayesian neural network[7]	9
3-1	Examples of a frame from the simulated or real pendulum video	14
3-2	The difference between the two methods for object recognition	14
3-3	The result of Canny edge detection algorithm	16
3-4	The result of Canny edge detection algorithm	16
3-5	Detected position results of a simulated free falling ball video and a swinging pendulum compared with ideal values	18
3-6	Schematic diagram of physical information extraction process	19
4-1	An example of a multilayer perceptron	22
4-2	Mathematical operation network (MathONet)	24
4-3	The structure of the Polynomial-network (PolyNet) and [Operation-network (Oper-Net)	25
4-4	An example of MathONet after model compression	27
4-5	An illustration for the dependencies between connections.	27
4-6	An example of a specific formula	29
4-7	The identification examples of linear and nonlinear system	30
6-1	Description of the synthetic visual physical dataset	39
6-2	Visualization of the generated synthetic video frame of free falling fall	41

6-3	Comparison between the extracted physical position and the true position	41
6-4	Identified governing equation for free fall dataset	43
6-5	The sparsity and predictive ability of MathONet for synthetics free fall dataset	44
6-6	Sparsity and predictive ability of MathONet with different optimization methods	45
6-7	Visualization of the real dataset of free fall	46
6-8	Detection result for the real free fall dataset	46
6-9	The sparsity and predictive ability of MathONet for real free fall dataset	47
6-10	Trajectory and the simplified video diagram of a Duffing oscillator	49
6-11	Error of extracted y and \dot{y}	50
6-12	The sparsity and predictive ability of MathONet when training the video dataset of Duffing oscillator	51
6-13	Synthetics video of a swinging pendulum	54
6-14	Comparison of the derived angular acceleration $\ddot{\theta}$ between two methods	55
6-15	The detection process of the positional information of the pendulum with contour detection	56
6-16	Sparsity and predictive ability of MathONet for the swinging pendulum dataset	57
6-17	Predicted results of the control group model trained with L1 and no regularization	58
A-1	The sparsity and predictive ability of MathONet trained with L1 regularization for the dataset of a free falling ball	64
A-2	The sparsity and predictive ability of MathONet trained with no regularization for the dataset of a free falling ball	65
A-3	The sparsity and predictive ability of MathONet trained with L1 regularization for the dataset of a Duffing oscillator	66
A-4	The sparsity and predictive ability of MathONet trained with no regularization for the dataset of a Duffing oscillator	67
A-5	Sparsity and predictive ability of MathONet for the swinging pendulum dataset (Experiment 1)	68
A-6	Sparsity and predictive ability of MathONet for the swinging pendulum dataset (Experiment 2)	69
B-1	A screenshot of the video dataset of a real pendulum	71
B-2	The process of video information extraction of the real pendulum video	75

List of Tables

2-1	Comparison between different Bayesian learning methods	11
3-1	Mean square error of the detected results of two detection methods on two simulated videos generated from the motion of a free falling ball and a swinging pendulum.	18
6-1	Different experiment setups for free fall dataset	42
6-2	Mean square error of the detected results of two detection methods	45
6-3	Mean square error of the detected results for the four variables	50
6-4	Different experiment setups for Duffing oscillator dataset	50
6-5	The importance analysis for different expression terms in the second distilled equation	52
6-6	The validation loss and the number of training cycles for distilled equations from two dataset with different optimization methods	53
6-7	Mean square error of the detected results for the three variables with two detection methods	55
6-8	Different experiment setups for swinging pendulum dataset	56
6-9	Validation loss and number of cycles needed to train MathONet for different optimization methods	58
A-1	Experiment setup for free fall dataset trained with L1 and no regularization	63
A-2	Experiment setup for free fall dataset trained with L1 and no regularization	65

Acknowledgements

I would like to express my gratitude for my supervisor Dr. Wei Pan. Thank you for giving me the chance to do a whole academic research that I have never experienced before. I want to thank my daily supervisor, Ir. Hongpeng Zhou. Your encouragement and support helped me a lot when I encountered problems and got lost.

Thanks to my friends for your accompany during my 2-year study in Delft. Every moment spent with you is a memory worthy of nostalgia.

Immense gratitude to my parents for your effortless support and love.

Delft, University of Technology
September 19, 2021

Zixuan Wan

“We shall meet in the place where there is no darkness.”

— *George Orwell*

Chapter 1

Introduction

Understanding the physical law is of great importance for that it can facilitate the study on robust prediction, system stability analysis and gain the physical insight of a dynamic process [47]. With the development of machine learning techniques and the diversity of sensors, data richness and accuracy have been improved. This gives rise to a new paradigm of discovering the underlying physical laws from data which is also called data-driven governing equation discovery [46, 12]. Considering that cameras have been widely utilized to record scientific videos and collect data, discovering physical laws or governing functions from a raw video has led to great interest in recent years [14].

The aim of this thesis is to create an end-to-end framework that can distill the laws of physics including the topology and parameters from a video by using a novel neural network-like structure Mathematical Operation Network (MathONet). The whole pipeline includes a video information extraction module and a physical law discovery module. In the video information extraction module, the pixel position of a moving object in the frame would be extracted. Then, the video information will be transformed into physical states we care about (e.g., distance, velocity, and angle). In the physical law discovery module, a novel deep neural network like hierarchical structure is designed, termed as MathONet. MathONet is initialized redundantly and composed of various possible basic mathematical operations that can contribute to the mathematical expression. The idea of Bayesian deep learning is included in the network optimization process for discovering the sparse optimal structure from an over-parameterized initialized model to represent the explicit mathematical expression. The end-to-end framework is demonstrated on the video of a free falling ball, a Duffing oscillator and a swinging pendulum. A few assumptions are made to handle the underdetermined nature of distilling physical laws from videos. First, the dynamic system has only one moving object of interest. That is, we will not consider a dynamic system containing a group of objects. Second, the the object to be explored in the video is known, especially when the background environment is complex and contains other inferring objects. Third, it is assumed that the variables of the physical law to be explored are known. In other words, the proposed framework can discover the mathematical expression among variable of interest.

The following section of this chapter will introduce the background information about this

topic. Then, the motivation and contribution of this work will be given, followed by the pipeline of the whole video to physical law framework.

1-1 Background

Data-driven physical law discovery represents building explicit mathematical expressions for dynamic systems given measured physical states. The early work begins with identifying linear models from data, followed by the interest for modelling non-linear systems[49]. One important breakthrough in this area is from [50], which implemented symbolic regression [19] to distill the underlying governing functions for even chaotic dynamic systems. Till now, symbolic regression has spawned a family of work. According to the definition for [52], symbolic regression refers to all methods which aim to learn a symbolic expression that matches data from an unknown function, which includes several interesting techniques such as sparse regression[9], genetic programming[2] and network-based symbolic regression[37]. However, these methods have their own disadvantages. For instance, genetic programming is computationally expensive and sparse regression requires prior information to choose a set of basis functions.

Instead of recovering the governing functions from given physical states, understanding a video to learn parameters for well-characterized physical equations has been a topic in recent years. [4] and [40]'s work estimates the parameters for a given mathematical structure. Fragkiadaki et al. further incorporated the known external physical engines with the agent to simulate a billiard game[18]. However, although these approaches can even predict the system precisely, they still require the prior of a well-characterized governing equation.

Recently, there has been attempts to uncover the closed-form governing equations from a raw video. [34] successfully distills first-order and second-order ordinary differential equations from videos through learning the spatial-physical transformation and implementing sparse regression toolbox. On the other hand, Chari et al. uncovers dynamic equations regarding gravitational and centripetal acceleration by adopting the variational auto encoder to determine the latent variables such as speed and acceleration[13]. They utilized existing genetic programming approach to discover the physical law from latent variables. In these works, the physical law is explored by using existing symbolic regression approaches (e.g., sparse regression, genetic programming), which may limit their application due to the inherent shortcomings of these methods. More detailed discussion on these symbolic regression approaches is in Section 1-2.

1-2 Motivation and Contribution

As introduced in the beginning of this chapter, existing approaches to explore physical laws from videos are mainly based on genetic programming or sparse regression. Since these symbolic regression methods have their own drawbacks, which may limit the performance of these methods in their application on physical law discovery from video. Especially, symbolic regression can be divided into four categories:

- **Brute force** This is the most intuitive way to discover the relationship among variables. Given the variables of interest and possible operations, the brute force method will try all possible mathematical expressions in the form of symbolic strings[52] or symbolic tree[28]. It can be easily understood that the brute force method is computational expensive which prevents it from being implemented for large systems. To reduce the computational burden, the work from [52] leverages the properties such as symmetries, separability and compositionality of the physical dataset to narrow the search space and simplify the discovering process.
- **Genetic programming** This is a commonly used method in the field of physical law discovery. Genetic programming is inspired by biology and includes operations such as mutation, selection, and crossover when doing a targeted search for symbolic expression. The search will evolve in the direction of satisfying the fitness criteria. Nevertheless, genetic programming still has limitations such as high computational complexity, over-parameterized output expression[9] and sensitivity to initial conditions[30].
- **Sparse regression** As another widely used approach to search mathematical expressions, sparse regression uses a different searching strategy from brute force or genetic programming. With an over-redundant basis function library, this method would distinguish the essential components with a linear sparse regression. A benchmark work is proposed by [9], which presents the sparse identification of nonlinear dynamics (SINDy) algorithm. Although SINDy has been successfully implemented on many dynamic systems, it suffers from the nontrivial task of choosing appropriate basis functions, which requires an amount of prior information.
- **Network-based symbolic regression** Due to the powerful data fitting capability, neural networks architectures are also used for discovering governing functions. Works from [37] and [44] proposed network-based structures which can act as an equation learner and be integrated with deep learning frameworks to be trained with backpropagation. These methods include elementary operators as the activation functions. However, these network-based approaches cannot fit the constant term in the equation. Besides, they use L1 regularization for training and compressing redundant connections.

Motivated by these symbolic regression methods, we designed a novel DNN-like hierarchical structure to serve for the physical law discovery module in the end-to-end framework. Our main contributions are as follows:

- **End-to-end framework design** We designed an end-to-end framework which can distill physical laws from raw videos directly, without requiring much prior knowledge and the true labelled data.
- **Mathematical operation network design** A deep neural network like hierarchical structure, termed MathONet, is designed to represent the mathematical expression of the governing functions of dynamics systems. The MathONet is built by stacking unary (log, sin) and binary (+, -, ×) operations.
- **Bayesian learning discovery algorithm** A Bayesian learning algorithm is included in the training process of MathONet, which can provide a sparse solution by reducing the redundancy of an initialized over-parameterized network and relieve the burden in hyperparameter tuning.

1-3 Pipeline

The pipeline of our end-to-end framework is as shown in Figure 1-1. The input is only the raw video. Then, the video information extraction module will extract the positional information of the moving object from the consecutive frames. After that, the extracted pixel coordinates would be transformed into physical states, which act as the input and output variables to train the physical law discovery network, namely, MathONet. A Bayesian learning algorithm is implemented to compress the over-redundant model and discover the sparse solution to represent the explicit mathematical expression of the physical law, which is the final output of the whole framework.

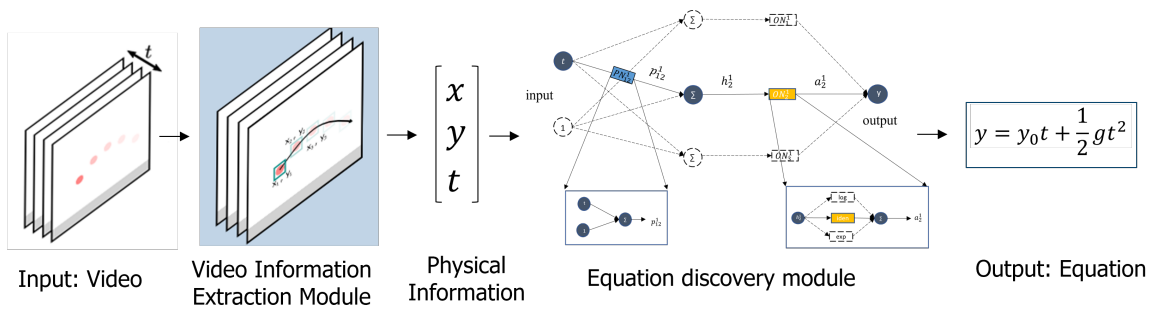


Figure 1-1: Schematic architecture of the end-to-end framework

Preliminary

This chapter will give a brief introduction to the background knowledge in this thesis. First, the principle and properties of deep neural networks will be reviewed, which is the foundation of MathONet. Second, how Bayesian learning works and the comparison between several typical Bayesian learning approaches will be introduced.

2-1 Deep Neural Network

As an important subset of machine learning algorithms, deep neural network(DNN) aims at discovering multiple levels of distributed representations[21]. Especially, DNN has become an indispensable tool for modelling complex nonlinear systems and has been applied for several domains such as computer vision, language processing and control theory[29, 15, 27]. Examples of a fully-connected layer and a convolutional layer are as shown in figure 2-1. Though deep neural networks are of great importance in the field of system modelling, overfitting and efficiency are two frequently encountered problems.

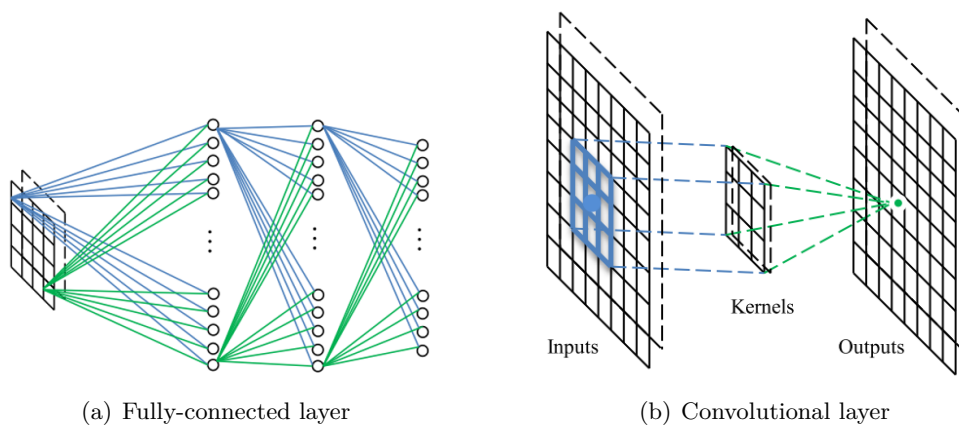


Figure 2-1: Two kinds of layers in deep neural networks[21]

2-1-1 Overfitting and Regularization

Since the field of deep learning has been focusing on achieving high accuracy, deeper models and more complex models appear one after another, which would bring the problem of overfitting. Figure. 2-2 can help understand the relationship among dataset, model complexity and prediction accuracy. When fitting the dataset with three different models, a linear model is the simplest one but it cannot capture the curvature of the data, which leads to underfitting. A quadratic function fits the dataset well, while a more complex polynomial of degree 9 suffers from overfitting because it cannot be generalized to unseen points. It can be concluded that there is a balance between model complexity and prediction accuracy. In the deep neural network domain, regularization can help beat overfitting and reduce the generalization error for the test dataset.

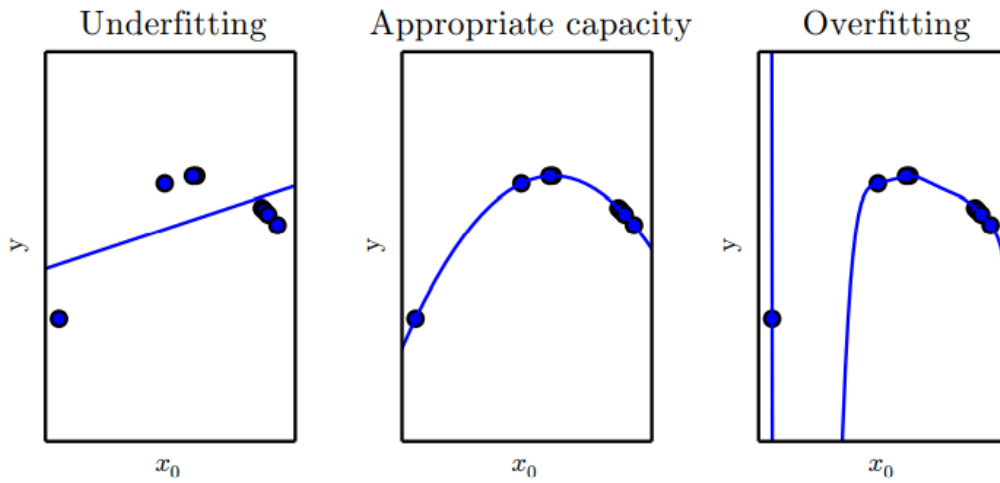


Figure 2-2: Underfitting and overfitting[20]

Regularization is defined as "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error" [20]. Though there are several methods that can help reduce overfitting like data augmentation, dropout and early stopping, normally regularization refers to strategies adding restrictions or penalties in the objective function. With W , \mathbf{x} and \mathbf{y} represent the weight matrix, input data and output data, $L(W; \mathbf{x}, \mathbf{y})$ indicates the loss function while \tilde{L} is the loss function with regularization, as shown in (2-1).

$$\tilde{L}(W; \mathbf{x}, \mathbf{y}) = L(W; \mathbf{x}, \mathbf{y}) + \alpha\Omega(W) \quad (2-1)$$

α is a hyperparameter that determines the contribution of the norm penalty term Ω . A larger α corresponds to more regularization. When training with the regularized objective function, both the original loss and parameter norm item would be decreased. Different choices of Ω will result in corresponding optimal solution W^* . Typically, there are two norm terms: $L1$ and $L2$.

$L1$ regularization term is defined as $\Omega(W) = \|W\|_1 = \sum_i |w_i|$, namely, the sum of the absolute values of all the weights. $L2$ parameter norm penalty is commonly known as weight decay. This regularization strategy adds a penalty term $\Omega(W) = \frac{1}{2}\|W\|_2^2$ to the optimization function, driving the weights closer to the origin.

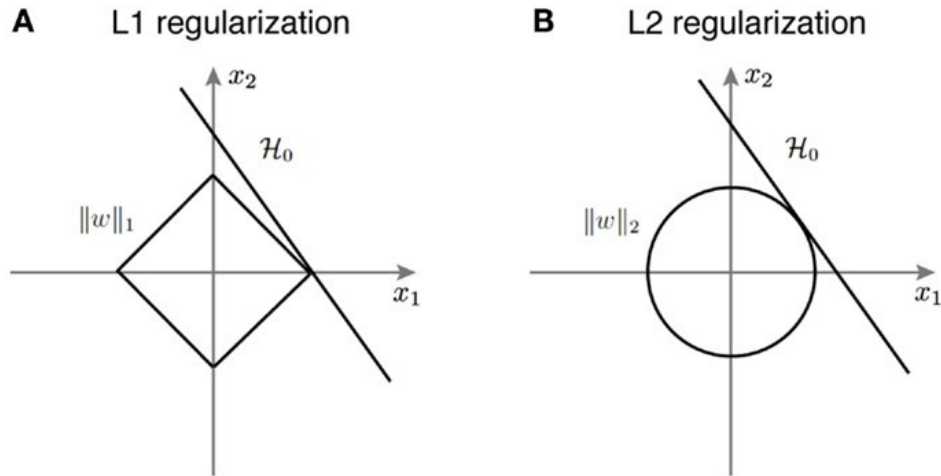


Figure 2-3: Comparison between L1 regularization and L2 regularization

Figure 2-3 gives an intuitive explanation about how $L1$ and $L2$ influence the final solution of parameters. In this simple case, the weight space has 2 dimensions while H_0 is the hypothesis and the solution is the points where H_0 meets the constraints. Due to the nature of $\|W\|_1$, the possible solutions will be limited in corners, thus lead to sparsity in the final weight set. As for $L2$ regularization, the hypothesis is tangential to $\|W\|_2$, the intersection has non-zero values both in x_1 and x_2 axis, but the values tend to approach the origin.

2-1-2 Efficiency and Model Compression

Complex models not only lead to overfitting, but also require extended and expensive training which prevent such models from being used by mobile devices. The model compression technique can not only improve the efficiency of memory and energy consumption, but also leads to less network bandwidth, faster processing, and better privacy [53]. Since model compression has raised a surge in the deep learning community, there are a number of successful compression algorithms. Typically, they can be categorized into three class: standard weight pruning, fixed point precision and Bayesian compression. Specifically, these three classes are not mutually exclusive. For example, Bayesian compression needs to prune redundant weights and it can reduce the bit size of weights at the same time.

1. **Standard weight pruning** It has been proven that there is a great redundancy in the parameters of a DNN [17, 54]. In this case, weight pruning, which was first proposed to improve generalization ability, can also be used for model compression and speed-up[31]. As the name gives away, weight pruning represents removal of redundant weights. Thus, improving the compression rate as much as possible without sacrificing accuracy is a topic worth exploring. Some experiments showed that neural networks can prune up to 99% parameters without significant loss of accuracy [24, 22].

Redundant weights can be pruned according to the ranking of how much each weight contribute to the final accuracy. Usually there are two common methods: pruning synapses and pruning neurons (Figure 2-4). As an alternative of pruning at once,

iterative pruning can repeat the process of training and pruning to avoid removing connections prematurely [24]. The result showed that it can boost compression rate 13x on VGG-16 without any loss of accuracy.

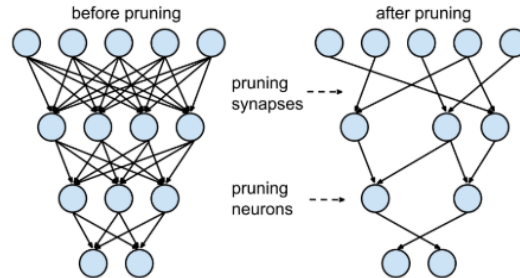


Figure 2-4: Conversion from a dense network to a sparse network

2. **Fixed point precision** Another approach to model compression is to reduce the bit size per weight. For example, reducing 32 bit floats to 1 bit leads to a $32\times$ storage improvement [53]. It has been demonstrated that neural networks are robust against certain amount of low precision, and there are several experiments proving that reducing the weight encoding precision has little influence on the prediction accuracy [38, 23, 16].
3. **Bayesian approaches** Bayesian theory states that the posterior probability is related to likelihood and prior. If replacing likelihood estimation with posterior probability estimation, the result will contain prior information about the dataset. Intuitively, this kind of approaches are more mathematically convincing, and they have been proven to have outstanding compression performance [56, 39].

2-2 Bayesian Learning

As introduced in the previous section, although DNNs can surpass humans in various domains, they are generally prone to overfitting on datasets with a small number of training examples and a complex model. In addition, the nature of the neural network determines that it is a black box model. To put it another way, the entire model lacks interpretability due to its complexity. In this case, a black box model that can give accurate predictions is meaningless. As illustrated before, we will design a novel model to explore closed-form mathematical expressions from data in this thesis. This model also has a neural network like structure which may suffer from overfitting. The Bayesian method can serve as a natural and elegant solution to address this issue because it yields posterior distribution instead of point estimation for each weight, making it convincing to distinguish the necessary connections and playing a role of information screening [6, 36, 1, 41]. Figure 2-5 illustrates the difference between traditional neural networks and neural networks with Bayesian learning. All weights in the traditional neural networks are specific values which will be updated through gradient descent of the loss function. However, Bayesian learning provides a statistical modelling method for learning from data because weights are calculated as posterior distributions. In other words, uncertainty is introduced in the weight of the neural network.

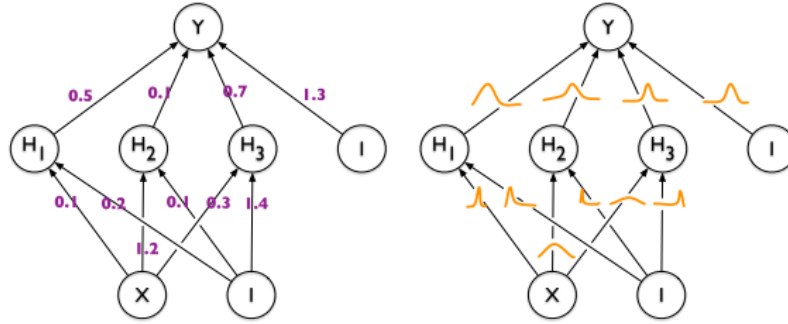


Figure 2-5: Comparison between a traditional neural network and a Bayesian neural network[7]

Specifically, for the latent variables W (including all the weights and biases to be optimized) that are of interest in a model, a prior distribution $p(W)$ could be specified depending on the basic estimation before observing the data \mathcal{D} . The likelihood can be represented as $p(\mathcal{D}|W)$. According to the Bayesian theory, the posterior distribution can be calculated as:

$$p(W|\mathcal{D}) = \frac{p(\mathcal{D}|W)p(W)}{p(\mathcal{D})} \quad (2-2)$$

$$\propto p(\mathcal{D}|W)p(W)$$

where the distribution of data \mathcal{D} is independent of parameter W so the term $P(\mathcal{D})$ can be removed. Although the main concept is easy to understand, the posterior distribution for machine learning models is always intractable. In order to perform Bayesian inference practically, several approaches have been proposed over the past decades[36, 26], including Markov Chain Monte Carlo (MCMC)[41], variational inference (VI)[33] and Laplace approximation[51].

In the following part, these typical approximation approaches for Bayesian learning will be introduced with their pros and cons.

2-2-1 Markov Chain Monte Carlo

As a sampling-based method, Markov Chain Monte Carlo (MCMC) can converge to the true posterior distribution after enough iterative sampling process. The operating process of MCMC starts from drawing the random parameters W_0 from the initial distribution $p(W_0)$ or $p(W_0 | \mathcal{D})$. Then a stochastic transition operator would be applied to the randomly chosen parameters so as to optimize W_t iteratively:

$$W_t \sim q(W_t | W_{t-1}, \mathcal{D}) \quad (2-3)$$

With a suitable transition function and sufficient operating times, the final result W_T will converge to the exact posterior $p(W | \mathcal{D})$. However, the drawback of MCMC is also obvious that it is hard to estimate how many times is sufficient for convergence and the training process will take a long time in practice.

2-2-2 Variational Inference

Variational inference uses the mathematical idea of variation, estimating the true posterior distribution $p(W|\mathcal{D})$ with a fixed form of distribution $q_\phi(W)$. KL divergence between the true posterior and the estimated posterior will be reduced by minimising the loss function. The KL divergence can be represented as:

$$\begin{aligned} D_{KL}(q_\phi(W)||p(W | \mathcal{D})) &= \int q_\phi(W) \log \frac{q_\phi(W)}{p(W | \mathcal{D})} dW \\ &= \mathbb{E}_{q_\phi(W)} \log \frac{q_\phi(W)}{p(W | \mathcal{D})} \end{aligned} \quad (2-4)$$

After applying Bayes's rule to $p(W | \mathcal{D})$ with $p(W)$ denoting the prior, Eq. 2-4 can be rewritten as:

$$\begin{aligned} D_{KL}(q_\phi(W)||p(W | \mathcal{D})) &= \mathbb{E}_{q_\phi(W)} \log \frac{q_\phi(W)}{p(\mathcal{D} | W)p(W)} p(\mathcal{D}) \\ &= D_{KL}(q_\phi(W)||p(W)) - \mathbb{E}_{q_\phi(W)} \log p(\mathcal{D} | W) + \log p(\mathcal{D}) \\ &= -\mathcal{L}(\phi) + \log p(\mathcal{D}) \end{aligned} \quad (2-5)$$

where \mathcal{L} is known as the evidence lower bound (ELBO). Considering that $\log p(\mathcal{D})$ is non-negative and our aim is to minimize the KL divergence, \mathcal{L} is the lower bound of $\log p(\mathcal{D})$. Therefore, the optimization objective is to maximize ELBO:

$$\mathcal{L}(\phi) = \mathbb{E}_{q_\phi(W)} \log p(\mathcal{D} | W) - D_{KL}(q_\phi(W)||p(W)) \quad (2-6)$$

The advantage of variational inference is that it has a specific optimization objective and can be faster in most cases [48]. However, since it is an approximation method, the optimization result of VI will always have errors.

2-2-3 Laplace Approximation

Similar to variational inference, Laplace approximation takes use of a Gaussian approximation to a probability density defined over a set of continuous variables [5]. To facilitate the explanation of the basic idea behind Laplace approximation method, a simple distribution $p(z)$ will be used as an example. How to learn from this idea for estimating the distribution of parameters in a model will be thoroughly clarified in chapter ???. Supposing $p(z)$ is defined by:

$$p(z) = \frac{1}{Z} f(z) \quad (2-7)$$

where Z is the unknown normalization coefficient. Laplace method will use a mode z_0 of the distribution as the mean value of the approximated Gaussian distribution $q(z)$, which means that:

$$\left. \frac{df(z)}{dz} \right|_{z=z_0} = 0 \quad (2-8)$$

Since the logarithm of a Gaussian distribution is a quadratic function of variables, a second-order Taylor expansion of $\ln f(z)$ centered on mode z_0 can be written as:

$$\ln f(z) \simeq \ln f(z_0) - \frac{1}{2} A (z - z_0)^2 \quad (2-9)$$

where

$$A = - \left. \frac{d^2}{dz^2} \ln f(z) \right|_{z=z_0} \quad (2-10)$$

Due to the reason that z_0 is the local maximum value of $p(z)$, the first-order term in the Taylor expansion does not appear. With the exponential, it can be obtained that:

$$f(z) \simeq f(z_0) \exp \left\{ -\frac{A}{2} (z - z_0)^2 \right\} \quad (2-11)$$

Then the estimated Gaussian distribution $q(z)$ can be acquired with the help of the normalization of a Gaussian:

$$q(z) = \left(\frac{A}{2\pi} \right)^{1/2} \exp \left\{ -\frac{A}{2} (z - z_0)^2 \right\} \quad (2-12)$$

2-2-4 Conclusion

To make a conclusion about these methods, each has both advantages and disadvantages. As shown in figure 2-1, MCMC is asymptotically exact to the true distribution, but it is also computationally expensive and not compatible with deep learning frameworks such as PyTorch and TensorFlow. As two approximation methods, both VI and Laplace approximation are fast and can be implemented on mainstream deep learning frameworks. At the same time, both can only give the approximate solution. The difference between them is that the Laplace approximation method can be employed for deep neural network models. But the VI method can only works for shallow neural networks. Besides, Laplace approximation includes the Hessian matrix which will help optimize and compress a network[8]. However, the extra drawback of Laplace is that it is sensitive to the initialization because the calculation of mean and variance depends on the decision of mode, making it necessary to do repeated experiments to avoid the local optimal solution.

Table 2-1: Comparison between different Bayesian learning methods

Bayesian Learning Method	Advantages	Disadvantages
Markov Chain Monto Carlo	Asymptotically exact	1. Computationally expensive 2. Not compatible with deep learning frameworks
Variational Inference	1. Fast to implement 2. Compatible with deep learning frameworks	1. Approximate result 2. Not suitable for deep networks
Laplace Approximation	1. Fast to implement 2. Compatible with deep learning frameworks 3. Hessian matrix helping optimization	1. Approximate result 2. Sensitive to initialization

For the work of this thesis, we adopt Laplace approximation method for that it is easy to deploy and we can use the concept of cycle to accelerate training (more details about how to implement Laplace approximation for our model are in Chapter 6).

Video Information Extraction

Given the video of a moving object, it is necessary to extract the physical information from the frames for identifying the system dynamics. This chapter will discuss about how to extract the positional information of the moving objects in the video. It should be noted that the extracted pixel position has no physical meaning. Therefore, a transformation between the pixel coordinates and the physical states of interest needs to be applied. However, the transformation process differs a lot for different dynamic systems and may require some prior information. The specific physical state transformation process for each testing example will be introduced in the experiment result section.

Since the aim of our end-to-end framework is to discover the physical law, the underlying assumption is that the system governing equation can be reflected in the dynamics of an object across different time steps. Therefore, an efficient and robust object detection algorithm is required to determine the localization of the moving object. There are several methods to realise the physical information extraction function. For example, a pretrained Mask R-CNN[25] was deployed to extract the bounding box of the moving object and the center of the bounding box can be considered as the object location[13]. L. Luan's work took use of a semantic segmentation model (i.e., U-Net[45]), whose result would be encoded into the pixel coordinate of the object[34]. These two schemes take the advantage of deep neural networks and have a strong generalization ability for different inputs. However, training and deploying a large neural network for pose detection or semantic segmentation always requires a large amount of computation resources. Besides, labelling the dataset for network training also demands significant time cost. Under this circumstance, some traditional methods for image processing and feature extraction are of great necessity, especially considering the assumption that only a small part of each frame is the region of interest.

In this chapter, I would like to introduce the methods for video preprocessing to highlight the main area of the moving object. Then an edge detection algorithm would be applied to discover the target contour which could be followed by a feature extraction part to describe the position of the target object.

3-1 Video Preprocessing

It is always easy for the human eye to recognize the moving object from a video. However, each frame of a video is processed as a digit matrix by a computer. How to help computers understand the picture and detect the area of importance would be an interesting topic. Figure 3-1 gives the examples of a simulated and a real pendulum system. In figure 3-1(a), the simulated red pendulum should be distinguished from the white background, while in figure 3-1(b), the green real pendulum needs to be identified from the complex background environment.

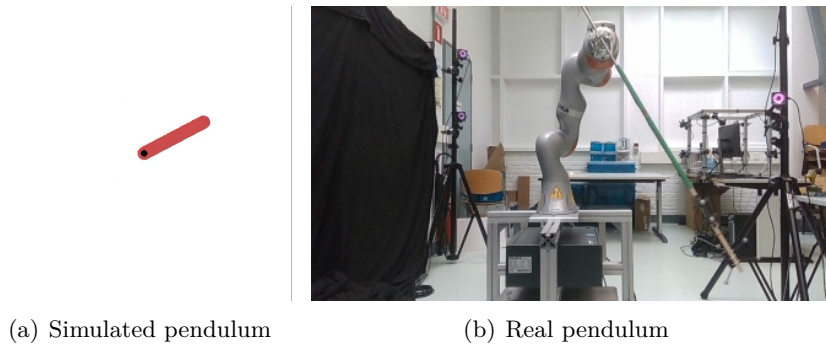


Figure 3-1: Examples of a frame from the simulated or real pendulum video

There are two methods to solve the problem of object recognition. The first one is to utilize the feature that the color of the pendulum is different from the background color so that a mask can be built according to a color filter. The second method is to subtract the background image from the frame of the dataset video. To make the explanation clear, an artificial image was rendered to represent a ball with a complex background environment.

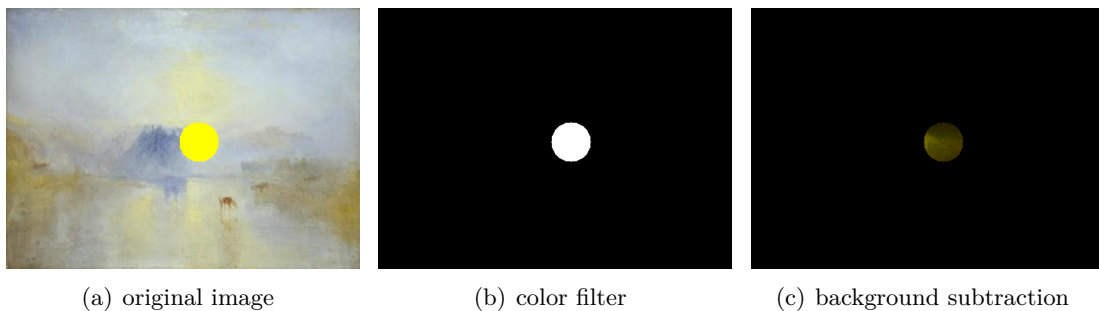


Figure 3-2: The difference between the two methods for object recognition

As shown in figure 3-2(a), a yellow ball is generated with a background of an abstract painting. Method one would detect the ball through traversing the entire image to detect the pixels with similar HSV color of the target object. The result is as illustrated in figure 3-2(b) which is a binary mask with the white area indicating the target object. At the same time, figure 3-2(c) gives the result of subtracting the background image from the original dataset. Because that background part is not changed, it is set to black while the color different in

the yellow area is reflected. Both methods have their own advantages and are suitable for different cases. For instance, Method one requires the target object has a relatively uniform color distribution that can also be distinguished from the environment. For method two, a background photo that is strictly same as the video background environment is necessary. Simulated dataset with an artificial moving object is suitable for method one. However, a real video with limited picture quality and interference factors in the background would have a more accurate detection result with method two.

Algorithm 1 Canny Edge Detection

- 1: Transfer the three-channel RGB image into a one-channel gray-scale image.

$$Value = 0.299 * R + 0.587 * G + 0.114 * B$$

- 2: Applying a Gaussian blur to get rid of the noise on the image. The Gaussian filter matrix would be generated with the following distribution function:

$$g(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where x and y indicate the position in the Gaussian matrix, and σ is variance.

- 3: Calculate the gradient of pixels' intensity to detect the edge intensity and direction. The derivatives I_x and I_y w.r.t x and y can be calculated through convolving the image I with Sobel kernels K_x and K_y . Then the magnitude G and the slope θ of the gradient can be calculated.

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

$$|G| = \sqrt{I_x^2 + I_y^2}$$

$$\theta(x, y) = \arctan\left(\frac{I_y}{I_x}\right)$$

- 4: Non-maximum suppression would be performed to thin out weak edges. This step goes through all points on the gradient intensity matrix and find the local maximum value in the corresponding edge direction detected in the previous step.
 - 5: A double threshold detection would identify whether each pixel contributes to the final edge according to the magnitude G and the two thresholds. Strong pixels whose intensity higher than the high threshold and weak pixels whose intensity smaller than the low threshold would be determined as edge points and non-relevant points. Those medium pixels whose values are between the two thresholds would be augmented into strong pixels if they are connected to strong pixels or if they can be connected to a strong pixel by passing through a medium pixel.
-

Till now, the region of interest has been distinguished from the background environment. The next step is to determine the shape of the target object to find the centroid (assuming the object mass is evenly distributed) or to extract the specific features such as the slope (if the angle of the target object is a parameter that should be considered). Canny edge detection is

one of the most widely used edge detection algorithms and has been proved to have the best performance in most cases[11, 43]. The specific calculation process is shown in Algorithm 1

Figure 3-3 shows the processing result of Canny edge detection algorithm. The left image is the original one while the right image shows the detection result. All edges including the small ones can be represented on the result image. However, most of the edges in the example image are not important and do not contribute to the necessary information such as centroid position and slope. The solution is to extract the main features from the detected edge pattern.

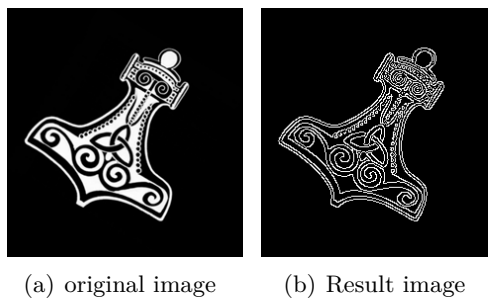


Figure 3-3: The result of Canny edge detection algorithm

3-2 Feature Extraction

Two methods can be used for feature extraction from the detected edges, which are contour detection and Hough transform. Both methods have their own advantages and are suitable for different application scenarios. This section will introduce the principles of two methods and their characteristics.

3-2-1 Contour Detection

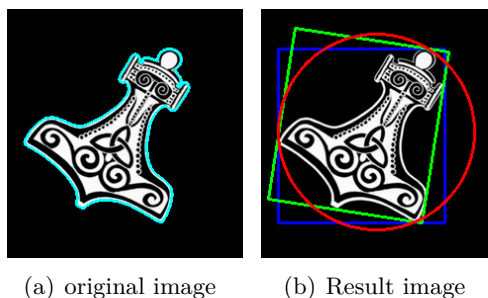


Figure 3-4: The result of Canny edge detection algorithm

Contours can be explained as a curve joining all continuous points with same intensity. This theory can help detect closed curves on the edges. Usually the outermost contour is the true shape of the object. Figure 3-4(a) uses the blue line to indicate the shape of the irregularly

shaped pattern, which can be further processed into a minimum enclosing circle or rectangle (as shown in figure 3-4(b)). This method is especially suitable for an irregular target object whose center position is difficult to determine because that the enclosing regular pattern can act as the ideal situation to simplify the problem.

3-2-2 Hough Transform

Hough transform is another widely used method for pattern recognition, especially for line and circle recognition[3]. Through transforming image space into parameter space, Hough transform takes parameters as variables to fit the given data to the specified shape. Taking straight line detection as an example, the formula $y = kx + b$ can be interpreted as a point (k, b) in the parameter space. Then all points from the original straight line have their corresponding straight lines in the parameter space which will intersect at a point representing the correct parameter. To avoid situations where the slope is infinite, Hesse norm form $r = x \cos \theta + y \sin \theta$ is used so that Hough line detection can be generalized for all straight lines in a plane. The principle of using Hough transform for circle detection is similar. The formula $r^2 = (x - x_0)^2 + (y - y_0)^2$ contains three parameters, which means that all the corresponding curve in the 3D parameter space would intersect at the correct point. It is worth mentioning that for practical scenarios, all edge points will participate in the transform from pixel space to parameter space and vote for the final intersection point. This process makes the method robust because it ignores small noises and repairs disappeared or misplaced pixels on the edges. To put it another way, Hough transform is especially suitable for detecting obvious regular patterns from noisy environment.

3-2-3 Comparison between Contour Detection and Hough Transform

In order to compare the effects of these two detection methods, two simulated videos were generated, which are a free falling ball with initial velocities and a swinging pendulum.

Since the free fall equation is related to the position of the ball in both x and y directions, the precision of the position information detected in these two directions will affect the accuracy of the equation exploration. Figure 3-5(a) and 3-5(b) shows the detected distance of the two methods compared with the desired values. In general, the detection results of these two methods are both precise and close to the real value except that for some frames the detected positions have obvious noise (the enlarged area in the figures), where contour detection has a better performance. This may be because Hough circle detection is more sensitive to noise. This level of error for both methods is acceptable, because the overall error is maintained at a low level, as shown in table 3-1.

Furthermore, a video of a swinging pendulum is used to test the difference between Hough line detection and contour detection. Since Hough transform can fit the given data into a specific shape, the slope can be extracted directly as the swing angle for a pendulum. On the other hand, contour detection can only give the minimum enclosing rectangle, which could be further processed to determine the rotation angle. Figure 3-5(c) compares the detected swing angle of the simulated pendulum of the two methods with the real values. It can be concluded that both methods perform well except that when the angular velocity of the pendulum changes drastically there will be some small detection errors. The last row of

table 3-1 also proves that the mean square errors of both methods are in the same order of magnitude.

To sum up, both methods perform well on the simulated videos. Nevertheless, for real videos, contour detection and Hough transform have their pros and cons, so they have more suitable conditions. It is also worth noting that in the process of generating videos from accurate data, noise has been encoded due to the image resolution limitation.

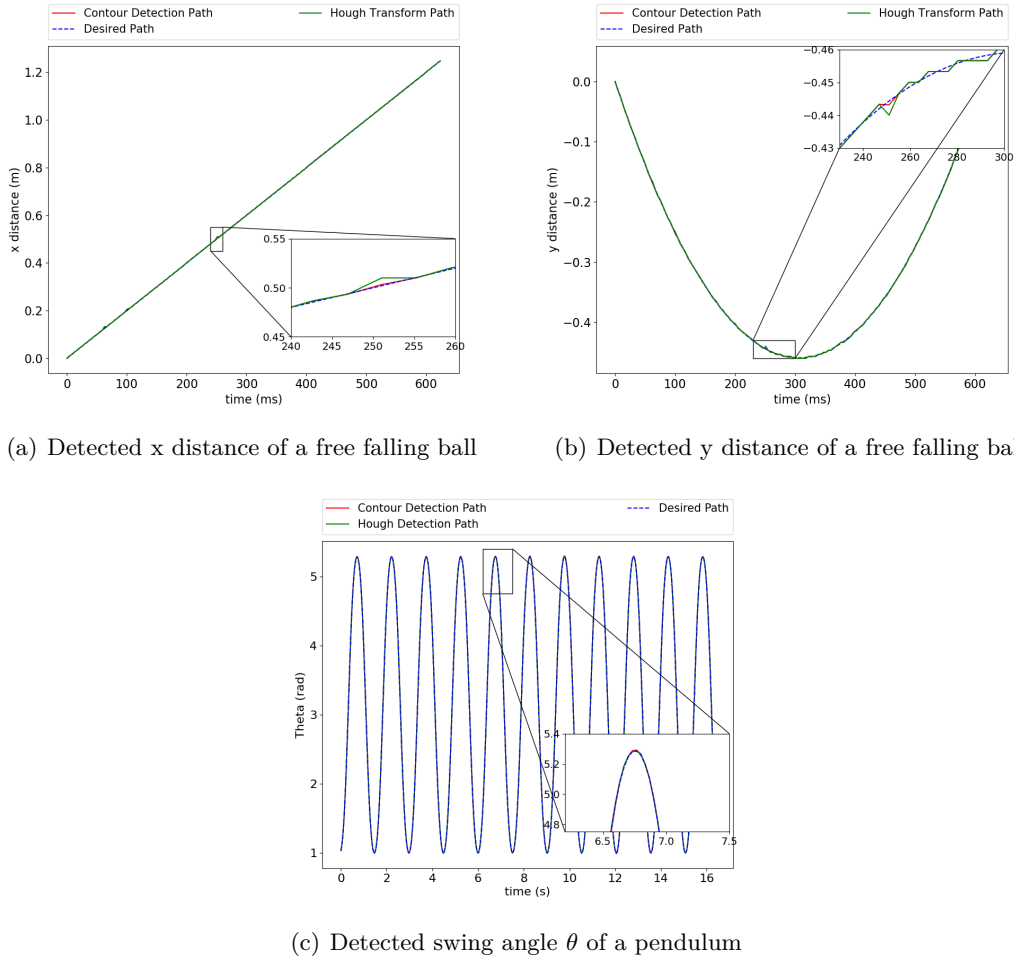


Figure 3-5: Detected position results of a simulated free falling ball video and a swinging pendulum compared with ideal values

Table 3-1: Mean square error of the detected results of two detection methods on two simulated videos generated from the motion of a free falling ball and a swinging pendulum.

Variable to be detected	Contour detection	Hough transform
Free falling ball - x distance	9.30e-07	1.96e-06
Free falling ball - y distance	9.82e-07	1.13e-06
Swinging pendulum - swing angle θ	5.15e-05	4.43e-05

3-3 Conclusion

Figure 3-6 gives a schematic diagram of the whole process of extracting the positional information from the original video. Firstly, the region of interest will be determined to eliminate the influence of useless background and remain the area of target object. Then a Canny algorithm will be applied to detect all the edges which would be used for contour detection. There are two methods to extract the physical feature from the edge pattern. If the shape of the target object is irregular, contour detection would be suitable because a minimum enclosing circle or rectangle would serve as an ideal alternative pattern. Under the case of noisy or discontinuous edges, Hough transform will be a solution to determine the target shape.

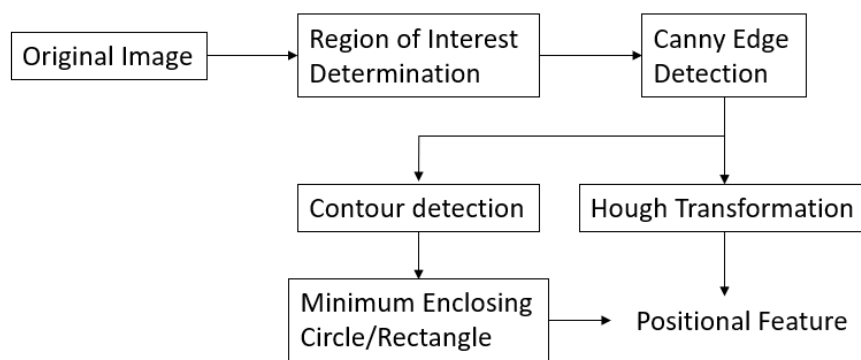


Figure 3-6: Schematic diagram of physical information extraction process

Physical Law Discovery

This chapter will give an introduction to the physical law discovery module which can learn the underlying governing equations from the extracted physical information. More specifically, we design the Mathematical Operation Network (MathONet), which has a neural network-like hierarchical structure as shown in Figure 4-2. Similar to multilayer perceptrons (MLP), MathONet is consist of of hidden layers and hidden neurons. The difference is that MathONet uses two substructures, Polynomial-Network (PolyNet) and Operational-Network (OperNet), to replace the weight scalar and activation function of MLP. The PolyNet and OperNet includes binary operations (e.g., \times , $+$) and unary operations (e.g., \sin , \log), respectively. An initialized MathONet is always composed of redundant operations and needs to be pruned to find the optimal subgraph to denote the governing equation.

In this section, traditional feedforward neural networks will be reviewed as the motivation. Then, the specific structure of MathONet will be explained. Finally, a simple example will be given to show how to correlate the mathematical expression and the sub-graph of MathONet after pruning.

4-1 Multilayer Perceptrons

As the prototype of MathONet, multilayer perceptrons (MLPs), or deep feedforward neural networks will be introduced in this part, as well as the training process of networks. On top of that, the origin of the loss function will also be discussed. They are the optimization theory foundation of DNN, which can be viewed as a kind of gradient-based learning.

Multilayer perceptrons are the quintessential deep learning models[20]. The purpose of a MLP is to approximate the optimal function $f^*(\cdot)$ that can fit the given dataset. Taking the classifier as an example, $y = f^*(\mathbf{x})$ defines a mapping relationship that categorizes the input \mathbf{x} into a class y . Generally, the MLP would try to learn a mapping $y = f(\mathbf{x}; \mathbf{w})$ through updating parameter matrix \mathbf{w} so as to achieve the best function approximation. The reason why these models are called feedforward is that all the information flows through intermediate functions from the input \mathbf{x} to the output \mathbf{y} without any feedback.

Usually, a MLP is consist of several linear layers and each layer is followed by an activation function to improve the model's fitting ability to nonlinear dataset. Figure 4-1 gives an example of an MLP with one hidden layer. With $x_1, x_2 \dots x_n$ being the input of the model, the output of the k_{th} neuron h_k from the linear layer is the weighted sum of all inputs. As shown in this figure, the k_{th} output of the first hidden layer can be represented as:

$$h_k^1 = \sum_{i=1}^n (w_{ik}^1 x_i + b_k^1) \quad (4-1)$$

where b_k^1 is the bias which is not shown in the figure for simplicity. Since the weighted sum operation can only result in outputs that are the linear combination of inputs, nonlinear activation functions like ReLU, tanh and sigmoid are necessary to improve the expressive ability of the network. The model in the figure uses ReLU as the activation function, which has the following expression:

$$\text{ReLU}(x) = \max(0, x) \quad (4-2)$$

After applying the activation function, the output a_k^1 can be represented as:

$$a_k^1 = \text{ReLU}(h_k^1) \quad (4-3)$$

In conclusion, as a linear model with one hidden layer, the optimization objective is to train $f(\mathbf{x}; \mathbf{w}, \mathbf{b}) = \text{ReLU}(\mathbf{x}^\top \mathbf{w} + \mathbf{b})$ to match the best estimation $f^*(\mathbf{x})$.

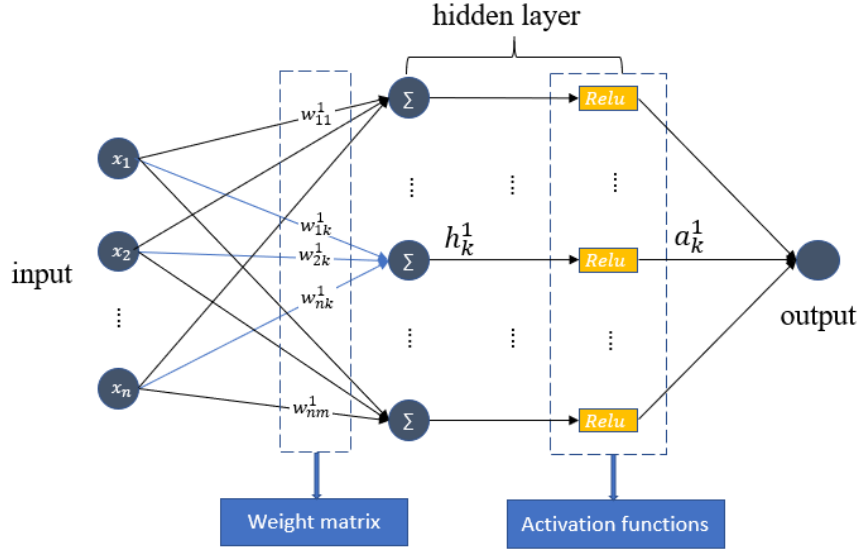


Figure 4-1: An example of a multilayer perceptron

The training process of a MLP can be summarized as three main steps. After initializing the weights of a model, the first step is to present a training sample for the model including the input and the corresponding label. Then the calculated result would be compared with the true label. Finally all the weights will be updated iterative by minimizing the loss function

until converging. To facilitate the following introduction, we can assume the loss function as the mean squared error $L(\mathbf{x}; \mathbf{y}; \mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (f(x_i) - y_i)^2$. Since the aim is to minimise the loss function, this can be realised by moving the inherent parameters \mathbf{w} in $f(\mathbf{x}, \mathbf{w})$ in the opposite sign of the gradient $\nabla_{\mathbf{w}} L(\mathbf{x}; \mathbf{y}; \mathbf{w})$. Thus the updated parameter can be represented by $\mathbf{w}' = \mathbf{w} - \epsilon \nabla_{\mathbf{w}} L(\mathbf{x}; \mathbf{y}; \mathbf{w})$, where ϵ is the learning rate. However, in the case of huge dataset, it is impractical to compute the gradient for all samples $\frac{1}{m} \sum_{i=1}^m L(x^{(i)}; y^{(i)}; \mathbf{w})$. As a solution, stochastic gradient descent (SGD) uses the gradient from a small number of samples m' as an approximation. Now, parameters can be updated taking use of the gradient of the mini-batch: $\mathbf{w}' = \mathbf{w} - \epsilon \frac{1}{m'} \sum_{i=1}^{m'} \nabla_{\mathbf{w}} L(x^{(i)}; y^{(i)}; \mathbf{w})$.

It is also worth mentioning how to choose the loss function from the perspective of probabilistic model [20]. Intuitively, a qualified loss function should provide suitable gradients that can help converge quickly when far away from the global optimal solution. In this case, maximum likelihood estimation can serve as the optimization objective. Given the dataset of m samples $\mathbb{X} = \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$, the true but unknown distribution can be represented as $p_{\text{data}}(x)$ under the assumption of independent and identical distribution. Our model can be viewed as a family of probability distributions $p_{\text{model}}(x; \mathbf{w})$ parameterized by \mathbf{w} .

$$\begin{aligned}
\mathbf{w}_{ML} &= \arg \max_{\mathbf{w}} p_{\text{model}}(\mathbb{X}; \mathbf{w}) \\
&= \arg \max_{\mathbf{w}} \prod_{i=1}^m p_{\text{model}}(x^{(i)}; \mathbf{w}) \\
&= \arg \max_{\mathbf{w}} \sum_{i=1}^m \log p_{\text{model}}(x^{(i)}; \mathbf{w}) \\
&= \arg \max_{\mathbf{w}} \mathbb{E}_{x \sim \hat{p}_{\text{data}}} [\log p_{\text{model}}(x; \mathbf{w})]
\end{aligned} \tag{4-4}$$

As shown in equation 4-4, the maximum likelihood estimation \mathbf{w}_{ML} represents the value of \mathbf{w} that maximizes the probability distribution in terms of all data samples. Due to that the product of small values between 0 and 1 would be very close to 0, log likelihood can transfer the multiplications into sums. Dividing by m does not change the maximum estimation result, so \mathbf{w}_{ML} can be rewritten as expectation with respect to the empirical distribution \hat{p}_{data} .

Maximum likelihood can be interpreted from the perspective of cross-entropy. Let H represent entropy and D be KL divergence which can measure the distance between two distributions. Because that the entropy of p_{data} is independent of the cross-entropy between p_{data} and p_{model} , the cross entropy is proportional to the KL divergence:

$$\begin{aligned}
H(p_{\text{data}}, p_{\text{model}}) &= H(p_{\text{data}}) + D(p_{\text{data}} || p_{\text{model}}) \\
&\propto D(p_{\text{data}} || p_{\text{model}})
\end{aligned} \tag{4-5}$$

The KL divergence term can be rewritten in terms of the expectation of data samples. Similar to the entropy term, $\log p_{\text{data}}(x)$ does not depend on the model, so it can be omitted:

$$\begin{aligned}
D(p_{\text{data}} || p_{\text{model}}) &= \sum_{i=1}^m \hat{p}_{\text{data}}(x^{(i)}) \log \left(\frac{p_{\text{data}}(x^{(i)})}{p_{\text{model}}(x^{(i)})} \right) \\
&= \mathbb{E}_{x \sim \hat{p}_{\text{data}}} [\log p_{\text{data}}(x) - \log p_{\text{model}}(x)] \\
&\propto \mathbb{E}_{x \sim \hat{p}_{\text{data}}} [-\log p_{\text{model}}(x)]
\end{aligned} \tag{4-6}$$

It can be observed that minimizing the cross-entropy is same with maximizing the likelihood:

$$\begin{aligned} \mathbf{w}^* &= \arg \max_{\mathbf{w}} \mathbb{E}_{x \sim \hat{p}_{\text{data}}} [\log p_{\text{model}}(x; \mathbf{w})] \\ &= \arg \min_{\mathbf{w}} -\mathbb{E}_{x \sim \hat{p}_{\text{data}}} [\log p_{\text{model}}(x; \mathbf{w})] \end{aligned} \quad (4-7)$$

Most of the modern neural networks are trained with maximum likelihood, while the specific forms of loss functions are determined by the selection of p_{model} . Actually, this is the advantage of maximum likelihood that it removes the burden of designing cost function for each model[20]. After calculating the loss through the feedforward network, the gradient would be propagated backwards for each layer to update parameters until convergence.

4-2 Model Design

The structure of MathONet is inspired by MLPs and it is a neural network-like hierarchical structure. As shown in figure 4-2, the original weight connection within the MLPs is replaced by a special linear layer called polynomial-network (PolyNet). And the original activation function within the MLPs is replaced by the operation network (OperNet) which is composed by several basic mathematical operations. PolyNets are shown as the blue blockers in the figure, which are independent trainable sub-networks. For instance, w_{nk}^1 is the weight to connect input x_n and output h_k^1 in the MLP. In MathONet, the output p_{nk}^1 of the PolyNet PN_{nk}^1 is equivalent to the traditional weight w_{nk}^1 while its value p_{nk}^1 is related to all input variables. It needs to be mentioned that besides the original input variables of the system (x_1, x_2, \dots, x_n), the input of the PolyNet PN_{nk}^1 contains an extra constant to represent the variable-invariant term in the governing equation. As an alternative of activation functions, OperNets are represented as the yellow blockers. Though the objective is also to induce nonlinearity, OperNets can improve the interpretability of the model for its intuitive expression form.

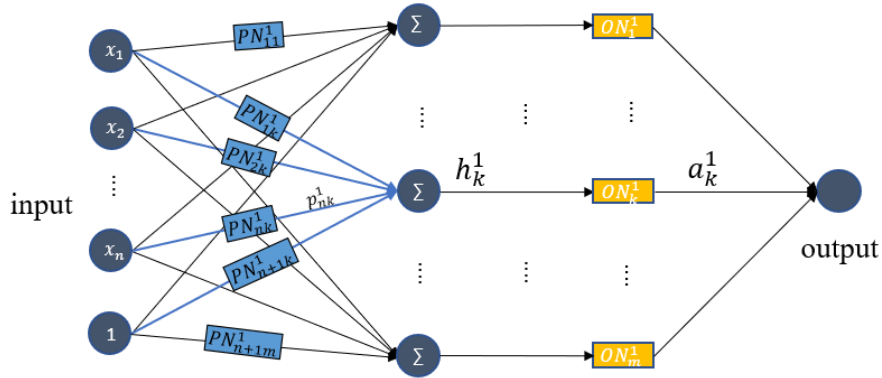


Figure 4-2: Mathematical operation network (MathONet)

4-2-1 Polynomial-Network

The structure of PolyNet is a simple Fully-Connected layer without any hidden layer or activation function. The input of each PolyNet is exactly the same as the whole model,

including the original system input variables and the constant term. Figure 4-3(a) illustrates PN_{nk}^1 , whose output p_{nk}^1 is multiplication result between the input x_n and the output h_k^1 of the k th neuron.

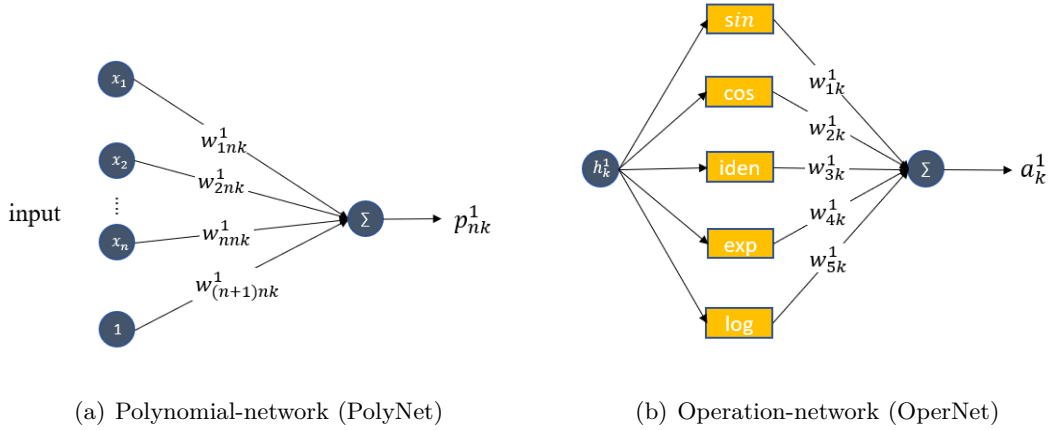


Figure 4-3: The structure of the Polynomial-network (PolyNet) and [Operation-network (OperNet)

Since h_k^1 is the weighted sum of all inputs:

$$h_k^1 = \sum_{i=1}^n (p_{ik}^1 x_i + b_k^1) + p_{(n+1)k}^1 \quad (4-8)$$

and each weight p_{ik}^1 is also the weighted sum of all inputs:

$$p_{ik}^1 = \sum_{j=1}^n (w_{ji}^1 x_j + b_{ik}^1) + w_{(n+1)ik}^1 \quad (4-9)$$

the final output of the model can be expressed as the polynomial of input variables, which embodies the addition and multiplication between any two input variables including the constant term. To put it another way, PolyNet can represent the binary operations in the governing equation of a dynamic system. On top of that, the introduced constant term can be used to represent the linear equation of one variable, as well as a constant term irrelevant to the variables. Obviously, one hidden layer can produce at most quadratic terms. If the physical equation to be identified has a higher order, the hierarchical structure is necessary to stack several hidden layers and more PolyNets.

4-2-2 Operation-Network

PolyNet can only provide the ability to fit the dataset whose governing equations are polynomials. Under this case, the subset of Operation-Network (OperNet) can serve as the activation function to find the polynomial boundary through distorting the feature space.

An example structure of OperNet is as shown in figure 4-3(b), it is a linear combination of several unary mathematical operations instead of one fixed activation function. Taking ON_k^1

which connects the hidden output h_k^1 and the output of this sub-network a_k^1 as an instance, a_k^1 can be represented as:

$$a_k^1 = ON_k^1(h_k^1) = \sum_{o=1}^{\mathcal{O}} w_{ok}^1 f_o(h_k^1) \quad (4-10)$$

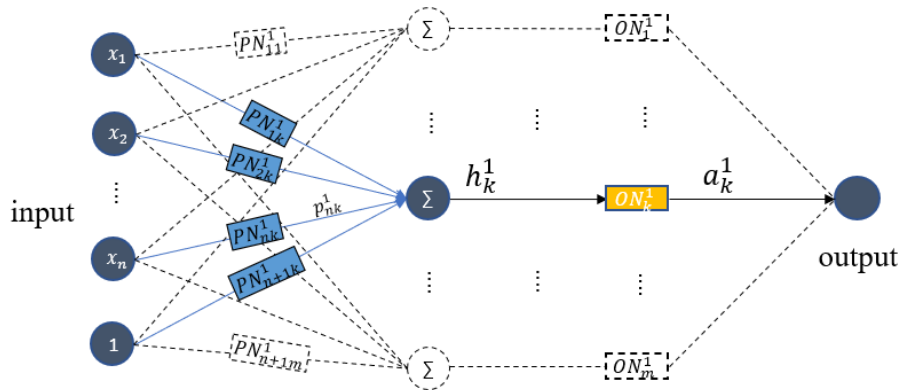
, where f_o denotes the unary functions, e.g., sin, cos, exp. An identity function is also necessary to remain the input in case that no nonlinear operation is required.

For a hierarchical structure, several PolyNets and OperNets are concatenated to approximate high-dimensional and complex functions. The output of the k_{th} neuron from the l_{th} activation function layer a_k^l is $\sum_{o=1}^{\mathcal{O}} w_{ok}^l f_o(h_k^l)$ where h_k^l is the output of the l_{th} hidden linear layer $h_k^l = \sum_{i=1}^n (p_{ik}^l a_i^{l-1} + b_k^l) + p_{(n+1)k}^l$.

4-2-3 Model Compression

From the introduction above, it can be concluded that MathONet includes all potential binary and unary operations that may make up of the accurate physical law. Under this case, the initialized MathONet is possible to be a redundant structure and the model compression method is necessary to prune useless connections.

Figure 4-4 illustrates how MathONet can be interpreted as a closed-form mathematical expression after model compression. Compared with figure 4-2, figure 4-4(a) uses dashed lines to represent unnecessary connections that will be pruned in the compression process. Specifically, in this example most of the hidden neurons are pruned and only k_{th} one is reserved together with its corresponding OperNet. Not only the main structure will be compressed, the two sub-networks, PolyNets and OperNets, will also be tailored to fit the linear and nonlinear properties of the governing function. Figure 4-4(b) gives a parse example of the PolyNet to calculate p_{nk}^1 : $p_{nk}^1 = w_{1nk}^1 x_1 + w_{nnk}^1 x_n$. When it comes to the OperNet, figure 4-4(c) shows an example. Since other hidden neurons have been pruned, the final output of the whole model equals to the output a_k^1 of PolyNet ON_k^1 , which can be represented as $w_{1k}^1 \sin h_k^1 + w_{3k}^1 h_k^1$. To put it another way, the compressed structure uses a sine function to reflect the nonlinear periodic characteristics and remains the identity channel to retain the linear expression.



(a) A pruned main structure

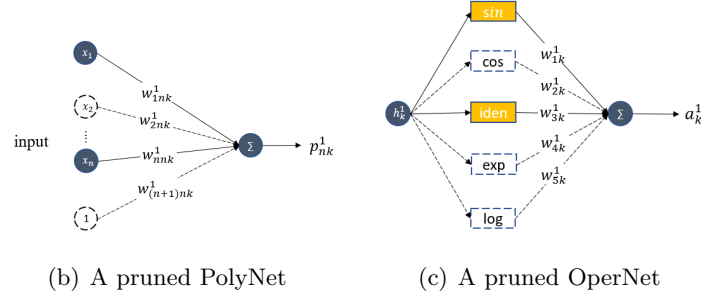


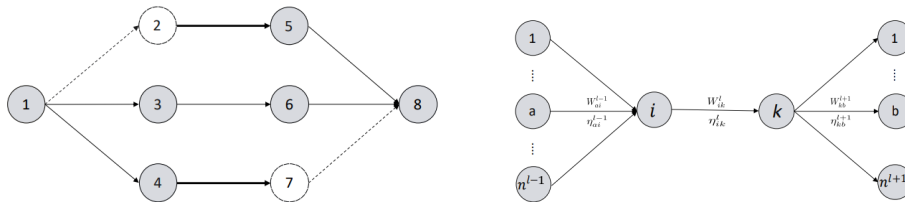
Figure 4-4: An example of MathONet after model compression

4-3 Dependency between connections

For neural network design and neural architecture search, it is often necessary to prune the redundant connections to address the overfitting problem[56, 32, 10]. Suppose a connection has two states {ON, OFF}, representing the reserved state and the redundant state, respectively. The common criterium of determining connection state is only based on its attributes, e.g. the magnitude of weight. However, this criterium ignores the dependencies between a connection and its predecessors and successors. It may result in a disconnected graph where some redundant structure cannot be identified.

As shown in Figure 4-5(a), we use a Directed Acyclic Graph (DAG) to represent a neural network. The number in the circle indicates the index of each neuron. The connection between neuron i and neuron k is denoted as e_{ik} . The colored edges (e_{25}, e_{47}) represent two types of disconnected connections. For e_{25} , no information is passed in it since the predecessor (e_{12}) has been deleted. For e_{47} , no information is passed out from it since the successor (e_{78}) has been deleted. Both edges should be pruned because neither of them performs the function of transmitting information within the network. However, their states may still be justified as ON if we mistreat the dependencies between connections. To address this problem, we first have the proposition 1:

Proposition 1. *The state {ON, OFF} of a connection e_{ik}^l as shown in Figure 4-5(b) is determined by whether it can transmit information within the neural network. Specifically, e_{ik}^l can be retained if and only if e_{ik}^l is ON, at least one predecessor connection of node i is ON and at least one successor connection of node k is also ON.*



(a) Two types of disconnected graph caused by mistreated of the dependencies. (b) A multi-in-multi-output motif to show the dependencies between connections

Figure 4-5: An illustration for the dependencies between connections.

Figure 4-5(b) shows a multi-in-multi-output motif for the connection e_{ik}^l which is used to encode the dependency. Assume s_{ik}^l represents the state of e_{ik}^l , proposition 1 can be encoded as following logic:

$$\bigcup_a s_{ai}^{l-1} \cap s_{ik}^l \cap \bigcup_b s_{kb}^{l+1} \quad \text{or} \quad \overline{\bigcup_a s_{ai}^{l-1} \cup s_{ik}^l \cup \bigcup_b s_{kb}^{l+1}} \quad (4-11)$$

where a is the neuron index for the predecessors with $a \in (1, \dots, n^{l-1})$. b is the neuron index for successors with $b \in (1, \dots, n^{l+1})$. The logic (4-11) can be expressed with two intuitive methods.

One method is to calculate the joint probability distribution of each connection which is affected by dependencies. To ease notation, W_{ik}^l is used to represent the weight parameter within the motif. We assume a Normal distribution over W_{ik}^l with $p(W_{ik}^l) = \mathcal{N}(W_{ik}^l|0, \eta_{ik}^l)$. η_{ik}^l is the variance which also stands for the uncertainty of W_{ik}^l . A larger η_{ik}^l means more confidence in the importance of W_{ik}^l . With the assumption that the initialized distribution of each connection is independent, the joint distribution over $W_{ik}^l, W_{ai}^{l-1}, W_{kb}^{l+1}$ can be expressed as:

$$\begin{aligned} & p(c(W_{ik}^l, W_{ai}^{l-1}, W_{kb}^{l+1})) \\ & \triangleq \mathcal{N}(W_{ik}^l|0, \eta_{ik}^l) \sum_{a=1}^{n^{l-1}} \mathcal{N}(W_{ai}^{l-1}|0, \eta_{ai}^{l-1}) \sum_{b=1}^{n^{l+1}} \mathcal{N}(W_{kb}^{l+1}|0, \eta_{kb}^{l+1}) \\ & = \mathcal{N}\left(W_{ik}^l \sum_{a=1}^{n^{l-1}} \frac{\eta_{ai}^{l-1} W_{ai}^{l-1}}{\sum_{a=1}^{n^{l-1}} \eta_{ai}^{l-1}} \sum_{b=1}^{n^{l+1}} \frac{\eta_{kb}^{l+1} W_{kb}^{l+1}}{\sum_{b=1}^{n^{l+1}} \eta_{kb}^{l+1}} \middle| 0, \bar{\eta}_{ik}^l\right) \end{aligned} \quad (4-12)$$

where

$$\bar{\eta}_{ik}^l \triangleq \left(\frac{1}{\sum_{a=1}^{n^{l-1}} \eta_{ai}^{l-1}} + \frac{1}{\sum_{b=1}^{n^{l+1}} \eta_{kb}^{l+1}} + \frac{1}{\eta_{ik}^l} \right)^{-1} \quad (4-13)$$

if κ_η is the threshold for uncertainty, the connection with $\bar{\eta}_{ik}^l$ smaller than κ_η will be determined as redundant.

The other method is to calculate the magnitude of weight in consideration of dependencies. The larger the magnitude of an edge, the more critical the edge is. Let's define κ_w as the weight threshold. As shown in the logic (4-11), the connection e_{ik}^l will be regarded as redundant if at least one of the following three conditions is satisfied: a) the magnitude of W_{ik}^l is smaller than κ_w ; b) the magnitude of all its predecessors W_{ai}^{l-1} is smaller than κ_w ; c) the magnitude of all its successors W_{kb}^{l+1} is smaller than κ_w . The redundancy of e_{ik}^l could be justified by the sign function $sgn(W_{ik}^l)$:

$$sgn(W_{ik}^l) = sgn(|W_{ik}^l|) \sum_{a=1}^{n^{l-1}} sgn(|W_{ai}^{l-1}|) \sum_{b=1}^{n^{l+1}} sgn(|W_{kb}^{l+1}|) \quad (4-14)$$

where $sgn(|x|) = \begin{cases} 0, & |x| \leq \kappa_w \\ 1, & |x| > \kappa_w \end{cases}$. By combining the uncertainty and magnitude of W_{ik}^l , s_{ik}^l could be determined by:

$$s_{ik}^l = \begin{cases} \text{OFF}, & \bar{\eta}_{ik}^l < \kappa_\eta \text{ or } sgn(W_{ik}^l) = 0 \\ \text{ON}, & \text{others} \end{cases} \quad (4-15)$$

4-4 Specific Example

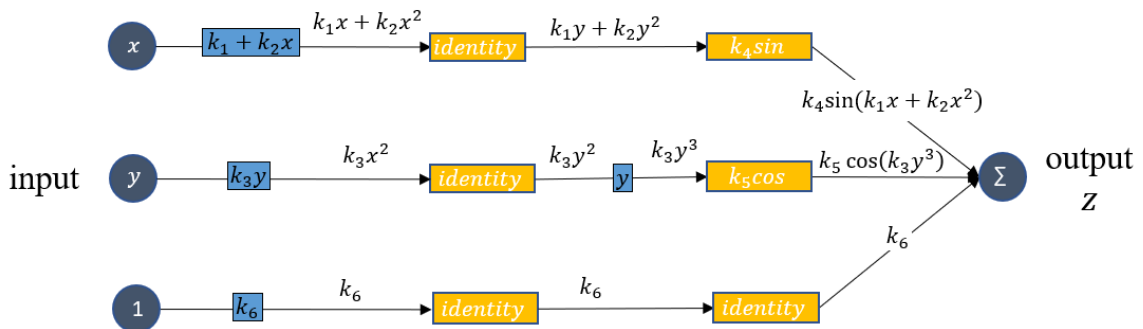
In this section, a specific example of a governing function will be given to explain how to identify the physical law. Considering the following equation:

$$z = k_4 \sin(k_1x + k_2x^2) + k_5 \cos(k_3y^3) + k_6 \tag{4-16}$$

This is a closed-form formula whose right-hand side has the highest order of three and contains linear and nonlinear operations (e.g., +, × and sin, cos). Since the aim is to represent the formula in a hierarchical expression that can be interpreted with MathONet, this formula should be decomposed into basic mathematical operations, as shown in figure 4-6(a). Figure 4-6(b) illustrates the consistent hierarchical representation of this decomposition procedure, which is equivalent to a compressed MathONet with two hidden layers. It can be noticed that this is not the only way to decompose and represent this specific formula. In the practical training process, all possible combinations of these operations have the probability to appear as the pruned model structure. However, this degree of freedom will not influence the simplest expression of final compressed model.

	Expression	Operation	Type
Decomposition ↓	$k_4 \sin(k_1x + k_2x^2) + k_5 \cos(k_3y^3) + k_6$	+	Binary
	$k_4 \sin(k_1x + k_2x^2), k_5 \cos(k_3y^3), k_6$	sin, cos	Unary
	$k_1x + k_2x^2, k_3y^3$	×	Binary
	$k_1x + k_2x^2, k_3y^2$	×, +	Binary
	k_2x, k_3y	×	Binary

(a) Decomposition procedure into mathematical operations



(b) Hierarchical representation of a formula

Figure 4-6: An example of a specific formula

4-5 Conclusion

By replacing the linear weights and activation functions with PolyNet and OperNet, the MathONet can construct the governing function of a dynamic system. This model has the following advantages:

- Compared with conventional neural networks which are black-box models, MathONet takes advantage of the sparse Bayesian algorithm to break the bottleneck of interpretability. In other words, this model reserves the necessary mathematical operations which make up of the governing function so that each weight is explainable.
- The MathONet has a considerable expression space even with a simple structure. For a MathONet with two input variables and only one hidden layer with one hidden neuron and two nonlinear operations, around 2^8 different mathematical expressions can be represented by model compression.
- Both linear and nonlinear dynamic system can be identified with MathONet. As shown in figure 4-7(a) and figure 4-7(b), a linear system $z = x + y$ and a nonlinear system $z = \sin zy + 1$ can be represented in the form of a hierarchical structure.

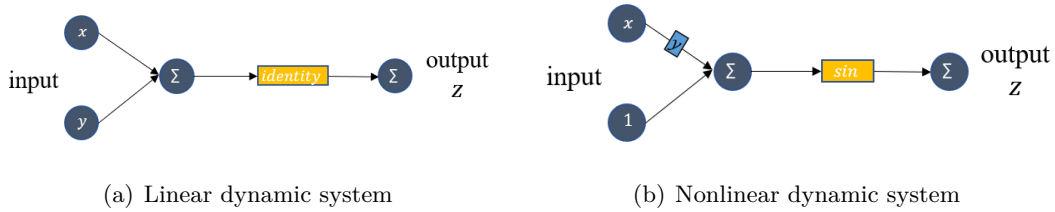


Figure 4-7: The identification examples of linear and nonlinear system

- Since the MathONet is inspired by conventional neural networks, the optimization method such as SGD can be utilized to train the model. At the same time, those commonly used deep learning frameworks (e.g., PyTorch, TensorFlow) to realise automatic differentiation and training acceleration can be implemented directly.

It should also be noted that the complexity of an initialized MathONet is determined by three hyperparameters that need to be defined before training the network:

- The number of hidden layers, which is similar to the number of layers in a traditional neural network. This value denotes the initialized model order and limits the depth of the MathONet.
- The number of hidden neurons in a layer, which also affects the complexity of network expression
- The number of unary functions in OperNet, which provides the ability to fit equations involving complex calculations.

It is intuitive that a more complex model has a stronger fitting ability and have a greater chance of containing the unknown path of correct answers. However, a complex model also requires a powerful discovery algorithm to avoid falling into a local optimal solution. This following chapter would introduce our optimization and compression algorithm based on Laplace approximation.

Sparse Bayesian Learning Algorithm

Since an initialized MathONet is overparameterized and has randomly specified weights, how to train the network and prune unnecessary weights to make sure that the remaining connections can represent the physical law will be a question of interest. A Bayesian learning framework was adopted for MathONet so that all weights in this network would be updated as posterior distributions instead of a simple point estimation. Given dataset \mathcal{D} , $W \in \mathbb{R}^{m \times 1}$ is used to represent the weight matrix in PolyNet or OperNet. To facilitate the following introduction, dependency between connections will not be considered in this chapter and it has been discussed in section 3-3. This chapter will explain how to estimate the posterior distribution $p(W | \mathcal{D})$ together with the principles of Bayesian compression.

5-1 Laplace Approximation

Given the dataset $\mathcal{D} = (X, Y) = \{(X_k, Y_k)\}_{k=1}^K$, assuming the targets Y are sampled from a model with additive zero-mean Gaussian noise σ^2 , then the training objective of our network is to fit the following equation:

$$Y_k = \text{Net}(X_k; W) + \epsilon_k \quad (5-1)$$

Thus, the likelihood can be defined in the form of a Gaussian distribution:

$$\begin{aligned} p(\mathcal{D} | W, \sigma^2) &= \prod_{k=1}^K \mathcal{N}(Y_k | \text{Net}(X_k, W), \sigma^2) \\ &= (2\pi\sigma^2)^{\frac{K}{2}} \exp\left(-\frac{1}{2\sigma^2} \sum_{k=1}^K (Y_k - \text{Net}(X_k, W))^2\right) \\ &= (2\pi\sigma^2)^{-\frac{K}{2}} \exp(-\mathbf{E}(W, \sigma^2)) \end{aligned} \quad (5-2)$$

Here $\mathbf{E}(W, \sigma^2)$ is denoted as the energy function, or the mean square error of the network given dataset \mathcal{D} . For further computation of the intractable integral of evidence, the energy

function can be expanded according to a second-order Taylor series expansion around a local optimal solution W^* :

$$\begin{aligned} E(W, \sigma^2) &\approx E(W^*, \sigma^2) + (W - W^*)^\top \mathbf{g}(W^*, \sigma^2) \\ &\quad + \frac{1}{2}(W - W^*)^\top \mathbf{H}(W^*, \sigma^2)(W - W^*) \end{aligned} \quad (5-3)$$

where $\mathbf{g}(\cdot) = \nabla \mathbf{E}(W, \sigma^2) |_{W^*}$ is the gradient of the energy function and $\mathbf{H}(\cdot) = \nabla \nabla \mathbf{E}(W, \sigma^2) |_{W^*}$ is the Hessian matrix. Then (5-2) would be rewritten as the following form:

$$\begin{aligned} p(\mathcal{D} | W, \sigma^2) &= (2\pi\sigma^2)^{\frac{K}{2}} \cdot \exp \left\{ -\frac{1}{2\sigma^2} \sum_{k=1}^K (Y_k - \text{Net}(X_k, W))^2 \right\} \\ &\approx (2\pi\sigma^2)^{\frac{K}{2}} \cdot \exp \left\{ -(E(W^*, \sigma^2) + (W - W^*)^\top \mathbf{g}(W^*, \sigma^2) + \frac{1}{2}(W - W^*)^\top \mathbf{H}(W^*, \sigma^2)(W - W^*)) \right\} \\ &= a(W^*, \sigma^2) \cdot \exp \left\{ -(W^\top \hat{\mathbf{g}}(W^*, \sigma^2) + \frac{1}{2}W^\top \mathbf{H}(W^*, \sigma^2)W) \right\} \end{aligned} \quad (5-4)$$

with

$$\begin{aligned} \hat{\mathbf{g}}(W^*, \sigma^2) &\triangleq \mathbf{g}(W^*, \sigma^2) - \mathbf{H}(W^*, \sigma^2)W^* \\ a(W^*, \sigma^2) &\triangleq (2\pi\sigma^2)^{\frac{K}{2}} \exp \left\{ -(E(W^*, \sigma^2) - W^{*\top} \mathbf{g}(W^*, \sigma^2) + \frac{1}{2}W^{*\top} \mathbf{H}(W^*, \sigma^2)W^*) \right\} \end{aligned}$$

A Gaussian prior distribution $p(W)$ is imposed on the weights for regularization:

$$p(W) = \mathcal{N}(W | \mathbf{0}, \Upsilon) = \prod_{i=1}^m \mathcal{N}(W_i | 0, \eta_i) \quad (5-5)$$

where $\Upsilon = \text{diag}[v]$ and $v \triangleq [\eta_1, \eta_2, \dots, \eta_m] \in \mathbb{R}_+^{m \times 1}$, which can be calculated through maximizing the evidence (evidence maximization is illustrated in section 4-2). According to the Bayesian theory, the posterior distribution is proportional to the multiplication of likelihood and prior:

$$p(W | \mathcal{D}, \sigma^2) \approx p(\mathcal{D} | W, \sigma^2)p(W) \quad (5-6)$$

Considering the Gaussian prior ((5-5)) and approximated likelihood ((5-4)), the posterior also belongs to a Gaussian distribution by the effect of the conjugacy rule:

$$p(W | \mathcal{D}, \sigma^2) = \mathcal{N}(W | \mu_W, \Sigma_W) \quad (5-7)$$

where

$$\mu_W = \Sigma_W \cdot \left[\mathbf{g}(W^*, \sigma^2) + \mathbf{H}(W^*, \sigma^2)W^* \right], \Sigma_W = \left[\mathbf{H}(W^*, \sigma^2) + \Upsilon^{-1} \right]^{-1}$$

5-2 Evidence Maximization

Considering that the given prior $p(W)$ is defined by hyperparameters, namely, the variances of zero-mean Gaussian distributions, it can be rewritten as $p(W | \Upsilon)$. The hyperparameter

$\Upsilon = \text{diag}[v]$ as well as the weight matrix W can be obtained through maximizing the marginal likelihood or evidence:

$$\begin{aligned}\hat{W}, \hat{v} &= \underset{v \geq 0, W}{\text{argmax}} \int p(\mathcal{D} | W, \sigma^2) p(W | v) dW \\ &= \underset{v \geq 0, W}{\text{argmax}} p(\mathcal{D})\end{aligned}\quad (5-8)$$

This problem is known as type II maximum likelihood or evidence maximization. By substituting the likelihood estimated by Laplace approximation ((5-4)) and the Gaussian distributed prior ((5-5)), the model evidence can be interpreted as the Gaussian posterior volume according to David MacKay's Bayesian framework [35]:

$$\begin{aligned}p(\mathcal{D}) &= \int p(Y | W, \sigma^2) p(W | \Upsilon) dW = \int p(Y | W, \sigma^2) \mathcal{N}(W | \mathbf{0}, \Upsilon) dW \\ &= \frac{a(W^*, \sigma^2)}{(2\pi)^{m/2} |\Upsilon|^{1/2}} \int \exp \left\{ \frac{1}{2} W^\top \mathbf{H}(W^*, \sigma^2) W + W^\top \hat{\mathbf{g}}(W^*, \sigma^2) + \frac{1}{2} W^\top \Upsilon^{-1} W \right\} dW\end{aligned}\quad (5-9)$$

The optimization objective is to maximize (5-9) through training hyperparameters v and weight matrix W . Although the direct integral operation is intractable and hard to derive, the standard result for a normalized Gaussian distribution can be utilized to simplify the integral calculation [5]:

$$p(\mathcal{D}) \propto \frac{a(W^*, \sigma^2)}{(2\pi)^{m/2} |\Upsilon|^{1/2}} \exp \left\{ \frac{1}{2} W^\top \mathbf{H}(W^*, \sigma^2) W + W^\top \hat{\mathbf{g}}(W^*, \sigma^2) + \frac{1}{2} W^\top \Upsilon^{-1} W \right\} |\Sigma_W|^{\frac{1}{2}}\quad (5-10)$$

It should be noted that W in (5-10) denotes all possible values which we want to obtain by optimization. Then \hat{W} could be acquired through maximizing the objective function as follows:

$$\underset{v \geq 0, W}{\text{argmax}} \frac{a(W^*, \sigma^2)}{(2\pi)^{m/2} |\Upsilon|^{1/2}} \exp \left\{ \frac{1}{2} W^\top \mathbf{H}(W^*, \sigma^2) W + W^\top \hat{\mathbf{g}}(W^*, \sigma^2) + \frac{1}{2} W^\top \Upsilon^{-1} W \right\} |\Sigma_W|^{\frac{1}{2}}\quad (5-11)$$

Applying a $-2 \log(\cdot)$ transformation to Eq. 5-11:

$$\begin{aligned}& \underset{v \geq 0, W}{\text{argmin}} -2 \log \left(\frac{a(W^*, \sigma^2)}{(2\pi)^{m/2} |\Upsilon|^{1/2}} \exp \left\{ \frac{1}{2} W^\top \mathbf{H}(W^*, \sigma^2) W + W^\top \hat{\mathbf{g}}(W^*, \sigma^2) + \frac{1}{2} W^\top \Upsilon^{-1} W \right\} |\Sigma_W|^{\frac{1}{2}} \right) \\ & \propto \underset{v \geq 0, W}{\text{argmin}} -2a(W^*, \sigma^2) + \log |\Upsilon| + \log |\Sigma_W| + \frac{1}{2} W^\top \mathbf{H}(W^*, \sigma^2) W + W^\top \hat{\mathbf{g}}(W^*, \sigma^2) + \frac{1}{2} W^\top \Upsilon^{-1} W \\ & \propto \underset{v \geq 0, W}{\text{argmin}} W^\top \mathbf{H}(W^*, \sigma^2) W + 2W^\top \hat{\mathbf{g}}(W^*, \sigma^2) + W^\top \Upsilon^{-1} W + \log |\Upsilon| + \log |\mathbf{H}(W^*, \sigma^2) + \Upsilon^{-1}|\end{aligned}\quad (5-12)$$

Therefore, evidence maximization is equivalent to minimizing the following objective function:

$$\mathcal{L}(W, \Upsilon) = W^\top \mathbf{H} W + 2W^\top (\mathbf{g} - \mathbf{H} W^*) + W^\top \Upsilon^{-1} W + \log |\Upsilon| + \log |\mathbf{H} + \Upsilon^{-1}|\quad (5-13)$$

5-3 Regularization Update Rules

As an optimization objective, (5-13) has two parameters W and Υ to be trained. This section will explain why the training objective is equivalent to including a regularizer on the model complexity and how the regularized loss function helps update parameters.

(5-13) is composed of two parts, $u(W, \Upsilon) = W^\top \mathbf{H}W + 2W^\top (\mathbf{g} - \mathbf{H}W^*) + W^\top \Upsilon^{-1}W$ which is convex jointly in W and Υ and $v(\Upsilon) = \log |\Upsilon| + \log |\mathbf{H} + \Upsilon^{-1}|$ which is concave in Υ . Then, the optimization problem can be solved as convex-concave procedure (CCCP). The proof is as follows.

For the convex part:

$$u(W, \Upsilon) = W^\top \mathbf{H}W + 2W^\top (\mathbf{g} - \mathbf{H}W^*) + W^\top \Upsilon^{-1}W \quad (5-14)$$

$u(W, \Upsilon)$ is a convex function as it is the sum of a monotonic function and two convex functions with the form of $f(X, Y) = X^\top Y^{-1}X$. As for the concave part, it can be rewritten as a log-determinant of an affine function of semidefinite matrix Υ :

$$\begin{aligned} v(\Upsilon) &= \log |\Upsilon| + \log |\mathbf{H} + \Upsilon^{-1}| \\ &= \log \left(|\Upsilon \|\Upsilon^{-1} + \mathbf{H} (W^*, \sigma^2)| \right) \\ &= \log \left| \begin{pmatrix} \mathbf{H} (W^*, \sigma^2) & \\ & -\Upsilon \end{pmatrix} \right| \\ &= \log |\Upsilon + \mathbf{H}^{-1} (W^*, \sigma^2)| + \log |\mathbf{H} (W^*, \sigma^2)| \end{aligned} \quad (5-15)$$

Till now, the optimization procedure of (5-13) can be solved with a CCCP. To be specific, Υ and W can be updated through computing the gradient of $v(\Upsilon)$ and $u(W, \Upsilon)$ respectively [55]:

$$\Upsilon = \arg \min_{\Upsilon \succeq 0} u(W, \Upsilon) + \nabla \Upsilon v(\Upsilon)^\top |_{\Upsilon=\Upsilon^*} \Upsilon \quad (5-16)$$

$$W = \arg \min_W u(W, \Upsilon) \quad (5-17)$$

According to the chain rule, the last term in the right-hand side of (5-16) can be analysed as:

$$\begin{aligned} \boldsymbol{\alpha} &\triangleq \nabla \Upsilon v(\Upsilon)^\top |_{\Upsilon=\Upsilon^*} = \nabla \Upsilon \left(\log |\Upsilon^{-1} + \mathbf{H} (W^*, \sigma^2)| + \log |\Upsilon| \right)^\top |_{\Upsilon=\Upsilon^*} \\ &= -\text{diag} \left\{ (\Upsilon^*)^{-1} \right\} \circ \text{diag} \left\{ \left((\Upsilon^*)^{-1} + \mathbf{H} (W^*, \sigma^2) \right)^{-1} \right\} \circ \text{diag} \left\{ (\Upsilon^*)^{-1} \right\} + \text{diag} \left\{ (\Upsilon^*)^{-1} \right\} \\ &= \text{diag} \left\{ \left[\begin{array}{ccc} \alpha_1 & \cdots & \alpha_m \end{array} \right] \right\} \end{aligned} \quad (5-18)$$

where Υ and \mathbf{H} are symmetric matrices and \circ represents the element-wise product. Thus Υ can be updated as:

$$\begin{aligned} \Upsilon &= \arg \min_{\Upsilon \succeq 0} u(W, \Upsilon) + \boldsymbol{\alpha} \Upsilon \\ &= \arg \min_{\Upsilon \succeq 0} W^\top \mathbf{H}W + 2W^\top (\mathbf{g} - \mathbf{H}W^*) + W^\top \Upsilon^{-1}W + \boldsymbol{\alpha} \Upsilon \\ &= \arg \min_{\Upsilon \succeq 0} W^\top \Upsilon^{-1}W + \boldsymbol{\alpha} \Upsilon \end{aligned} \quad (5-19)$$

Since $\frac{W_i^2}{\eta_i} + \alpha_i \eta_i \geq 2 |\sqrt{\alpha_i} \cdot W_i|$, for each i , the optimal η_i can be represented as:

$$\eta_i = \frac{|W_i|}{\sqrt{\alpha_i}} \quad (5-20)$$

It can be found that each η_i can be determined once W_i is given. Then, W_i can be obtained according to the following objective function, with the definition $\beta \triangleq \sqrt{\alpha}$:

$$\begin{aligned}
W &= \arg \min_W \frac{1}{2} W^\top \mathbf{H} W + W^\top (\mathbf{g} - \mathbf{H} W^*) + \frac{1}{2} \sum_{i=1}^m \|\beta_i \cdot W_i\|_{\ell_1} \\
&\propto \arg \min_W E(W^*, \sigma^2) + (W - W^*)^\top \mathbf{g} (W^*, \sigma^2) + \frac{1}{2} (W - W^*)^\top \mathbf{H} (W^*, \sigma^2) (W - W^*) \\
&\quad + \frac{1}{2} \sum_{i=1}^m \|\beta_i \cdot W_i\|_{\ell_1} \\
&\approx \arg \min_W E(W, \sigma^2) + \frac{1}{2} \sum_{i=1}^m \|\beta_i \cdot W_i\|_{\ell_1}
\end{aligned} \tag{5-21}$$

Till now, W can be calculated according to (5-21), and the hyperparameter Υ can be updated correspondingly according to (5-20). It is worth mentioning that W obtained at time t is just the W^* at time $t + 1$ for the reason that the updated W and Υ can be viewed as a temporary local optimum.

Considering the loss function (5-21), it includes a mean square error $E(W, \sigma^2)$ and an extra regularization term which is similar to a L1 regularization. However, the regularizer contains a coefficient β_i that can be trained for each element W_i . In the training process, $\Upsilon = \text{diag}[\eta_1, \eta_2, \dots, \eta_m]$ is adopted as the criteria to prune redundant connections and explore the optimal structure because each η_i denotes the uncertainty for the weight W_i . Since the prior for weight is defined as a zero-centered Gaussian distribution, a small variance η_i means a high confidence that the corresponding weight W_i has a high probability of being zero and does not contribute to the final result. Then, a binary mask matrix C can be generated with the same dimension of W and it will be optimized in the training stage. The value of C can be determined as:

$$C = \begin{cases} 0, & \eta_i < \kappa_\eta \quad \text{or} \quad |W_i| < \kappa_w \\ 1, & \text{others} \end{cases} \tag{5-22}$$

where κ_η stands for the threshold and each weight W_i with a variance smaller than the threshold would be pruned and no longer involved in the training process. At the same time, each weight W_i with absolute value smaller than the threshold κ_w would also be compressed. After the whole training and pruning process, a sparse structure would be remained to represent the explored physical law.

5-3-1 Conclusion

The pseudo code for illustrating the process of sparse Bayesian learning is as shown in algorithm 2. Two consistent terminologies (epoch and cycle) need to be defined as they are used in the training strategy. Same as conventional neural networks, one epoch means that each sample in the entire dataset has an opportunity to update the parameters in MathONet. The number of epochs will be represented as N_{epoch} . Regarding cycle, it includes N_{epoch} epochs, and the network pruning is performed at the last epoch of each cycle. Since the computation of Hessian matrix requires huge computation resource, the definition of cycle can help reduce the amount of calculations and speed up the training process. Obviously, the degree

of network training depends on N_{cycle} . A N_{cycle} that is too large will consume more time and lead to overfitting, while a N_{cycle} that is too small will result in underfitting. In our training process, an early stopping is implemented to limit overfitting. In addition, a tuning hyper-parameter λ is also introduced to control the degree of compression.

Algorithm 2 Bayesian learning discovery algorithm

Initialize: hyper-parameters $\beta, \eta = I$; threshold for pruning $\kappa_\eta, \kappa_w \in \mathbb{R}^+$; regularization tuning parameter $\lambda \in \mathbb{R}^+$; $N_{cycle} \in \mathbb{Z}^+$ denotes the maximum cycles; $N_{epoch} \in \mathbb{Z}^+$ denotes the number of epochs in each cycle.

for $i = 1$ to N_{cycle} **for** $j = 1$ to N_{epoch}

1. Update the weight W by applying the gradient decent with loss function as

$$\arg \min_W E(W, \sigma^2) + \lambda \sum_{i=1}^m \|\beta_i \cdot W_i\|_{\ell_1}$$

end for

2. Update v as Eq. 5-16.
3. Update mask C as Eq. 5-22.

end for

Experiment Result

To evaluate the performance of the proposed method, we implemented the whole video to physical law framework on videos of three dynamic systems, including a free-falling ball, a Duffing oscillator and a swinging pendulum. Synthetic videos were generated for three cases and real videos of a free-falling ball were recorded to test the feasibility of the framework for practical scenarios.

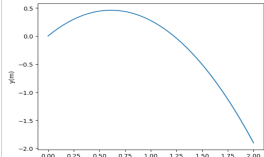
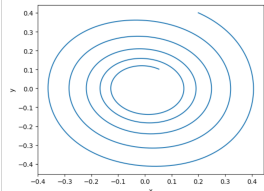
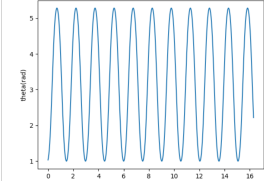
Dataset	Trajectory	Description
Free Falling Ball		This video consists of 150 frames with a frame rate of 240 frames per second. The frame size is 720x720 pixels. The scale of frames is 300 pixels per meter. A free-falling ball is released from a random starting point with random initial velocities. The object is acted upon by earth's gravity ($9.8m/s^2$), which is the only active external force.
Duffing Oscillator		This video has 600 frames with a frame rate of 20 frames per second. The synthetic data is generated with a time step of 0.01s for higher data accuracy. The scale is 720 pixels per unit. The Duffing oscillator is randomly initialized with conditions $x_0, y_0 \in [-0.8, 0.8]$. Duffing oscillator is a nonlinear system which will evolve itself without external forces.
Swinging Pendulum		This video consists of 500 frames with a frame rate of 30 frames per second. This video is generated based on the OpenAI gym pendulum environment. The object of interest is a swinging pendulum with randomly initialized position and angular velocity. With earth's gravity, the swinging angle of the pendulum would change correspondingly.

Figure 6-1: Description of the synthetic visual physical dataset

Figure 6-1 gives a table which overviews the trajectory and the basic properties of each synthetic visual dataset. For all simulated and real videos, the discovered results are closed-form

mathematical expressions with specific coefficients for each term in the expression, making it intuitive to compare the proposed approach with the ground truth. This chapter will introduce the experiments on three dynamic systems, including the analysis and the discovered results with the proposed method.

6-1 Free Falling Ball

As a canonical dynamic model, the free fall motion can be viewed as a common phenomenon in life, but it represents the core of classical physics theory. When the object is in the air, it is only affected by the gravity of the earth in the downward direction. Defining that to the right is the positive direction of the x-axis, and downward is the positive direction of the y-axis. Let g represents the acceleration of gravity, v_{x0} and v_{y0} stand for the initial velocity, x and y be the relative position from the initial point, the free fall equation can be described as:

$$\begin{aligned} x &= v_{x0}t \\ y &= v_{y0}t + \frac{1}{2}gt^2 \end{aligned} \tag{6-1}$$

This experiment was inspired by Newton. According to rumors, his idea of writing *Mathematical Principles of Nature Philosophy* [42] was inspired by seeing an apple drop. Similarly, our proposed framework should distill the physical law of motion (6-1) from a video of a free-falling ball. For this task, both synthetic dataset and real dataset are tested to discover the topology and parameters of a physical equation. The following section will introduce the generation process of the two datasets, together with how to extract the physical information from the videos. Then, the extracted data would be used to train and evaluate MathONet.

6-1-1 Synthetic Dataset

In this section, we generate a synthetic video with the dataset calculated from the standard mathematical formula ((6-1)) to simulated the process of a free-falling ball. Then, this synthetic video would be used to evaluate our end-to-end framework, including a comparison between our optimization algorithm with a widely used L1 regularization and no regularization technique.

Video Generation

The size of each frame in the video is 720x720 pixels. The time step to calculate the vertical and horizontal positions of the ball is $\frac{1}{240}s$, and the frame rate of the video is also 240 frames per second. 150 frames are included in a video. With a randomly initialized velocities v_{x0} and y_{y0} , and a random start position in the upper left area, the ball reaches the lower right corner of the video within 150 frames. Some screenshots from the video are shown in figure 6-2.

In order to show our experiments and comparison results more clearly, all the following experiments on this dataset are based on the initial velocity: $v_{x0} = 2m/s^2$, $v_{y0} = -3m/s^2$, with the right being the positive x axis and downward being the positive y axis. Earth's

gravitational acceleration g is determined as $9.8m/s^2$. The scale between real world and a synthetic video is 300 pixels per meter. It should be noted that the movement of objects in the real world is a continuous process, while a video is discrete. Thus, some inevitable derivations will be included in the video.

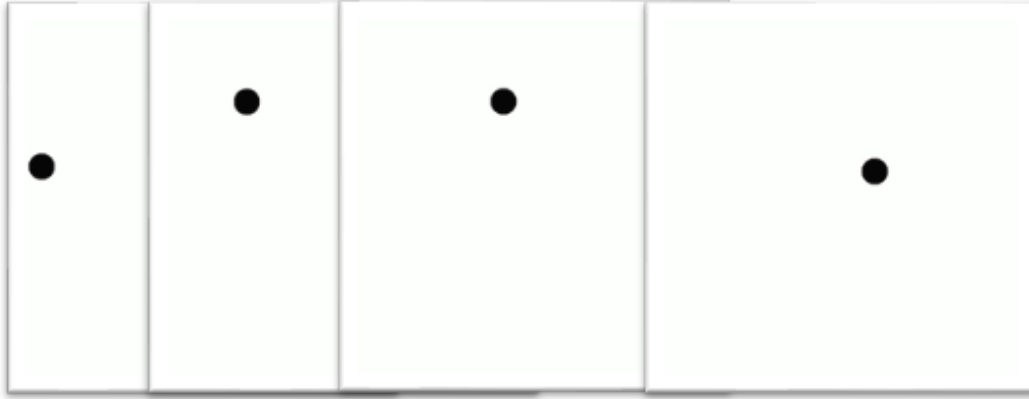
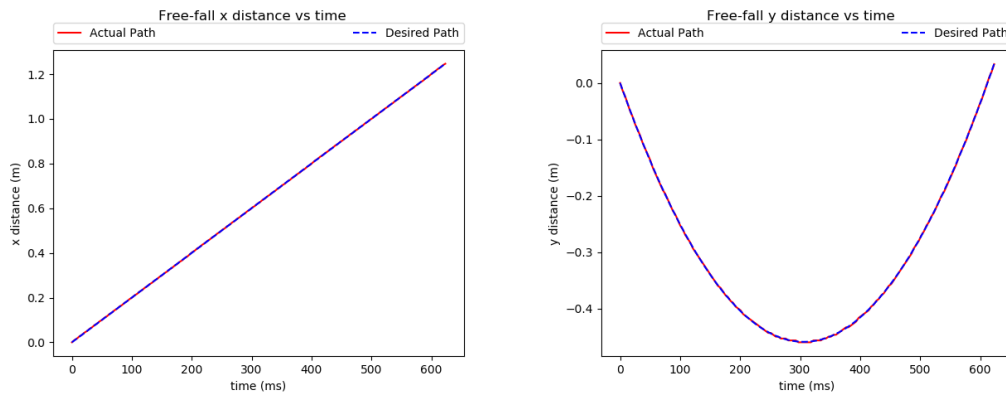


Figure 6-2: Visualization of the generated synthetic video frame of free falling fall

Physical Information Extraction

As illustrated in chapter 2, both contour detection and Hough transform perform well on the synthetic video. In this case, contour detection was chosen to detect the pixel coordinates of the ball.



(a) Detected x distance of a free falling ball

(b) Detected y distance of a free falling ball

Figure 6-3: Comparison between the extracted physical position and the true position

The only information that can be directly obtained from the video is the pixel coordinates and their corresponding timestamps, but this information has no physical meaning. What we want to discover is the relationship between the distance and time the ball moves in the horizontal and vertical directions. Therefore, a conversion between a pixel coordinate system and a physical coordinate system is necessary. The conversion relationship between the

physical coordinates and the picture coordinate system usually includes translation, scaling, and rotation. Because both the generated video and the real world are based on sea level, the rotation factor can be ignored. The distance that the ball moves is measured between the current position and the starting point, while the extracted pixel position from the video takes the upper left corner as the origin point. Then, the translation can be operated according to the position of the ball in the first frame. In regard to the scaling information, it can be obtained through the calibration of videos and real world. Since this is a synthetic video, the scale is known as 300 pixels per meter. So far, all physical information we care about has been extracted. Figure 6-3 shows the detected vertical and horizontal distance of the ball, compared with the true values generated from the mathematical formula.

Physical Law Discovery Experiment Setup

After the physical information of interest is extracted, the governing functions should be distilled from the MathONet. Because the governing equations describe the relationship between time and positions, the input variable for MathONet is the timestamp t corresponding to each frame in the video and the constant term. Two models will be trained separately to explore the two functions for vertical and horizontal distances. For the initialized MathONet, we tried several model hyperparameters (as illustrated in section 4-5) to test if the discovery algorithm can find the correct physical equations. The model structures we implemented are as shown in table A-2. To be specific, all experiments have 1 hidden layer and the number of neurons in the hidden layer varies from 1 to 3. In the OperNet, the alternative mathematical operations are also added from the simplest *ident* to other complex operation selections such as *exp* and *sin*.

Table 6-1: Different experiment setups for free fall dataset

Experiment setup	Number of hidden layers	Number of hidden neurons	Unary functions
Structure 1	1	1	<i>ident</i>
Structure 2	1	3	<i>ident</i>
Structure 3	1	3	<i>ident, exp, log</i>
Structure 4	1	3	<i>ident, exp, log, sin, cos</i>

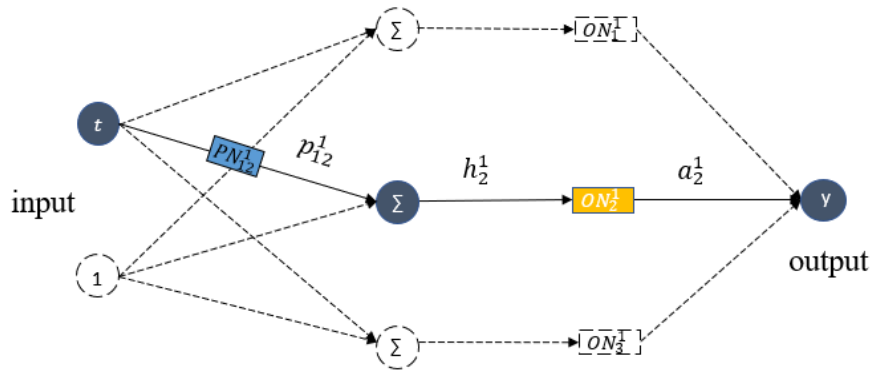
Normally, an initialized model with higher complexity will have a stronger ability to fit a more complex dataset. However, the over-redundancy may also lead to overfitting, which can be addressed by the network compression technique. In our experiment, we tuned the hyperparameter λ to control the degree of compression. It is chosen as the following four values: $1e-6$, $1e-4$, $1e-2$, $1e0$, $1e2$. Each value of λ is repeated for 10 times to prevent falling into a local optimal solution. In addition, we also implemented the conventional L1 regularization method using the same λ values and a traditional optimization with no regularization. For each model structure, the first 20 results with the smallest validation loss will be considered as the most probable solutions. In the case of similar losses, the simplest, that is, the most sparse network structure is viewed as the optimal solution.

Result Analysis

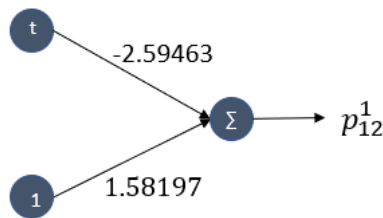
Since the video is generated with initial velocity: $v_{x0} = 2m/s^2$ and $v_{y0} = -3m/s^2$, the specific equation needs to be identified is:

$$\begin{aligned} x &= 2t \\ y &= -3t + 4.9t^2 \end{aligned} \tag{6-2}$$

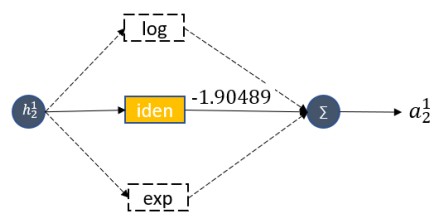
Because the second equation is more complex and contains the topology of the first formula, the following discussion is based on the discovery process of $y = -3t + 4.9t^2$. As illustrated before, we adopted different experiment settings (Table A-2). Impressively, both the topology and parameters of the correct model y can be identified correctly for all experiments except structure 4. Specifically, the successfully identified result for the most complex model (structure 3) is as followed:



(a) Identified MathONet



(b) Identified PolyNet



(c) Identified OperNet

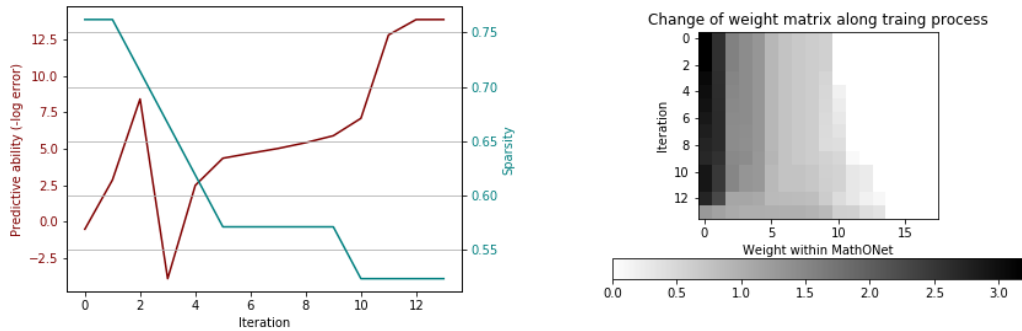
Figure 6-4: Identified governing equation for free fall dataset

The initialized MathONet has a hidden layer with three neurons, and there are 3 different mathematical operations (*identity*, *exp* and *log*) in the OperNet. At the same time, the input variables are timestamp t and the linear item. After the training stage finished, there is only one hidden neuron left and all connections related to the linear item are also pruned. As for the OperNet, only the identity operation is remained while other redundant mathematical

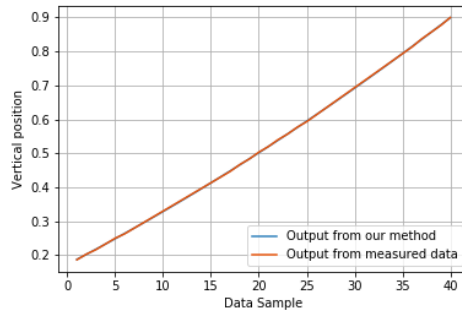
operations are compressed. The specific distilled equation from our model is :

$$\begin{aligned} y &= -1.90489 \times t (-2.59463 \times t + 1.58197) \\ &\approx -3.01347 \times t + 4.9425 \times t^2 \end{aligned} \quad (6-3)$$

which has the same topology as the correct solution while the parameters are approximated to the true values. It is also acceptable to not get very accurate model parameters, because this video-to-equation task is essentially unsupervised learning. To put it another way, the information provided to discover the physical formula is only a video, making the discovered result encoded with the physical information extraction error.



(a) Sparsity and predictive ability of MathONet in each cycle (b) The number of nonzero weights in the model



(c) Comparison between predicted output and the extracted information

Figure 6-5: The sparsity and predictive ability of MathONet for synthetics free fall dataset

Figure 6-5 visualizes the sparsity and predictive ability of MathONet generated in each cycle. The red line in Figure 6-5(a) demonstrates the changes of network predictive ability in the form of minus log error, while the green line indicates the sparsity of the compressed model. It can be concluded that after fluctuations, the network gradually fits the given data set and performs better on the validation set until convergence. At the same time, the network becomes more and more sparse at the end of each cycle, reaching a sparsity of about 50% at the end of the training process, where sparsity is defined as the remaining number of nonzero weight parameters over the number of all parameters in the initialized model. Figure 6-5(c) also proves the change of sparsity by visualising how the weight matrix changes in the

model along with the training process. The x axis denotes the number of nonzero weights in the model, while the y axis is the index of training cycles. The white area in the picture represents weights with values of 0. It is obvious that with the increasing of training cycles, the number of nonzero weights in the model is decreasing gradually, which means that the correct expression path is gradually explored. Figure 6-5(b) shows the comparison between the predicted output and the measured data extracted from the video, representing that the predicted trajectory basically coincides with the real trajectory.

In addition to our discovery algorithm, we also test MathONet of the same structure with other optimization methods to evaluate the rationality of Bayesian deep learning. A conventional L1 regularization is applied to the loss function to check whether the redundant connections can be compressed. The pruning threshold is $1e - 3$, which will be compared with the absolute value of weights. Besides, the conventional training without regularization is also applied to compare the results. Figure 6-6(a) and 6-6(b) illustrate the sparsity and predictive ability of MathONet at each cycle with L1 regularization and no regularization. Neither method compresses redundant connections in the network, so MathONet with both methods still keeps the sparsity of 100%. In another work, unnecessary mathematical operations such as *log* and *sin* are still kept in the network, making the discovered governing function lengthy and useless (the detailed result can be found in Appendix A-1). It can be concluded that the Bayesian learning algorithm has a better performance for relieving overfitting and makes more contribution to the physical law discovery network when the model is redundant.

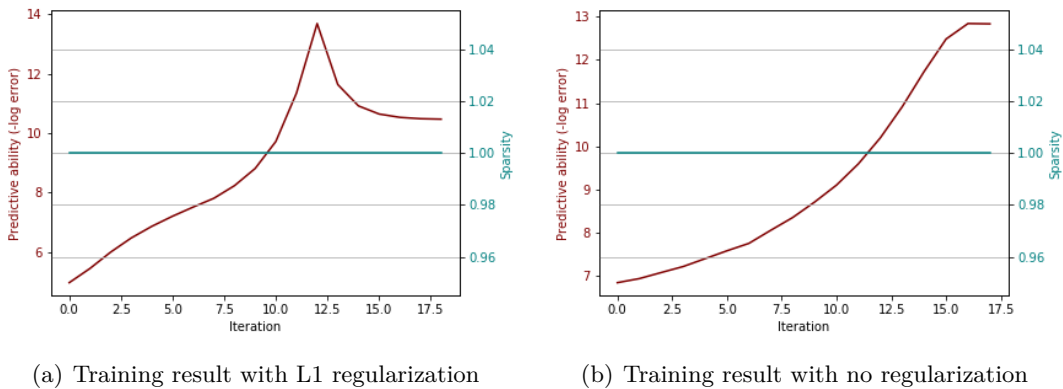


Figure 6-6: Sparsity and predictive ability of MathONet with different optimization methods

Optimization Method	Validation Loss
Bayesian Deep Learning	9.121822e-7
L1 Regularization	1.206884e-6
No Regularization	2.637105e-6

Table 6-2: Mean square error of the detected results of two detection methods

Table A-2 shows the validation loss of each method after the whole training stage finishes. MathONet optimized with Bayesian deep learning method has the smallest validation loss,

while both training methods with L1 regularization and no regularization have larger validation loss. Considering the fact that only the Bayesian algorithm successfully explored the correct governing equation and achieved the smallest validation loss, another conclusion could be drawn, that is, the smaller the validation loss, the higher the probability that the physical laws are accurately recognized.

6-1-2 Real Dataset

After evaluating our framework in idealization conditions with synthetic data, we also test the validity in practice. A real video of a free falling ball is recorded on a smart phone with a frame rate of 240 frames per second. Figure 6-7 shows some screenshots from the video, which indicates that the ball is thrown flat without an initial velocity in the vertical direction.

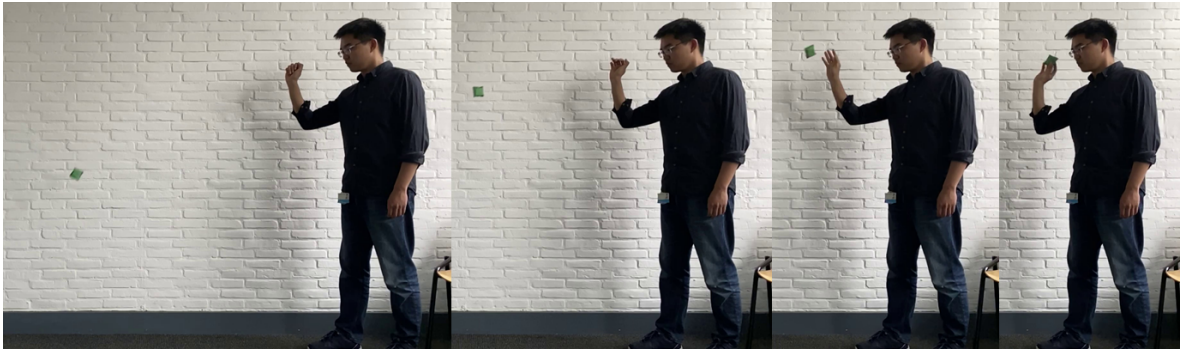
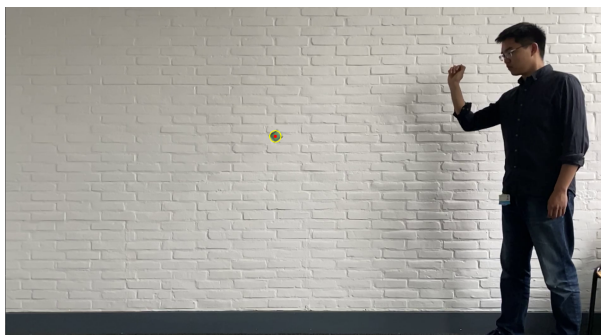
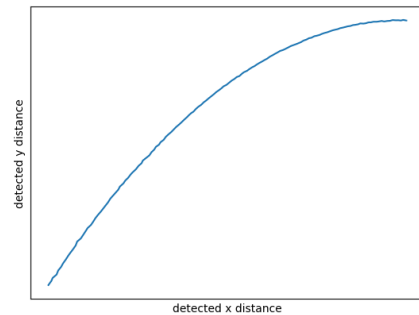


Figure 6-7: Visualization of the real dataset of free fall

Considering the fact that the thrown object is not a standard circle but has a color different from the background, contour detection is the best solution for physical information extraction in this task. Figure 6-8(a) shows the detected results with the yellow circle representing the minimum enclosing circle for the green object and the red dot being viewed as the centroid. The picture 6-8(b) gives the plot of the detected position for each frame, which is a unsmooth parabola encoded with detection noise..



(a) Contour detection result



(b) Detected x and y position

Figure 6-8: Detection result for the real free fall dataset

For a real video shot recorded by an ordinary mobile phone, there is no way to get the real initial velocity of the ball. For the same reason, we don't have an accurate equipment to calibrate the camera's external parameters which can provide the scale between an image and the real world. Therefore, the explored governing equation should have the same topology as the standard dynamic formula, but all parameters will be multiplied by an unknown zoom scale. Under this condition, all data used to train and evaluate our framework is the raw video. For specific experiment settings, the hyperparameters in Table A-2 are adopted. The result shows that the most complex model structure 3 (3 hidden neurons and 3 mathematical operations) can identify the equation successfully. The distilled equations are as followed:

$$\begin{aligned}x &= -3.095503 \times t \\y &= 5.029509 \times t^2 - 0.0026104\end{aligned}\tag{6-4}$$

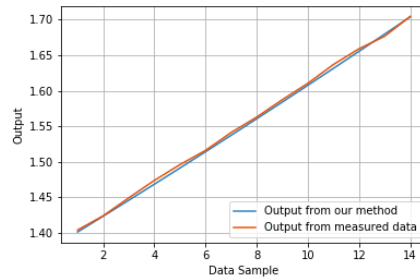
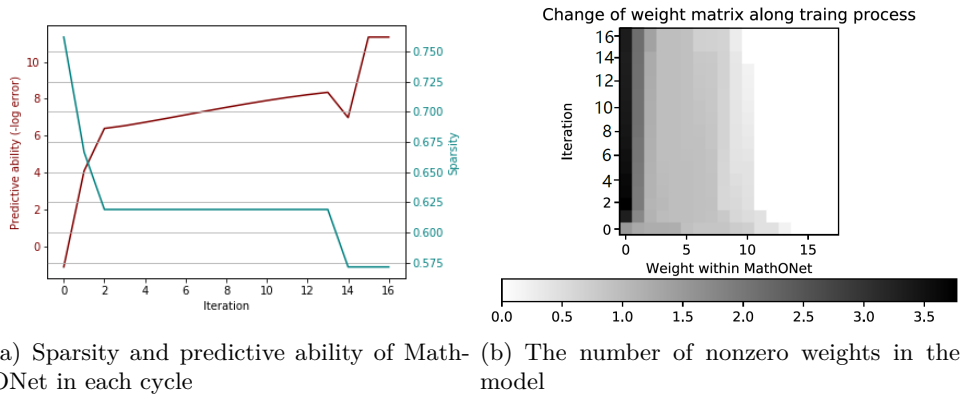


Figure 6-9: The sparsity and predictive ability of MathONet for real free fall dataset

Since the ball is thrown flat and has no initial speed in the y axis, the identified equation for vertical distance should be proportional to $\frac{1}{2}gt^2$. However, the second equation in Eq. 6-4 has a very small constant term. It may be caused by the noise from the physical extraction module. Another reason is that human throwing behavior does not occur strictly in a two-dimensional plane, which may lead to unpredictable input on the y axis. Even so, this result can prove the robustness of our model for being not disturbed by the noise. Figure 6-9(c) also provides support for this conclusion. The red line in the picture represents the trajectory of

the extracted data from the raw video, which is unsmooth and has small fluctuations. Instead of fitting the irregular noise encoded in the data, MathONet found a concise solution close to the real equation with the help of Bayesian compression.

6-2 Duffing Oscillator

Duffing oscillator is a non-linear second-order differential equation used to model certain damped and driven oscillators. The equation can be described as:

$$\ddot{x} + \delta\dot{x} + \alpha x + \beta x^3 = \gamma \cos(\omega t) \quad (6-5)$$

with δ controlling the amount of damping, α determining the linear stiffness, β controlling the amount of nonlinearity in the restoring force and γ being the amplitude of the periodic driving force. If $\gamma = 0$, the dynamic system has no external force and will evolve itself. If $\delta > 0$, Duffing oscillator will exhibit limit cycle vibration. If $\delta < 0$, the system will be chaotic and the phase diagram will have unpredictable attractors. For our work, the characteristics of Duffing oscillator are not worthy of attention. What should be discovered is that whether the governing function for the dynamic system can be discovered from a related video. Assuming the dynamic system has no driving force, $\delta = 0.1$, $\alpha = 1$ and $\beta = 3$, (6-5) can be rewritten as the form of state space expression:

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= -0.1y - x - 2x^3 \end{aligned} \quad (6-6)$$

Then, this system equation can be viewed as a first-order ordinary differential function with the candidate function polynomial order being 3. This is also the factor that distinguishes this system from the free fall system. Apart from the position, the first order information speed also needs to be extracted from the video. In addition, one hidden layer in MathONet can fit at most a second-order polynomial, which means that at least two hidden layers are enough for a third-order polynomial.

6-2-1 Synthetic Video Generation

Letting x and y denote the distance in the horizontal and vertical position in the Cartesian coordinate, \dot{x} and \dot{y} can represent the velocity in the x and y axes. With the given initial position, the Duffing oscillator can evolve itself without any external force. Then, a synthetic video can be generated to represent the evolution process accordingly. The initial position x_0 and y_0 is generated randomly from the range $[-0.8, 0.8]$. The following experiments are all based on the initial condition: $x_0 = 0.2$ and $y_0 = 0.4$. It should be noted that the values of x and y are not in meters, for that it should depend on the environment of the oscillator. When creating the synthetic video, the frame size is chosen as 720×720 pixels, and the scale is 720 pixels per unit.

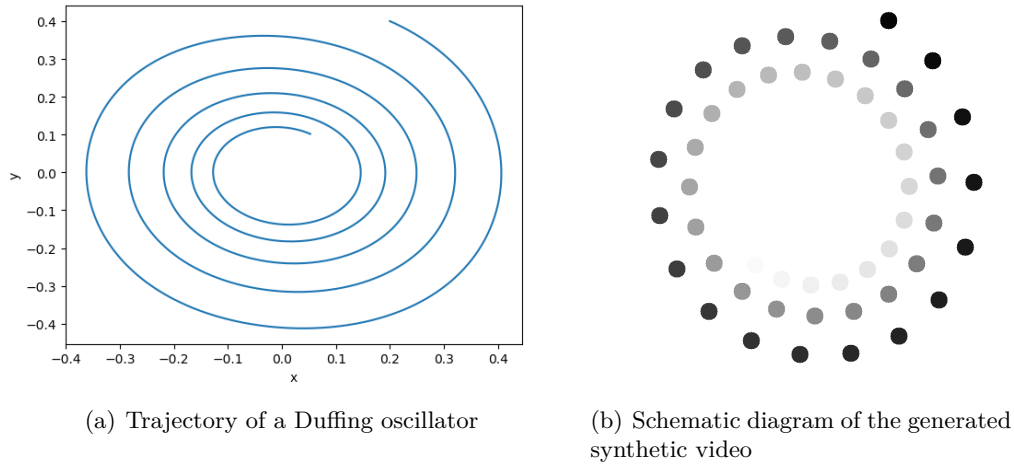


Figure 6-10: Trajectory and the simplified video diagram of a Duffing oscillator

There is no doubt that the smaller the time step, the more accurate the data generated. However, as shown in Figure 6-10(a), the trajectory of the state space of a Duffing oscillator will gradually converge to the original point. It may lead to that the the position detected in two adjacent frames will not change, because the pixel information must be integers. In this case, a time step of 0.01s is applied, while the frame rate is chosen as 20 frames per second. The generated synthetic video lasts from 30 seconds and includes 600 frames. Figure 6-10(b) illustrates how the synthetic video looks like. The black ball represents the current state of the oscillator and it moves following the trajectory to the left.

6-2-2 Physical Information Extraction

Given the raw video, how to extract the physical states and derive the first-order differential information is the topic for this section. Contour detection is utilized for determining the centroid of the ball. The next step is to take advantage of translation and scaling to transform the pixel coordinates to physical states. Same as the free fall dataset, the scale is known as 720 pixels per unit. However, the problem is that the coordinate origin is the upper left corner of the picture while the physical state origin cannot be distinguished directly from the trajectory. In this case, the initial state of the Duffing oscillator is given as a prior information. The distance of translation in the horizontal and vertical axis can be calculated through comparing the detected initial position of the object from the first frame and the given initial state of the dynamic system. Regarding the extraction of differential information \dot{x} and \dot{y} , they are solved by the first-order forward Euler method.

Figure 6-11 shows the error of the directly extracted y and the derived \dot{y} respectively. It can be observed that y is basically consistent with the real value, but the derived \dot{y} has some noise when approaching the local extremes. This phenomena also applies for x and \dot{x} . Table 6-3 shows the mean square error for the four variables. Obviously, \dot{x} and \dot{y} have larger error comparing with x and y .

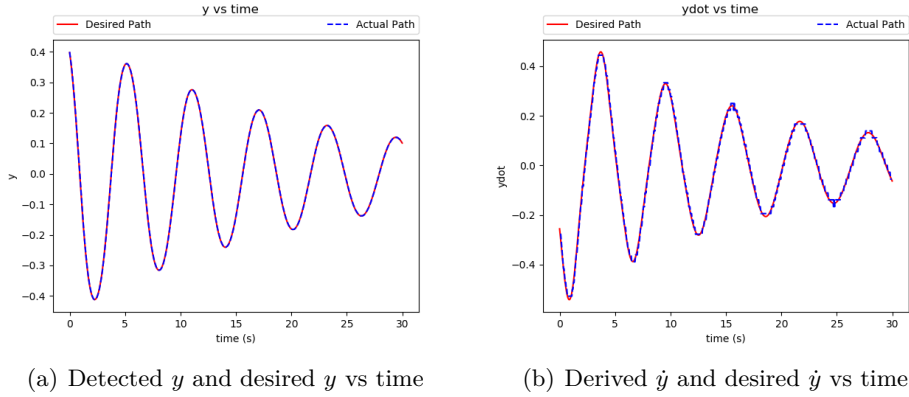


Figure 6-11: Error of extracted y and \dot{y}

Table 6-3: Mean square error of the detected results for the four variables

Mean Square Error	
x	3.747374e-6
y	4.466127e-6
\dot{x}	9.759904e-5
\dot{y}	9.711813e-5

6-2-3 Physical Law Discovery Experiment Setup

Since the equation to be discovered is a three-order polynomial, two hidden layers are necessary to fit the dataset. As shown in table 6-4, we tested two structures. Both have two neurons in the first and second hidden layer, while structure 1 has only one unary function (*ident*) and structure 2 has two (*ident* and *log*). According to the assumption that we know

Table 6-4: Different experiment setups for Duffing oscillator dataset

Experiment setup	Number of hidden layers	Number of hidden neurons	Unary functions
Structure 1	2	2, 2	<i>ident</i>
Structure 2	2	2, 2	<i>ident, log</i>

the variables that will contribute to the governing equation, the input for MathONet would be x , y and the linear item. For the two outputs, \dot{x} and \dot{y} are trained separately. The number of frames is 900 with 90% of the data for training and 10% for validation.

6-2-4 Result Analysis

The experiment result shows that only the distilled result from structure 1 is close to the

correct answer. The discover equations are:

$$\begin{aligned} \dot{x} &= 0.993245y \\ \dot{y} &= -1.914316x^3 + 0.199283x^2y + 0.073783xy^2 - 1.007207x - 0.070319y \end{aligned} \tag{6-7}$$

It needs to be noted that (6-7) are simplified polynomials since the result obtained directly from MathONet is lengthy and complex. In addition, we find that some items have small coefficients, making them not contribute to the final expression. Therefore, the terms of the polynomial with coefficients less than 0.05 were removed from the explored governing functions. Comparing (6-7) with (6-6), it can be found that the discovered result for the first equation has both correct topology and correct parameter. However, the second distilled equation has two redundant expression terms though it also contains the correct equation structure and approximately correct parameters. Figure 6-12(a) to 6-12(c) illustrate the sparsity and predictive ability of the trained MathONet for the first equation. Since this equation is relatively simple, the final pruned model has a sparsity of under 50%. In addition, since the calculated position for the moving objects must be rounded as integers to be plotted on a frame, some adjacent frames may have the object at the same position especially when the speed of the object is small. This will lead to jagged derived velocity \dot{x} and \dot{y} . As shown in figure 6-12(c) and 6-12(f), the orange lines represent the noisy extracted data from the video, while the blue lines indicate the fitting result of MathONet which is more smooth and closer to the true shape. Another interesting thing is that the sparsity of the model to fit the second governing equation does not drop a lot. That is because the initialized structure we used is simple and does not contain many unnecessary connections.

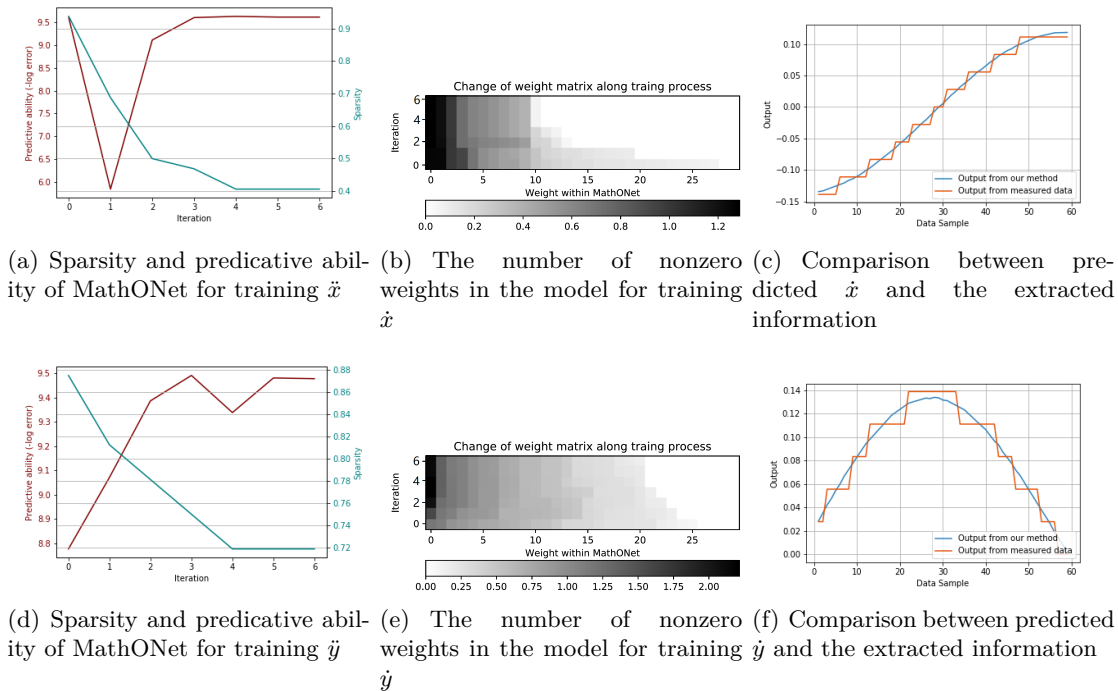


Figure 6-12: The sparsity and predictive ability of MathONet when training the video dataset of Duffing oscillator

As for the distilled function itself, we also designed two experiments to discuss whether it is reasonable and why it has extra expression terms. The first experiment is to explore the importance of each mathematical expression term in the distilled equation, while the second experiment is to use the noiseless true dataset to test the performance of our physical law discovery method.

The first experiment is implemented for exploring the importance of each term in the second equation of (6-7). The result is shown as in Table 6-5. For each term, we calculated the changed validation loss, which refers to the prediction loss on the validation dataset of the distilled equation after removing the corresponding expression term. Then, the importance is defined as the changed validation loss over the original validation loss. Obviously, a higher importance means that the corresponding expression term makes more contribution to match the given dataset. This table is sorted by the importance of each term. It can be found that all necessary terms have importance higher (all more than 15%) than the redundant expression terms (all smaller than 0.1%). Therefore, it can be concluded that the importance can act as the criterion for checking if the corresponding term makes contribution to the correct solution. On the other hand, this experiment also proves that although the distilled function is reasonable though it contains redundant terms.

Table 6-5: The importance analysis for different expression terms in the second distilled equation

Expression Item	Changed Validation Loss	Importance
$-1.007207x$	8.827e-3	11597.455%
$-0.070319y$	4.189e-5	55.039%
$-1.914316x^3$	1.214e-5	15.953%
$0.199283x^2y$	3.09e-8	0.041%
$0.073783xy^2$	2.80e-8	0.037%

¹ Changed validation loss indicates the prediction accuracy of the retained expression, which is obtained by removing the corresponding expression term.

² Importance is calculated by the changed validation loss over original validation loss

In addition, a possible factor to raise redundant expression terms in the distilled equation may be the quality of the extracted data. As introduced in the synthetic video generation process, determining the pixel coordinate from physical states needs to round the decimals, which may cause the loss of some information. Therefore, we designed the second experiment to test our method on the noiseless dataset. With using the true dataset for generating the synthetic video to train MathONet, the distilled functions are:

$$\begin{aligned} \dot{x} &= 1.000y \\ \dot{y} &= -1.998x^3 - 1.000x - 0.099y \end{aligned} \quad (6-8)$$

(6-8) is nearly same as the correct solution. Comparing the discovered result from the video dataset and the true dataset, we can get the conclusion that the accuracy of the discovered result depends on the quality of the training dataset.

Table 6-6 compares the validation loss and the number of cycles needed to finish the training process for different optimization methods, e.g., Bayesian approach, L1 regularization and no regularization. For the video dataset, all these three optimization methods can have similar

performance that can identify mathematical expressions with almost the same loss . At the same time, for the true dataset, the Bayesian approach has obvious advantages with a smaller validation loss, but it requires more training cycles to converge. It is also worth mentioning that although L1 and no regularization have more validation loss, both of them can identify the governing functions with correct topology and nearly accurate parameters. Details of the identified results of models trained with L1 and no regularization can be found in Appendix A-2. In general, a Bayesian approach can always achieve training results that are not worse than L1. In addition, all models trained with L1 and no regularization in Table 6-6 are not sparse. In other words, no connection is removed from the model. However, this cannot be used as evidence of the bad performance of L1 and no regularization, because the initialized model has no redundant unary operation. Another conclusion can be achieved that under the case of no redundant connections, Bayesian approach, L1 and no regularization can have similar performance because the ability to fit the data set is mainly provided by the model.

Table 6-6: The validation loss and the number of training cycles for distilled equations from two dataset with different optimization methods

	Bayesian optimization	L1 regularization	No regularization
First distilled equation for video dataset	6.631e-5 & 5	6.767e-5 & 5	6.756e-5 & 4
Second distilled equation for video dataset	7.551e-5 & 6	7.819e-5 & 5	7.673e-5 & 9
First distilled equation for true dataset	1.603e-17 & 8	3.022e-10 & 3	8.954e-10 & 3
Second distilled equation for true dataset	1.782e-12 & 6	4.288e-9 & 4	1.169e-7 & 4

6-3 Swinging Pendulum

As an instrument that can display many mechanical phenomena, the pendulum has been studied for many years, and it can even be used as scientific tools such as the accelerometer and seismometer. The most obvious property of a swinging pendulum is its periodic oscillation. Using the governing function of a swinging pendulum as an example to test, our experiments on a synthetic video of pendulum can help demonstrate the ability of our model to fit periodic functions. In addition, we also implemented our method on a real-recorded video of a pendulum, details are in Appendix B.

6-3-1 Synthetic Video Generation

The synthetic video of a pendulum is generated based on the environment from OpenAI gym, which provides a powerful physical engine to simulate object motion. The schematic diagram of the simulated video of a pendulum is as shown in Figure 6-13. The pendulum will swing around the black frictionless pivot, while the upward direction is recorded as the origin of the angle θ . The length of the pendulum is l and the mass is m . Initialized at a random position, the pendulum will swing because of the gravity mg .

(a) Schematic diagram of the video \ddot{x}

(b) Properties of the pendulum

Figure 6-13: Synthetic video of a swinging pendulum

Then, the governing equation of the system should describe the swinging process, which is related to the angle θ , angular velocity $\dot{\theta}$ and angular acceleration $\ddot{\theta}$. According to the equation that the angular acceleration $\ddot{\theta}$ equals to the torque M over the moment of inertia I . Since the torque is raised by the gravity, it can be represented as $M = \frac{l}{2}mg \sin \theta$. As for the moment of inertia of a rod about end, it can be expressed as $I = \frac{1}{3}ml^2$. Then, the governing equation can be written as:

$$\delta\dot{\theta}(t+1) = \ddot{\theta}(t+1) \cdot dt = \frac{3g}{2l} \sin(\theta(t)) \cdot dt \quad (6-9)$$

With $g = 10$, $l = 1$, and the time step $dt = 0.05s$, the specific equation needs to be identified as:

$$\delta\dot{\theta}(t+1) = 0.05\ddot{\theta}(t+1) = 0.75 \sin(\theta(t)) \quad (6-10)$$

To be consistent with the time step, the frame rate of the video is 20 frames per second. 500 frames are included in the synthetic video with 90% for training and 10% for validation. The initial swinging angle is set as 1 *rad* and the initial angular velocity is 0 *rad/s*.

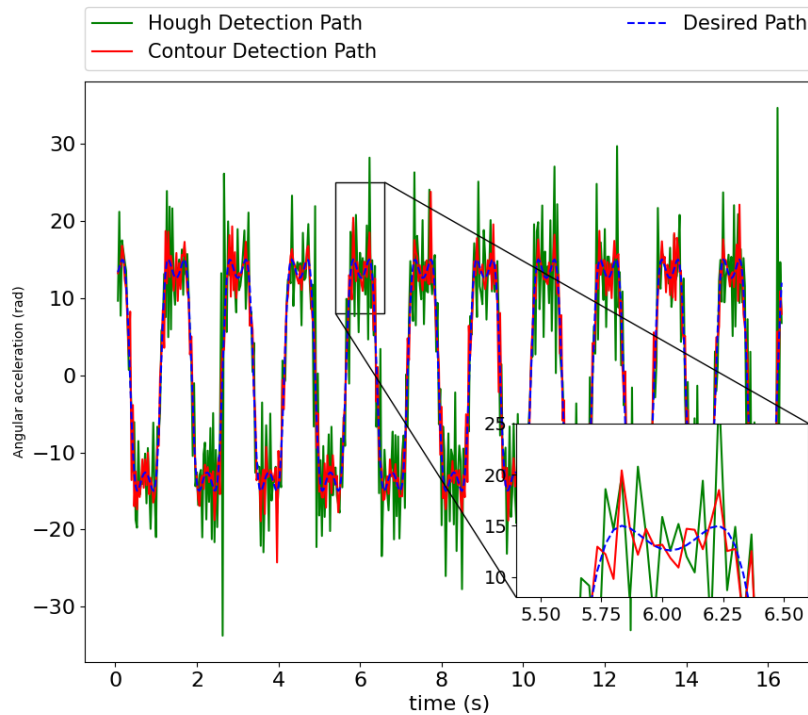
6-3-2 Physical Information Extraction

According to our assumption that we know the variables of the physical law to be explored, the swinging angle θ and the angular acceleration $\ddot{\theta}$ of the pendulum needs to be detected from the video. However, the angular acceleration should be derived from the angle velocity $\dot{\theta}$, making it necessary to detect $\dot{\theta}$.

Both contour detection and Hough transform are implemented for extracting the physical information to compare the accuracy. Hough transform can extract the slope of the pendulum directly, while the angular velocity needs to be derived with a first-order Euler method. As for the contour detection, the pixel position of the black pivot can be detected firstly as the

Table 6-7: Mean square error of the detected results for the three variables with two detection methods

Extracted variables	Contour Detection	Hough Transform
Swinging angle θ	5.146e-05	4.434e-05
Angular velocity $\dot{\theta}$	0.007	0.031
Angular acceleration $\ddot{\theta}$	7.044	36.294

**Figure 6-14:** Comparison of the derived angular acceleration $\ddot{\theta}$ between two methods

original point. Then, the vertical and horizontal position of the pendulum centroid for each frame in the video can be acquired by determining a minimal enclosing rectangle. After that, the slope of the pendulum can be calculated by the two pixel positions. Similarly, the angular velocity can be achieved through calculating the positional change of two adjacent frames. In other words, Hough transform can extract the angle θ from the video directly, while contour detection is able to extract the positional information θ together with the first-order information velocity $\dot{\theta}$ from the frames. As for the angular acceleration, both methods cannot extract it directly from the video, so Euler-method is implemented. Table 6-7 compared the accuracy of the extracted variables with two methods. It can be found that both methods perform similarly on the angle detection. However, contour detection has a obvious advantage when detecting the angular velocity, and mean square error of the angular acceleration detected with contour detection is also much smaller. As a picture to illustrate the difference of the extracted angular acceleration $\ddot{\theta}$ with two methods, Figure 6-14 also proves that both methods will produce large noise but contour detection has a better

performance in general. In the following experiments, the dataset for training is generated from contour detection.

Figure 6-15 shows the process of detecting the centroid position of a pendulum from a given frame. Figure 6-15(b) is the mask indicating the area of interest, which is used for edge detection and contour detection. The detected result is as shown by the blue contour in Figure 6-15(c), while the blue point is the center of the minimal enclosing rectangle, which indicates the pendulum centroid.

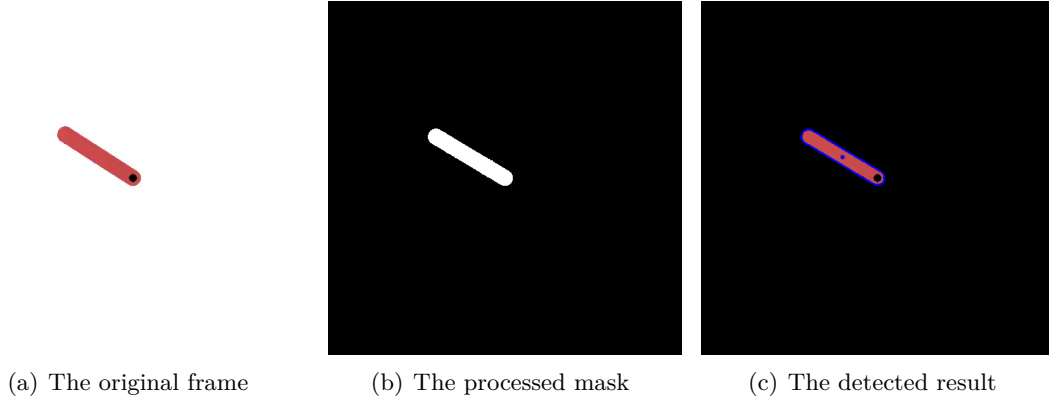


Figure 6-15: The detection process of the positional information of the pendulum with contour detection

6-3-3 Experiment Setup

According to the governing function, the angle of the pendulum at the current state is the input variables of our model, while the angular velocity at the next state is the output. However, the work [57] has proven that neural networks are always hard to learn periodic functions for that DNNs with periodic functions as activation functions are easy to be trapped in the local optima. We also implemented some experiments with one of activation functions set as \sin , the result shows that our model not always can discover the correct function. More specific details can be viewed in Appendix A-3. As an alternative solution, encoding the periodic characteristics of the pendulum system as the prior information in the input variables can help us discover the governing function successfully. Then, the input variables of MathONet are θ , $\sin(\theta)$ and the constant term. Therefore, this problem has been transferred into discovering a linear function from discovering a periodic function. The model structures we tested are as follows:

Table 6-8: Different experiment setups for swinging pendulum dataset

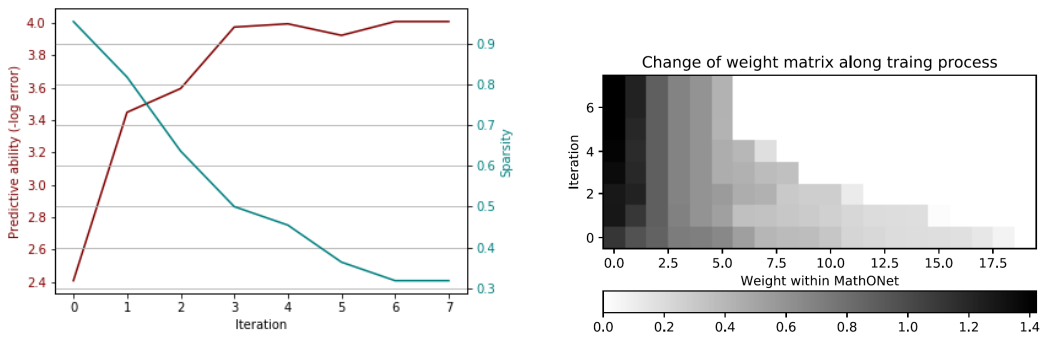
Experiment setup	Number of hidden layers	Number of hidden neurons	Unary functions
Structure 1	1	2	<i>ident</i>
Structure 2	1	2	<i>ident, log, exp</i>

6-3-4 Result Analysis

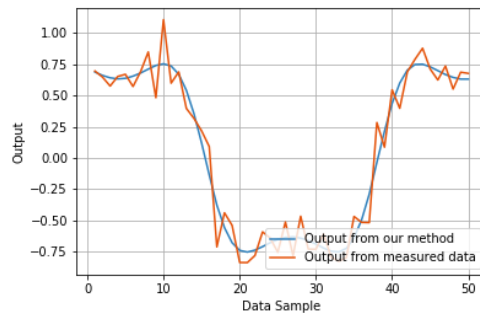
Both experiments successfully identify the governing equations although the extracted angular acceleration $\ddot{\theta}$ has much noise. For structure 2, the distilled function is:

$$\ddot{\theta}(t+1) \cdot dt = 0.752 \sin(\theta(t)) \quad (6-11)$$

Clearly, the distilled result has the same structure and similar parameters to the correct solution. The sparsity and predictive ability of our model in each cycle is as shown in Figure 6-16(a). Since the initialized MathONet is over-parameterized and contains redundant unary operations (e.g., log and exp), the sparsity of MathONet decreases obviously to almost 30%, which can also be reflected on Figure 6-16(b). At the same time, the predictive ability of our model increases steadily with the training of the network. Figure 6-16(c) compares the predicted output of our model and the extracted data, where the blue smooth line represents the model inference result and the noisy orange line denotes the extracted information from the video. Although MathONet is trained on the noisy dataset, our Bayesian learning method can compress all redundant connections and surpass the overfitting completely.



(a) Sparsity and predictive ability of the model in each cycle (b) The number of non-zero weights in the model in each cycle



(c) Comparison between predicted result and extracted data

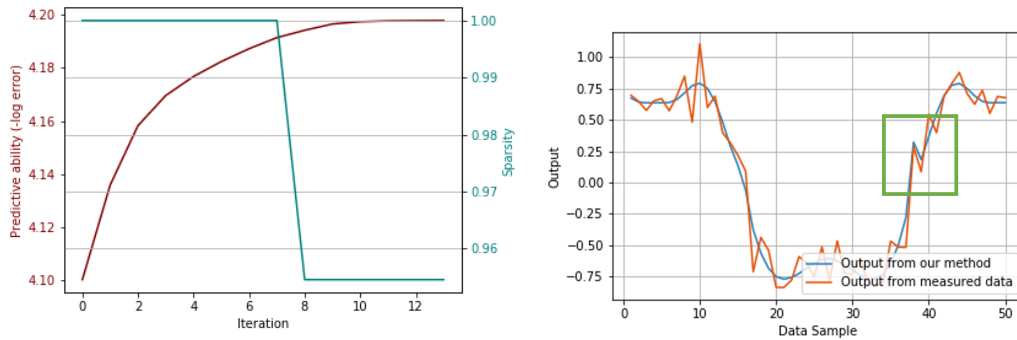
Figure 6-16: Sparsity and predictive ability of MathONet for the swinging pendulum dataset

To evaluate the performance of our discovery algorithm, we also implemented L1 and no regularization for training MathONet with the same structure. The validation loss of the

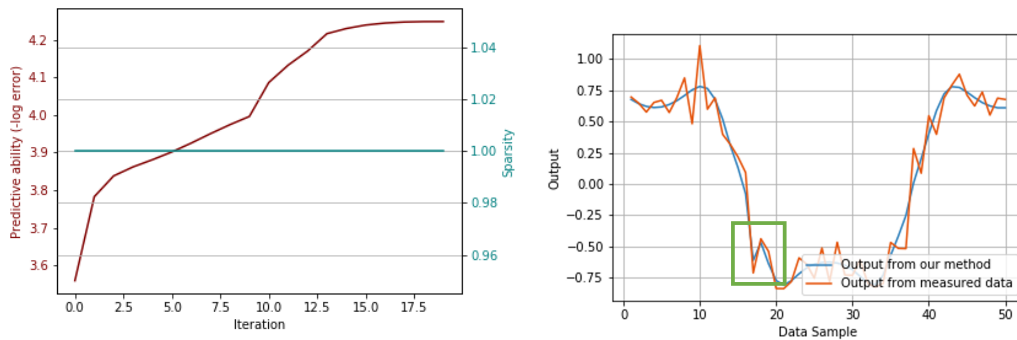
predicted results from the three optimization methods are as shown in Table 6-9. The experiment results show that only the Bayesian approach can discover the correct governing equation. However, the validation loss of these three methods does not differ a lot and even the Bayesian method has the highest one. The reason is that the data used for calculating the validation loss is the noisy extracted data. A small validation loss may even mean that the model is overfitting.

Table 6-9: Validation loss and number of cycles needed to train MathONet for different optimization methods

Optimization Method	Validation Loss	Number of cycles
Bayesian Learning	0.018	8
L1 Regularization	0.015	14
No Regularization	0.014	20



(a) Sparsity and predictive ability of the model trained with L1 regularization (b) Predicted result of the model trained with L1 regularization



(c) Sparsity and predictive ability of the model trained with no regularization (d) Predicted result of the model trained with no regularization

Figure 6-17: Predicted results of the control group model trained with L1 and no regularization

Figure 6-17 illustrates the sparsity and predictive ability of models trained with L1 and no regularization. It is clear that L1 regularization only leads to a limited decrease in terms of sparsity, and there still retains some redundant connections which may result in overfitting.

As shown in Figure 6-17(b), despite the fact that the overall predicted result is smooth and close to the correct curve, there is still overfitting in the green box area. Similarly, no redundant connection is pruned by a training process with no regularization method, which will lead to a more complex mathematical expression compared with Bayesian method and L1 regularization. The obvious overfitting can be observed in Figure 6-17(d).

To sum up, for this noisy swinging pendulum dataset, the Bayesian learning method has a better performance to compress the redundant operations in an over-parameterized MathONet and suppress overfitting, which can provide an accurate equation discovery.

Chapter 7

Conclusion

In this thesis, an end-to-end framework is designed to distill physical laws from raw videos. The whole framework is composed of a video information extraction module and a physical law discovery module. In the video information extraction module, a video can be divided into contiguous frames for object position detection. The two methods we discussed and compared are contour detection and Hough transform. The first one is especially suitable for object detection with irregular shapes, while the second method is more useful when detecting lines or circles in a complex background. Experiments in this thesis adopted contour detection for its better performance. Then, the extracted pixel position information would be transformed into physical states (e.g., distance, velocity, and angle), which serve as the dataset for discovering the governing equation. In the physical law discovery module, we designed a novel deep neural network like model Mathematical Operation Network (MathONet), which is stacked with binary operations and unary operations. MathONet can learn the closed-form expression of the governing equations of dynamic systems. An initialized MathONet is over-parameterized and contains various possible mathematical operations. A Bayesian learning algorithm is proposed to compress redundant connections and give a sparse solution to explore the physical law. We implemented the proposed framework on synthetic videos of three dynamic systems, which are a free falling ball, a Duffing oscillator and a swinging pendulum. A real-recorded video of a free falling ball was also tested to prove the validity of our method in practice. The experiment result shows that the proposed method effectively identifies the governing equations for all dataset.

7-1 Discussion

From the experiment result, the pros and cons of our framework can be summarized as:

Advantages

- The video information method is easy to deploy and implement. Instead of utilizing neural networks to extract positional features, our object position detection algorithm

is based on traditional image processing methods, which does not require complex environment deployment and a large amount of computing resource consumption.

- We proposed an interpretable deep neural network model to represent the governing equation. MathONet is consisted of possible mathematical operations coming up of the mathematical expression of the governing equation. The physical law discovery problem can be transformed into searching the optimal substructure of an over-parameterized model.
- A Bayesian deep learning approach is implemented as the discovery algorithm to compress redundant operations and discover the sparse solution. Bayesian learning algorithm is a powerful approach for model compression and overfitting suppression. Applying a Bayesian optimization for MathONet, the physical law can be discovered without any prior knowledge on the underlying equation.

Disadvantages

- The video information extraction module cannot be generalized to all dynamic systems because the decision of the object position detection method depends on the properties of the system. Besides, the transformation from pixel coordinate to physical states may require some prior information, e.g., scale and initial position.
- Using unary mathematical operations such as \sin and \log to be the activation function of a neural network may have some intrinsic drawbacks. For example, a model with periodic function as the activation function tends to fall into the local optima, while \log cannot process negative input.
- Since the Bayesian learning approach we used is based on Laplace approximation, which is sensitive to the initialization. Many repeated experiments are necessary to avoid a local optimal solution.

7-2 Further Works

In this section, we suggest several next step topics to explore. We come up with these ideas during conducting the thesis.

- The transformation parameters between pixel position and physical state can be learned through a generative model instead of relying on the prior information.
- More experiments can be implemented to evaluate our model and discovery algorithm. Especially, dynamic systems with the governing equation containing \log or \exp can be tested. In addition, dataset with high input dimensionality would be another topic worthy paying attention.
- The expressive ability of the model can be improved to fit some special operations such as division and periodic function.

Appendix A

Supplement Experiment result

The proposed method is demonstrated on a free falling ball, a Duffing oscillator and a swinging pendulum. The following is some supplement results for the three datasets.

A-1 Free Fall

The governing equation to be discovered for the synthetic video of a free falling ball is:

$$\begin{aligned}x &= 2t \\ y &= -3t + 4.9t^2\end{aligned}\tag{A-1}$$

In our experiment, MathONet is initialized with the following structure:

Table A-1: Experiment setup for free fall dataset trained with L1 and no regularization

Number of hidden layers	Number of hidden neurons	Unary functions
1	3	<i>ident, exp, log</i>

A-1-1 Distilled results of L1 regularization

The distilled result of MathONet trained with L1 regularization is:

$$\begin{aligned}x &= -0.598t^2 + 0.612 - 0.908 \times \exp(0.253t^2 - 0.557t) - 0.351 \times \exp(0.7t^2 - 0.403t) \\ &\quad - 0.814 \times \log\left(\left|0.253t^2 - 0.557t + 0.376\right|\right) - 0.541 \times \log\left(\left|0.7t^2 - 0.403t - 0.449\right|\right) + 0.032 \\ y &= 0.097t^2 + 0.304t + 0.022 \times \exp(-0.691t^2 + 0.258t) - 0.502 * \exp(0.287t^2 + 0.796t) \\ &\quad + 0.015 \times \log\left(\left|-0.691t^2 + 0.258t + 0.212\right|\right) + 0.613 \times \log\left(\left|0.287t^2 + 0.796t + 0.349\right|\right) + 0.142\end{aligned}\tag{A-2}$$

It is clear that the discovered result does not contain the correct solution and still has unnecessary exponential and logarithmic operations. The sparsity and predictive ability of MathONet after training the dataset with L1 regularization is as shown in Figure A-1. For training the first equation which is related to x and t , the model prunes only around 5% connections and remains a lot of unnecessary operations. As for the second equation related to y , no connection is compressed. Generally, for the initialized over-parameterized MathONet, L1 regularization does not have enough compression ability to suppress overfitting. Though Figure A-1(c) and A-1(f) shows that the predictive result of two models is close to the validation dataset, this is due to the strong fitting ability of the neural network but not the contribution of L1 regularization.

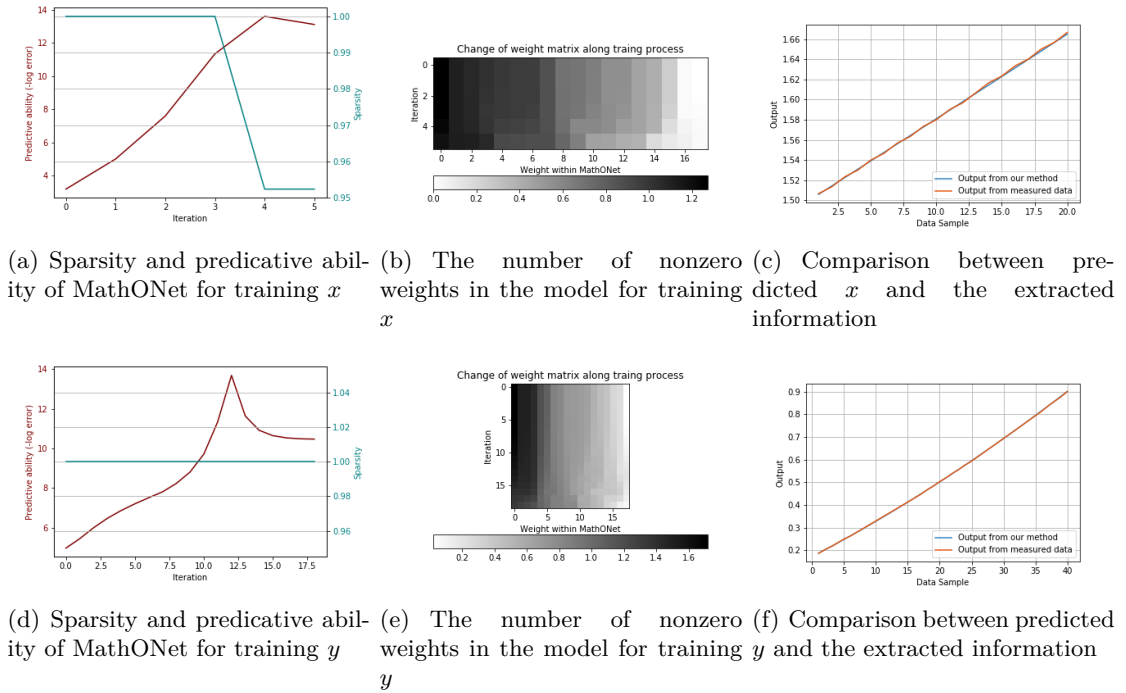


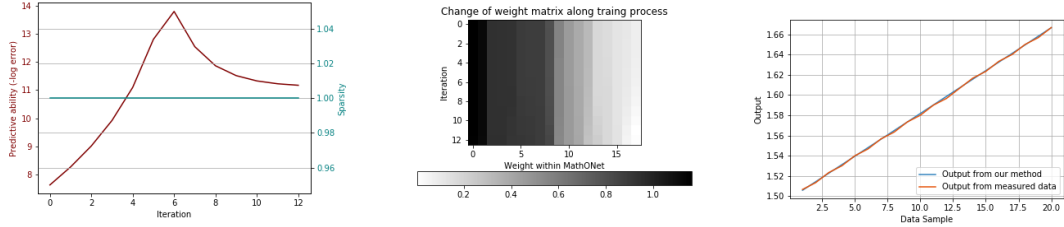
Figure A-1: The sparsity and predictive ability of MathONet trained with L1 regularization for the dataset of a free falling ball

A-1-2 Distilled results of no regularization

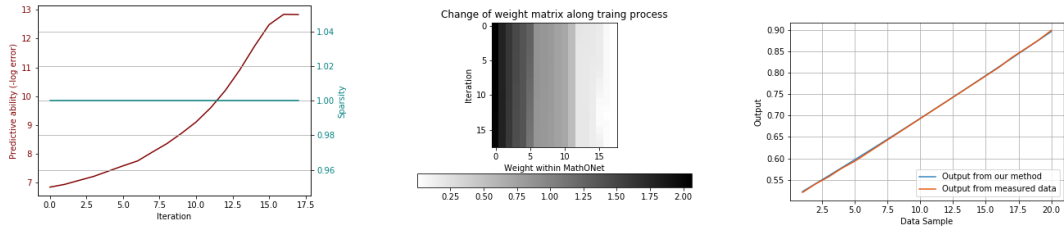
The distilled result of MathONet trained with no regularization is:

$$\begin{aligned}
 x &= 0.145t + 0.387 \times \exp(-0.175t^2 - 0.105t) + 0.75 \times \exp(0.07t^2 + 0.56t) \\
 &\quad + 0.254 \times \exp(0.318t^2 - 0.181t) + 0.455 \times \log\left(|0.07t^2 + 0.56t + 0.716|\right) \\
 &\quad + 0.516 \times \log\left(|0.175t^2 + 0.105t + 0.051|\right) + 0.12 \times \log\left(|0.318t^2 - 0.181t + 0.806|\right) + 0.333 \\
 y &= 3.89t^2 - 2.32t + 0.351 \times \exp(0.058t^2 - 0.192t) + 1.176 \times \exp(1.478t^2 - 0.873t) \\
 &\quad + 0.272 \times \log\left(|-1.478t^2 + 0.873t + 0.564|\right) + 0.128 \times \log\left(|0.058t^2 - 0.192t + 0.888|\right) - 1.357
 \end{aligned} \tag{A-3}$$

Similar to the result of L1 regularization, the result of MathONet trained with no regularization does not include the correct answer and contains redundant mathematical expression terms. The sparsity and predictive ability of MathONet after training the dataset with L1 regularization is as shown in Figure A-2.



(a) Sparsity and predicative abil- (b) The number of nonzero (c) Comparison between pre-
ity of MathONet for training x weights in the model for training x dicted x and the extracted
 x information



(d) Sparsity and predicative abil- (e) The number of nonzero (f) Comparison between predicted
ity of MathONet for training y weights in the model for training y and the extracted information
 y

Figure A-2: The sparsity and predictive ability of MathONet trained with no regularization for the dataset of a free falling ball

A-2 Duffing Oscillator

The governing equation needs to be discovered is:

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= -0.1y - x - 2x^3 \end{aligned} \tag{A-4}$$

In our experiment, MathONet is initialized with the following structure:

Table A-2: Experiment setup for free fall dataset trained with L1 and no regularization

Number of hidden layers	Number of hidden neurons	Unary functions
2	2	<i>ident</i>

A-2-1 Distilled results of L1 regularization

Video dataset

For the video dataset, the distilled result of MathONet trained with L1 regularization is:

$$\begin{aligned} x &= 0.985y - 0.061x^3 + 0.083x^2y + 0.18y^3 \\ y &= -1.711x^3 - 1.027x - 0.071y + 0.15x^2y + 0.210xy^2 \end{aligned} \quad (\text{A-5})$$

Similar to the experiment results trained with Bayesian learning method, expression terms with coefficients smaller than 0.05 have been removed. Both discovered functions contain the correct solution but have some redundant expression terms. Same as the discussion for the results of a Bayesian method, the redundant terms are raised by the noise in the extracted dataset from the video. Figure A-3 shows the sparsity and predictive ability of MathONet trained with L1 regularization.

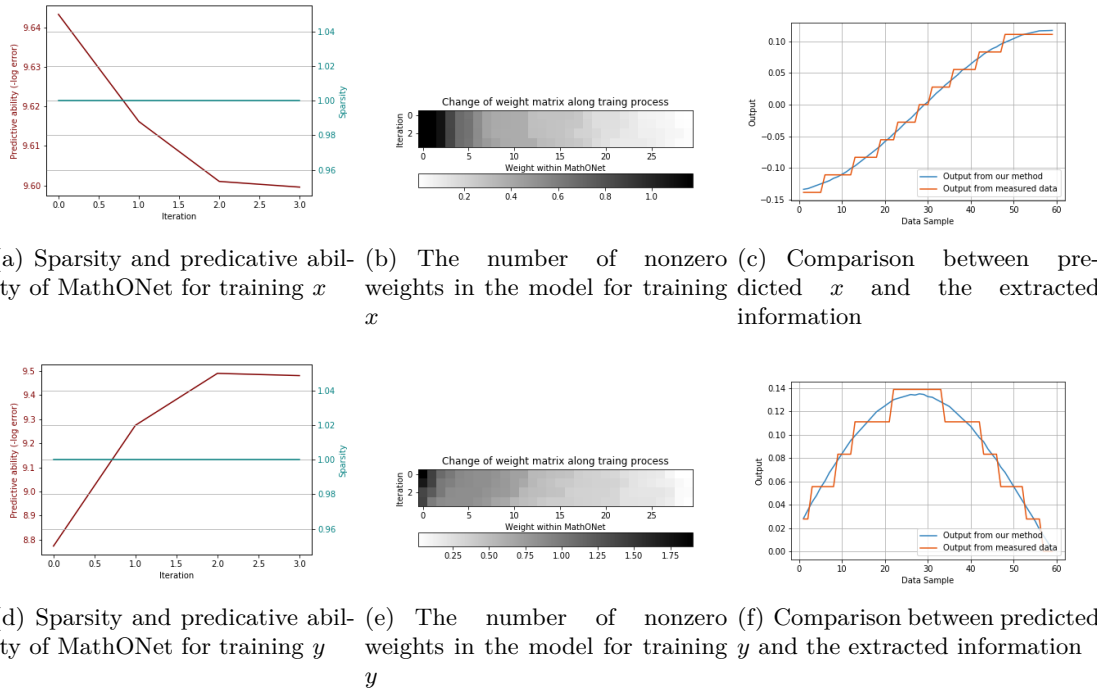


Figure A-3: The sparsity and predictive ability of MathONet trained with L1 regularization for the dataset of a Duffing oscillator

Noiseless dataset

The distilled result for the noiseless dataset is:

$$\begin{aligned} x &= 0.999y \\ y &= -1.993x^3 + 1.001x - 0.101y \end{aligned} \quad (\text{A-6})$$

The distilled result is almost same as the correct answer, which is similar to the result of a Bayesian approach.

A-2-2 Distilled results of no regularization

Video dataset

For the video dataset, the distilled result of MathONet trained with no regularization is:

$$\begin{aligned} x &= 0.983y + 0.089xy^2 + 0.206y^3 \\ y &= -1.890x^3 - 1.008x - 0.071y + 0.188x^2y + 0.089xy^2 \end{aligned} \quad (\text{A-7})$$

Obviously, the distilled result is close to the results of L1 regularization and Bayesian approach for that it contains the correct solution and includes redundant expression terms. The sparsity and predictive ability of MathONet trained with no regularization is shown in Figure ??.

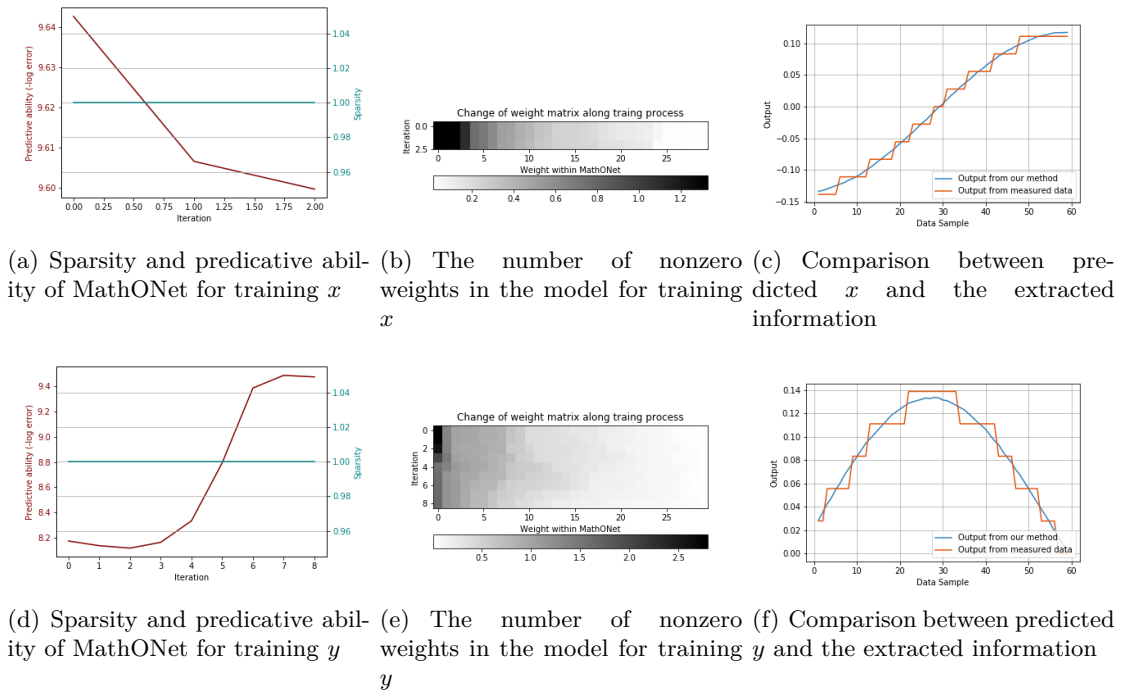


Figure A-4: The sparsity and predictive ability of MathONet trained with no regularization for the dataset of a Duffing oscillator

Noiseless dataset

The distilled result for the noiseless dataset is:

$$\begin{aligned} x &= 0.999y \\ y &= -1.994x^3 - 1.001x - 0.100y \end{aligned} \quad (\text{A-8})$$

It is nearly the same the correct answer. Considering that the initial structure of MathONet is not over-parameterized, whether the model can discover the correct answer does not depend on the discovery algorithm, but due to the fitting ability of a neural network.

A-3 Swinging Pendulum

To explore the ability of our method to recognize periodic functions, we defined two experiments. The first one is using θ as input and $\delta\dot{\theta}$ as output to train MathONet. The equation to be discovered is:

$$\delta\dot{\theta}(t+1) = 0.75 \sin(\theta(t)) \quad (\text{A-9})$$

The model is initialized with one hidden layer with one hidden neuron, and OperNet has only *ident* operation.

The second experiment is using $\dot{\theta}(t)$ and $\theta(t)$ as input variables and $\dot{\theta}(t+1)$ as output to train MathONet. Then, the equation to be discovered is :

$$\dot{\theta}(t+1) = \dot{\theta}(t) + 0.75 \sin(\theta(t)) \quad (\text{A-10})$$

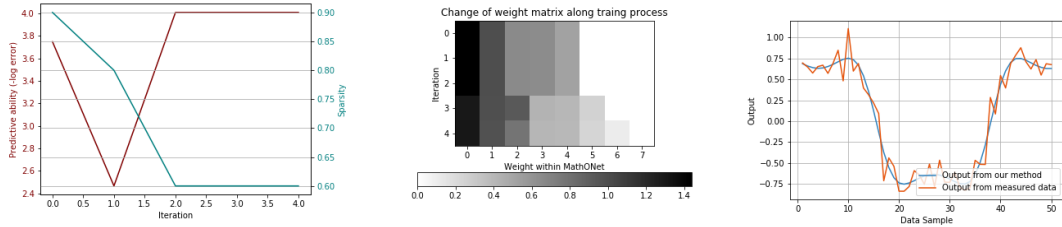
The model is initialized with one hidden layer with two hidden neurons, and the OperNet has two operations *ident* and *sin*.

A-3-1 Experiment 1

The distilled result is:

$$\delta\dot{\theta}(t+1) = 0.751 \times \sin(0.998\theta) \quad (\text{A-11})$$

The discovered result is almost close to the correct answer although the activation function is *sin*. Figure A-5 shows the sparsity and predictive ability of MathONet trained with input θ and output $\delta\dot{\theta}$.



(a) Sparsity and predictive ability of the model in each cycle (b) The number of non-zero weights in the model in each cycle (c) Comparison between predicted result and extracted data

Figure A-5: Sparsity and predictive ability of MathONet for the swinging pendulum dataset (Experiment 1)

A-3-2 Experiment 2

The distilled result is

$$\dot{\theta}(t+1) = 1.000\dot{\theta} - 0.448 \sin(0.965\theta - 2.975) - 0.163\theta(t) + 0.502 \quad (\text{A-12})$$

Our method only finds part of the correct answer and the periodic function part has wrong coefficients and redundant expression terms. The sparsity and predictive ability of our model

is as shown in Figure A-6. Although the predicted result can nearly fit the validation set, the discovered result is far from the correct solution. This may be raised by the intrinsic of a sinusoidal function for that it has many local optimal solutions.

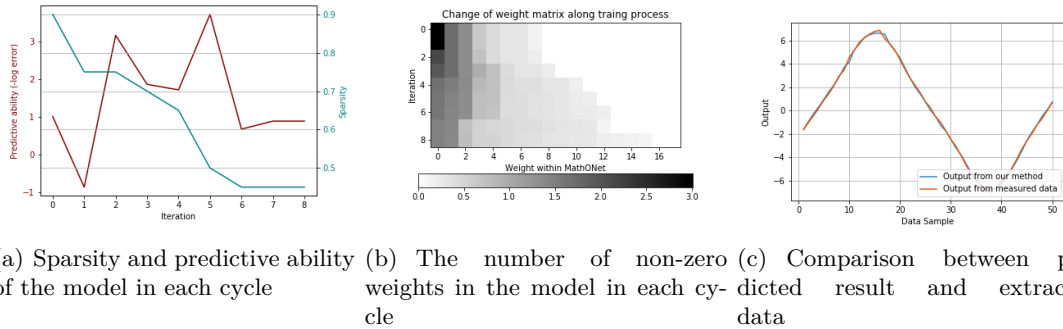


Figure A-6: Sparsity and predictive ability of MathONet for the swinging pendulum dataset (Experiment 2)

Real Dataset for Pendulum

We also implemented our method on a real pendulum dataset, which was recorded with a ROS framework. The data collection process and the experiment result are as follows.

B-1 Data Collection

B-1-1 ROS framework

As shown in Figure B-1, the real pendulum is fixed on a KUKA robot, while there is an optical tracking system to capture the ground truth of the position of the pendulum in the Cartesian coordinate system. The video is recorded through a RealSense stereo camera with a frame rate of 60 frames per second. These sensors are all connected to a ROS framework so that a new ROS node can be established to subscribe to the message from all sensors.

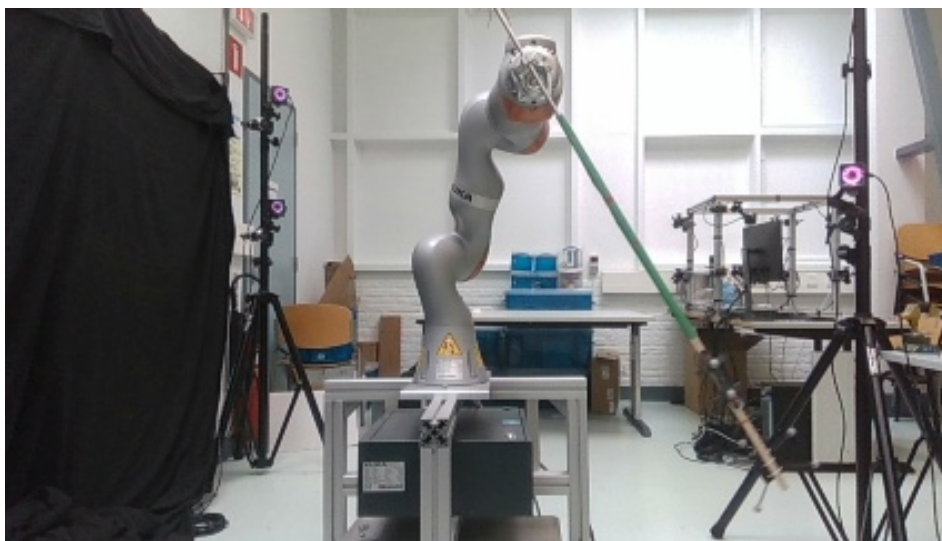


Figure B-1: A screenshot of the video dataset of a real pendulum

Our new ROS node subscribes to three ros topics. The first one is 'kuka_ee', which records the vertical and horizontal position of the manipulator of the KUKA robot. The second topic is '/vrpn_client_node/RigidBody1/pose', recording the Cartesian coordinate position of the pendulum from the optical tracking system. The last one is '/camera/color/image_raw' representing the images from the RealSense node. At the same time, our node publishes the action command to the topic 'iiwa/CustomControllers/command' to control the next step of the manipulator of KUKA robot. Since the pendulum is released with an initial angle, the action command keeps being stable.

B-1-2 Python code

```

1
2 import numpy as np
3 import time # CC
4 import rospy # CC
5 import cv2
6 import cv_bridge
7 import os
8 from cv_bridge import CvBridge
9 from std_msgs.msg import String # CC
10 from sensor_msgs.msg import JointState
11 from sensor_msgs.msg import Image
12 from std_msgs.msg import Float64MultiArray
13 from geometry_msgs.msg import PoseStamped
14
15
16 def kukaee_call(data): #robot pose
17     global position
18     position = data.data
19
20 def body_call(data): #Pendulum pose
21     #rospy.loginfo(rospy.get_caller_id() + "I heard pose %s", data.pose)
22     global alpha, xB, yB
23     xB = data.pose.position.x
24     yB = data.pose.position.y
25
26 img_count = 0
27 bridge = CvBridge()
28 img_path = 'realsense_dataset/pendulum/dataset3_action2/img/'
29 data_path = 'realsense_dataset/pendulum/dataset3_action2'
30 if not os.path.exists(img_path):
31     os.makedirs(img_path)
32 file = open(os.path.join(data_path, 'data'), 'w')
33 flag = 0
34
35 def img_call(img):
36     global img_count, bridge, img_path, data_path, flag, action
37     if flag == 1:
38         img_count += 1
39         cv_image = bridge.imgmsg_to_cv2(img, 'bgr8')
40         img_name = str(img_count) + '.jpg'

```

```

41
42     img_savepath = img_path + image_name
43
44     cv2.imwrite(img_savepath, cv_image)
45     cv2.imshow('frame', cv_image)
46     cv2.waitKey(3)
47     rospy.loginfo('Image %d saved!', img_count)
48
49     file.write('%s %d %.10f %.10f %.10f %.10f %.10f\n' % (
        img_savepath, action, float(xB), float(yB), float(position[0])
        , float(position[1]), float(position[2])))
50
51 def reset():
52     global count, posrequest, restart_flag, nomove
53     count = 0
54     pose1.data = [0, np.pi / 2, 0, 0.65, 0.2, 0.6]
55     posrequest = [0.65, 0.2, 1]
56     pub_pos.publish(pose1)
57     rospy.sleep(5.0)
58     #state = get_state()
59     restart_flag = True
60     nomove = False
61     #return [state]
62
63 def get_state():
64     global alpha, alpha_1, xB1, xB2_r_1, position, fish_1
65     fish = position[1] - xB1
66     fish_vel = fish - fish_1
67     velocity = np.array(position) - np.array(position_1)
68     state = np.array([ position[1], velocity[1], fish, fish_vel])
69     position = position_1
70     fish_1 = fish
71
72     return state
73
74 def step(action):
75     global count, posrequest, yB1, restart_flag, nomove, posnomove
76     poslimit = [0.65, 0.65, -0.3, 0.3, 1, 1] # xlow xup ylow yup zlow zup
77     delta = 1.5
78     """if not nomove:
79         posnomove = position[1]
80     if action == 0:
81         posrequest[1] = position[1] - delta
82         nomove = False
83     if action == 1: # CC
84         posrequest[1] = position[1] + delta
85         nomove = False
86     if action == 2:
87         posrequest[1] = posnomove
88         nomove = True"""
89     if not nomove:
90         posnomove = position[1]
91     if action == 0:

```

```

92     posrequest[1] = position[1] - delta
93     nomove = False
94     if action == 1: # CC
95         posrequest[1] = position[1] - delta/3
96         nomove = False
97     if action == 2:
98         posrequest[1] = posnomove
99         nomove = True
100    if action == 3:
101        posrequest[1] = position[1] + delta/3
102        nomove = False
103    if action == 4: # CC
104        posrequest[1] = position[1] + delta
105        nomove = False
106    #print(poslimit)
107    #print(posrequest)
108    posrequest[1] = poslimit[2] if posrequest[1] < poslimit[2] else
        posrequest[1]
109    posrequest[1] = poslimit[3] if posrequest[1] > poslimit[3] else
        posrequest[1]
110    posrequest[2] = poslimit[4] if posrequest[2] < poslimit[4] else
        posrequest[2]
111    posrequest[2] = poslimit[5] if posrequest[2] > poslimit[5] else
        posrequest[2]
112    pose1.data = [0, np.pi / 2, 0, posrequest[0], posrequest[1],
        posrequest[2]]
113
114    #state = get_state()
115
116    """while restart_flag or yB1 > position[2]:
117        pose1.data = [0, np.pi / 2, 0, 0.6, 0.2, 0.6]
118        pub_pos.publish(pose1)
119        rospy.sleep(0.1)
120        #print(11111)
121        state = get_state() #
122        if yB1 > position[2]:
123            restart_flag = False"""
124    pub_pos.publish(pose1)
125    count = count + 1
126
127    #return state
128
129 def main():
130     global alpha, alpha_1, position, position_1, pub_pos, pose1,
        posrequest, xB1, fish_1, h_joy, flag, action
131     rospy.init_node('teleop', anonymous=True)
132     rospy.Subscriber("kuka_ee", Float64MultiArray, kukaee_call)
133     rospy.Subscriber("/vrpn_client_node/RigidBody1/pose", PoseStamped,
        body_call)
134     rospy.Subscriber("/camera/color/image_raw", Image, img_call)
135     pub_pos = rospy.Publisher("iiwa/CustomControllers/command",
        Float64MultiArray, queue_size=1)
136     pose1 = Float64MultiArray()

```

```

137     time.sleep(3)
138     pose1.data = [0 * np.pi / 4, np.pi / 2, 0, position[0], position[1],
139                 position[2]]
139     pub_pos.publish(pose1)
140     posrequest = [0.65, position[1], position[2]] # position[0]
141     reset()
142     ee = Float64MultiArray()
143     r = rospy.Rate(50) # frquency
144     action = 2
145     timer = 0
146     while (True):
147         flag = 1
148         step(action)
149         timer += 1
150         r.sleep()
151         if rospy.is_shutdown():
152             print('shutdown')
153             break
154
155 if __name__ == "__main__":
156     main()

```

B-2 Video Information Extraction

Since the background environment of the pendulum is complex and contains other straight lines, a background subtraction is applied to eliminate the influence of the complex background and remain the area of interest. Then, a Hough transform can be utilized to detect the slope of the existing line in the figure.

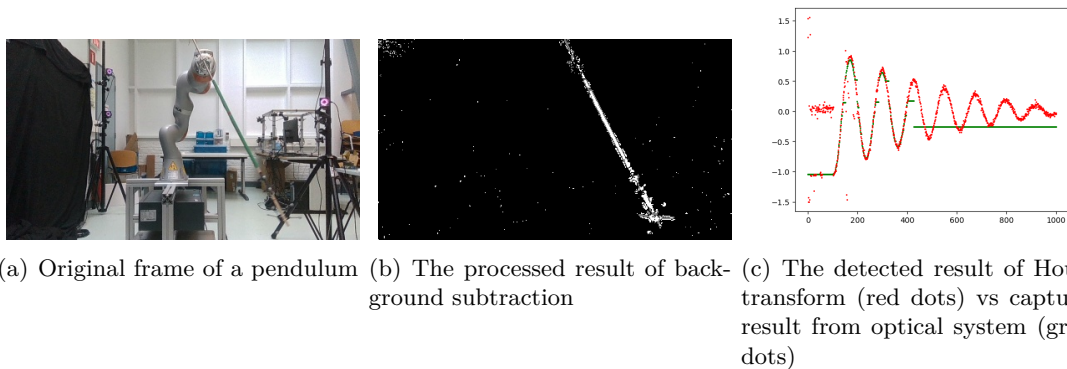


Figure B-2: The process of video information extraction of the real pendulum video

Figure B-2(a) is the original frame from the video, while Figure B-2(b) shows the background subtraction result. Nearly all unnecessary information has been removed from the figure and only the silhouette of the pendulum is retained. Figure B-2(c) illustrates the detected result, with the x axis being the index of frames and y axis being the angle θ . The red dots represent the detected angle from Hough transform while the green dots stand for the captured value

from optical system. Due to the limitations of the optical tracking system, some captured values are constants. However, according to the valid real value information, it can be found that the Hough transform detection result is basically close to the real value.

B-3 Experiment Result

For the governing equation to be discovered, an extra unknown resistance is applied to the pendulum compared with (6-9). With the moment inertia still being $I = \frac{1}{3}ml^2$, the torque is raised by the gravity and resistance $M = \frac{1}{2}mg \sin \theta + u_r$. Then, the angular acceleration can be represented as:

$$\ddot{\theta}(t+1) = \frac{3g}{2l} \sin \theta(t) + \frac{3u_r}{ml^2} \quad (\text{B-1})$$

However, the resistance torque u_r is unknown and varied, which means that the specific equation cannot be distilled when missing the information of the extra resistance torque. The experiment result also proves the conclusion. With $\sin \theta(t)$ and the constant term as input variable, and $\ddot{\theta}(t+1) \cdot dt$ as output variable, the discovered result is:

$$\ddot{\theta}(t+1) \cdot dt = 0.023 \sin \theta(t)^2 + 0.037 \sin \theta(t) + 0.003 \quad (\text{B-2})$$

which only tries to fit the dataset but does not have physical meaning.

Bibliography

- [1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. Ieee, 2017.
- [2] Thomas Bäck, David B Fogel, and Zbigniew Michalewicz. *Evolutionary computation 1: Basic algorithms and operators*. CRC press, 2018.
- [3] Dana H Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2):111–122, 1981.
- [4] Kiran S Bhat, Steven M Seitz, Jovan Popović, and Pradeep K Khosla. Computing the physical parameters of rigid-body motion from video. In *European Conference on Computer Vision*, pages 551–565. Springer, 2002.
- [5] Christopher M Bishop. Pattern recognition. *Machine learning*, 128(9), 2006.
- [6] Christopher M Bishop and Cazhaow S Qazaz. Regression with input-dependent noise: A bayesian treatment. *Advances in neural information processing systems*, pages 347–353, 1997.
- [7] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR, 2015.
- [8] Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical gauss-newton optimisation for deep learning. In *International Conference on Machine Learning*, pages 557–565. PMLR, 2017.
- [9] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

- [10] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.
- [11] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.
- [12] Kathleen Champion, Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116(45):22445–22451, 2019.
- [13] Pradyumna Chari, Chinmay Talegaonkar, Yunhao Ba, and Achuta Kadambi. Visual physics: Discovering physical laws from videos. *arXiv preprint arXiv:1911.11893*, 2019.
- [14] Zhenfang Chen, Jiayuan Mao, Jiajun Wu, Kwan-Yee Kenneth Wong, Joshua B Tenenbaum, and Chuang Gan. Grounding physical concepts of objects and events through dynamic visual reasoning. *arXiv preprint arXiv:2103.16564*, 2021.
- [15] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008.
- [16] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.
- [17] Misha Denil, Babak Shakibi, Laurent Dinh, Marc’Aurelio Ranzato, and Nando De Freitas. Predicting parameters in deep learning. *arXiv preprint arXiv:1306.0543*, 2013.
- [18] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. *arXiv preprint arXiv:1511.07404*, 2015.
- [19] David E Goldberg and John Henry Holland. Genetic algorithms and machine learning. 1988.
- [20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- [21] Yanming Guo, Yu Liu, Ard Oerlemans, Songyang Lao, Song Wu, and Michael S Lew. Deep learning for visual understanding: A review. *Neurocomputing*, 187:27–48, 2016.
- [22] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. *arXiv preprint arXiv:1608.04493*, 2016.
- [23] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International conference on machine learning*, pages 1737–1746. PMLR, 2015.
- [24] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

-
- [25] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.
- [26] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13, 1993.
- [27] Kenneth J Hunt, D Sbarbaro, R Żbikowski, and Peter J Gawthrop. Neural networks for control systems—a survey. *Automatica*, 28(6):1083–1112, 1992.
- [28] Ying Jin, Weilin Fu, Jian Kang, Jiadong Guo, and Jian Guo. Bayesian symbolic regression. *arXiv preprint arXiv:1910.08892*, 2019.
- [29] Salman Khan, Hossein Rahmani, Syed Afaq Ali Shah, and Mohammed Bennamoun. A guide to convolutional neural networks for computer vision. *Synthesis Lectures on Computer Vision*, 8(1):1–207, 2018.
- [30] Samuel Kim, Peter Y Lu, Srijon Mukherjee, Michael Gilbert, Li Jing, Vladimir Čeperić, and Marin Soljačić. Integration of neural network-based symbolic regression in deep learning for scientific discovery. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.
- [31] Yann LeCun, John S Denker, Sara A Solla, Richard E Howard, and Lawrence D Jackel. Optimal brain damage. In *NIPs*, volume 2, pages 598–605. Citeseer, 1989.
- [32] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.
- [33] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. *arXiv preprint arXiv:1705.08665*, 2017.
- [34] Lele Luan, Yang Liu, and Hao Sun. Uncovering closed-form governing equations of nonlinear dynamics from videos. *arXiv preprint arXiv:2106.04776*, 2021.
- [35] David JC MacKay. Bayesian interpolation. *Neural computation*, 4(3):415–447, 1992.
- [36] David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.
- [37] Georg Martius and Christoph H Lampert. Extrapolation and learning equations. *arXiv preprint arXiv:1610.02995*, 2016.
- [38] Paul Merolla, Rathinakumar Appuswamy, John Arthur, Steve K Esser, and Dharmendra Modha. Deep neural networks are robust to weight binarization and other non-linear distortions. *arXiv preprint arXiv:1606.01981*, 2016.
- [39] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pages 2498–2507. PMLR, 2017.

- [40] Roozbeh Mottaghi, Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. Newtonian scene understanding: Unfolding the dynamics of objects in static images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3521–3529, 2016.
- [41] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.
- [42] Isaac Newton. *Philosophiæ naturalis principia mathematica*, volume 2. typis A. et JM Duncan, 1833.
- [43] Felice Andrea Pellegrino, Walter Vanzella, and Vincent Torre. Edge detection revisited. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(3):1500–1518, 2004.
- [44] Brenden K Petersen, Mikel Landajuela Larma, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*, 2019.
- [45] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
- [46] Samuel H Rudy, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4):e1602614, 2017.
- [47] Samuel H Rudy, J Nathan Kutz, and Steven L Brunton. Deep learning of dynamics and signal-noise decomposition with time-stepping constraints. *Journal of Computational Physics*, 396:483–506, 2019.
- [48] Tim Salimans, Diederik Kingma, and Max Welling. Markov chain monte carlo and variational inference: Bridging the gap. In *International Conference on Machine Learning*, pages 1218–1226. PMLR, 2015.
- [49] Peter J Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of fluid mechanics*, 656:5–28, 2010.
- [50] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85, 2009.
- [51] Michael E Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of machine learning research*, 1(Jun):211–244, 2001.
- [52] Silviu-Marian Udrescu and Max Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.
- [53] Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.
- [54] Seul-Ki Yeom, Philipp Seegerer, Sebastian Lapuschkin, Alexander Binder, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Pruning by explaining: A novel criterion for deep neural network pruning. *Pattern Recognition*, page 107899, 2021.

- [55] Alan L Yuille and Anand Rangarajan. The concave-convex procedure. *Neural computation*, 15(4):915–936, 2003.
- [56] Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. Bayesnas: A bayesian approach for neural architecture search. In *International Conference on Machine Learning*, pages 7603–7613. PMLR, 2019.
- [57] Liu Ziyin, Tilman Hartwig, and Masahito Ueda. Neural networks fail to learn periodic functions and how to fix it. *arXiv preprint arXiv:2006.08195*, 2020.

