Bayesian deep learning for system identification

Zhou, H.

DOI
10.4233/uuid:94b0cdd5-280b-4afb-a210-f19ecf12cf66

Publication date
2022

Document Version
Final published version

Citation (APA)
Zhou, H. (2022). *Bayesian deep learning for system identification*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:94b0cdd5-280b-4afb-a210-f19ecf12cf66

BAYESIAN DEEP LEARNING FOR SYSTEM IDENTIFICATION

BAYESIAN DEEP LEARNING FOR SYSTEM IDENTIFICATION

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology,
by the authority of the Rector Magnificus Prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates,
to be defended publicly on
Thursday 12 May 2022 at 15:00 o'clock

by

Hongpeng ZHOU

Master of Engineering in Control Science and Engineering,
Harbin Institute of Technology, China
born in Nanyang, China

This dissertation has been approved by the promotors

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. ir. M. Wisse,	Delft University of Technology, promotor
Dr. W. Pan,	Delft University of Technology, copromotor
Independent members:	
Prof. dr. ir. J. Hellendoorn,	Delft University of Technology
Prof. dr. ir. M. H. G. Verhaegen,	Delft University of Technology
Prof. dr. ir. J. Schoukens,	Vrije Universiteit Brussel, Belgium
Dr. ir. R. Tóth	Eindhoven University of Technology
Dr. -ing J. Kober,	Delft University of Technology, reserve member
Other member:	
Prof. dr. W. X. Zheng,	University of Western Sydney, other member

This research was financially supported by China Scholarship Council (CSC).

Keywords: System identification, Deep neural networks, Sparse Bayesian learning, Hessian calculation, Symbolic regression, Neural architecture search, Network compression

Printed by: Ridderprint, the Netherlands

Front & Back: Designed by Hongpeng Zhou.

ISBN 978-94-6384-329-4

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

Dedicated to:
my family for supporting me in this journey.

Hongpeng Zhou

CONTENTS

SUMMARY

Applying deep neural networks (DNNs) for system identification (SYSID) has attracted more and more attention in recent years. The DNNs, which have universal approximation capabilities for any measurable function, have been successfully implemented in SYSID tasks with typical network structures, e.g., feed-forward neural networks and recurrent neural networks (RNNs). However, DNNs also have limitations. First, DNNs can easily overfit the training data due to the model complexity. Second, DNNs are normally regarded as black-box models, which lack interpretability and cannot be used for white-box modelling. In this thesis, we develop sparse Bayesian deep learning (SBDL) algorithms that can address these limitations in an effective manner.

In Chapter 2, we present the Bayesian treatment of DNNs for SYSID by adopting the Laplace approximated method to approximate the posterior distribution of model parameters. However, although the Laplace approximated method can scale well to DNNs, it requires the calculation of the inverse Hessian of model parameters. This is infeasible for DNNs due to the intensive computation and storage burden. To address this challenge, we develop efficient and recursive Hessian calculation methods for DNN layers (i.e., fully-connected, convolutional and recurrent layers), which can extract the block-diagonal value of the Hessian and be calculated recursively along with the backpropagation process. Compared to a previous Hessian calculation approach, the required multiply accumulate operation (MACs) with the proposed Hessian calculation method could be reduced from $n(2m^2+2n^2+4mn+3m-1)$ to $n(2+4m)$ with $W \in \mathbb{R}^{n\times m}$ (e.g., if $n = 100, m = 100$, the original method requires 107.97 MMACs compared with only 0.04 MMACs for the proposed method.). The presented Hessian calculation methods turn the intractable optimization problem into a tractable one. Besides, as effective sparse regression solutions, the proposed SBDL algorithms prune the model redundancy by employing sparsity-inducing priors on the model parameters. To extend the generalization ability of the Bayesian approach, three kinds of priors are considered in this thesis, i.e., single prior, group prior, and fused prior. The selection of priors will affect both training and pruning results and is adapted to different applications' specifics. Specifically, the single prior is enforced on a single parameter and can be used to achieve non-structural model sparsity. The group prior is enforced on a group of parameters and can be used to obtain the structural sparsity. The fused prior is a combination of single prior and group prior, and can be used to achieve both non-structural and structural sparsity. The detailed procedures for the Hessian calculation and three SBDL algorithms enforcing single prior, group prior and fused prior, respectively, are explained in this chapter.

Chapter 3 investigates how to implement the SBDL algorithm with the single prior to identifying a repressilator model. The repressilator model is a classical nonlinear dynamical system in synthetic biology to represent mRNA transcription and and protein translation dynamics. It can be described by nonlinear ordinary differential equations (ODEs) involving polynomial and rational functions. Identifying the topology and parameters of

the repressilator model is a typical research problem that has attracted a lot of attention. In this chapter, we design a combined neural network that consists of a linear and nonlinear sub-network. The activation function of the nonlinear sub-network is replaced by the Hill function, which reflects the binding of ligands to macromolecules (e.g., mRNA and protein). With such model design, an initialized combined neural network can be regarded as a super-graph with a redundant structure, whose sub-graph can represent the underlying mathematical model. The SBDL algorithm with the single prior, which can achieve the nonstructured model sparsity, is adopted to identify the redundancy. Finally, both the topology and parameters of the repressilator model can be identified. To address several typical problems in SYSID, including input feature selection and easily overfitting to the training dataset, the SBDL algorithm with the group prior is applied in Chapter 4. With the multi-layer perceptron (MLP) and long short term memory networks (LSTM) as the backbone, the Bayesian approach offers a solution by marginal likelihood/model evidence approximation and structured group sparsity-inducing priors construction. In this way, the input features of DNN models (i.e., MLP and LSTM) can be selected. The structural model sparsity can also alleviate the overfitting issue. Besides, a practical calculation approach based on the Monte-Carlo integration method is derived for the uncertainty quantification of the parameters and predictions. The effectiveness of the method is demonstrated on several linear and nonlinear system identification benchmarks. Specifically, the proposed method can achieve competitive simulation accuracy with the same magnitude as other system identification methods adopting classical model types, e.g., state-space method, autoregressive with exogenous terms (ARX) model. Especially, we can obtain the best simulation accuracy compared with existing methods in the cascaded tank system.

In Chapter 5, the governing equations of several dynamic systems can be discovered from data directly with the benefit of two key contributions. First, we present how to apply the SBDL algorithm with the fused prior to the white-box modelling. Second, we design a new representation of mathematical expressions with a NN-like hierarchical structure, termed as Mathematical Operation Network (MathONet). The MathONet is stacked by layers consisting of unary (e.g. sin, cos, log), and binary operations (e.g. $+, -, \times$). An initialized MathONet is typically regarded as a over-parameterized model, whose sub-graph can yield the underlying governing equation. By applying the SBDL algorithm, the essential sub-graph can be extracted by employing both non-structurally and structurally constructed priors over the model parameters. To ensure a connected derived graph after pruning, the connections' dependencies are also considered in this chapter. It should also be noted that since the structure of MathONet is similar to a neural network, the proposed method belongs to the network-based symbolic regression approach. It provides an encouraging solution that can be trained end-to-end through backpropagation on a fully developed deep learning framework (e.g., PyTorch and TensorFlow). The proposed approach can identify ordinary differential equations (ODEs) or partial differential equations (PDEs) from observations for several linear and nonlinear dynamic systems, i.e., Kolmogorov–Petrovsky–Piskunov, Lotka-Volterra and chaotic Lorenz systems. The extensions of the proposed Bayesian approaches to two deep learning topics with high-dimensional datasets (i.e., the neural architecture search (NAS) and neural network compression) are also discussed in Chapter 6.

SAMENVATTING

Het toepassen van diepe neurale netwerken (DNNs) voor systeem identificatie (SYSID) krijgt de laatste jaren steeds meer aandacht. De DNNs, die elke meetbare functie universeel kunnen benaderen, zijn met succes geïmplementeerd in SYSID-taken met typische netwerkstructuren, zoals feed-forward neurale netwerken en recurrente neurale netwerken (RNNs). DNNs hebben echter ook beperkingen. Ten eerste kunnen DNNs de trainingsdata gemakkelijk overfitten vanwege de complexiteit van het model. Ten tweede worden DNNs normaal gesproken beschouwd als black-box-modellen, die niet interpreteerbaar zijn en niet kunnen worden gebruikt voor white-box-modellering. In dit proefschrift ontwikkelen we algoritmen voor schaars Bayesiaans diep leren (SBDL) die deze beperkingen op een effectieve manier kunnen aanpakken.

In Hoofdstuk 2 presenteren we de Bayesiaanse behandeling van DNN's voor SYSID door de Laplace-benaderde methode toe te passen om de posterieure verdeling van modelparameters te benaderen. Hoewel de Laplace-benaderde methode goed kan schalen naar DNN's, vereist deze de berekening van de inverse Hessiaan van modelparameters. Dit is niet haalbaar voor DNN's vanwege de intensieve reken- en opslaglast. Om deze uitdaging aan te gaan, ontwikkelen we efficiënte en recursieve Hessiaan berekeningsmethoden voor DNN-lagen (d.w.z. volledig verbonden, convolutieve en terugkerende lagen), die de blokdiagonale waarde van de Hessiaan kunnen extraheren en recursief kunnen worden berekend samen met het terugpropagatie-proces. Vergeleken met een eerdere berekeningsmethode van de Hessiaan, zou de vereiste vermenigvuldigen-accumuleren operaties (MACs) met de voorgestelde methode kunnen worden teruggebracht van $n(2m^2 + 2n^2 + 4mn + 3m - 1)$ naar $n(2 + 4m)$ met $W \in \mathbb{R}^{n \times m}$ (bijv. met $n = 100, m = 100$ vereist de oorspronkelijke methode 107,97 MMACs, in vergelijking tot slechts 0,04 MMACs voor de voorgestelde methode). De gepresenteerde Hessiaan-berekeningsmethoden maken van het onhandelbare optimalisatieprobleem een handelbaar probleem. Bovendien, als een effectieve schaarse regressie oplossing, verminderen de voorgestelde SBDL-algoritmen de modelredundantie door gebruik te maken van schaarsheid inducerende priors op de modelparameters. Om het generalisatievermogen van de Bayesiaanse benadering uit te breiden, worden in dit proefschrift drie soorten priors beschouwd, d.w.z. enkele prior, groep prior en samengestelde prior. De selectie van priors is van invloed op zowel de trainings- als de getrimde resultaten en moet worden aangepast aan de specifieke kenmerken van de verschillende toepassingen. In het bijzonder wordt de enkele prior afgedwongen op een enkele parameter en kan deze worden gebruikt om niet-structurele modelschaarsheid te bereiken. De groep prior wordt afgedwongen op een groep parameters en kan worden gebruikt om de structurele schaarsheid te verkrijgen. De samengestelde prior is de combinatie van de enkele prior en een groep prior en kan worden gebruikt om zowel niet-structurele als structurele schaarsheid te bereiken. De gedetailleerde procedures voor berekening van de Hessiaan en drie SBDL-algoritmen die respectievelijk enkele prior, groep prior en samengestelde prior afdwingen, worden in dit hoofdstuk uitgelegd.

Hoofdstuk 3 onderzoekt hoe het SBDL-algoritme kan worden geïmplementeerd met de enkele prior. Het repressilator model is een klassiek niet-lineair dynamisch systeem in de synthetische biologie om mRNA-transcriptie en eiwittranslatie dynamiek weer te geven. Het kan worden beschreven door niet-lineaire gewone differentiaalvergelijkingen (GDVs) met polynome en rationele functies. Het identificeren van de topologie en parameters van het repressilator model is een typisch onderzoeksprobleem dat veel aandacht heeft gekregen. In dit hoofdstuk ontwerpen we een gecombineerd neuraal netwerk dat bestaat uit een lineair een niet-lineair subnetwerk. De activatiefunctie van het niet-lineaire subnetwerk wordt vervangen door de Hill-functie, die de binding van liganden aan macromoleculen (bijv. mRNA en eiwit) weerspiegelt. Met een dergelijk modelontwerp kan een geïnitialiseerd gecombineerd neuraal netwerk worden beschouwd als een supergraaf met een redundante structuur, waarvan de subgraaf het onderliggende wiskundige model kan vertegenwoordigen. Het SBDL-algoritme met de enkele prior, die de niet-gestructureerde modelschaarsheid kan bereiken, wordt gebruikt om de overtolligheid te identificeren. Ten slotte kunnen zowel de topologie als de parameters van het repressilator model nauwkeurig worden geïdentificeerd. Om een aantal typische problemen in SYSID aan te pakken, waaronder de selectie van invoerkenmerken en het gemakkelijk overfitten van de trainingsdataset, wordt het SBDL-algoritme met de groep prior toegepast in Hoofdstuk 4. Met het meerlaagse perceptron (MLP) en lange kortetermijngeheugen netwerken (LSTM) als de ruggengraat, biedt de Bayesiaanse benadering een oplossing door priors te construeren die leiden tot marginale waarschijnlijkheid/benadering van modelbewijs en gestructureerde groepsschaarsheid. Op deze manier kunnen de invoerkenmerken van DNN-modellen (d.w.z. MLP en LSTM) worden geselecteerd. The structurele modelschaarsheid kan ook het probleem van overfitten verlichten. Daarnaast wordt een praktische berekeningsaanpak afgeleid op basis van de Monte-Carlo-integratiemethode voor de kwantificering van de onzekerheid van de parameters en voorspellingen. De effectiviteit van de methode is aangetoond op verschillende lineaire en niet-lineaire systeem identificatie benchmarks. In het bijzonder kan de voorgestelde methode competitieve simulatienauwkeurigheid bereiken met dezelfde omvang als andere systeem identificatie methoden die klassieke modeltypen gebruiken, bijv. state-space methode, autoregressief model met exogene termen (ARX) model. Met name kunnen we de beste simulatienauwkeurigheid verkrijgen in vergelijking met bestaande methoden in het trapsgewijze tanksysteem.

In Hoofdstuk 5 kunnen de geldende vergelijkingen van verschillende dynamische systemen direct uit data worden ontdekt met het voordeel van twee belangrijke bijdragen. Eerst presenteren we hoe het SBDL-algoritme kan worden toegepast op de white-box-modellering met de samengestelde prior. In de tweede plaats ontwerpen we een nieuwe representatie van wiskundige uitdrukkingen met een NN-achtige hiërarchische structuur, genaamd mathematisch operatienetwerk (MathONet). De MathONet is gestapeld door lagen bestaande uit unaire (bijv. sin, cos, log) en binaire operaties (bijv. $+, -, \times$). Een geïnitialiseerd MathONet wordt doorgaans beschouwd als een subgraaf, waarvan de supergraaf de onderliggende geldende vergelijkingen kan opleveren. Door het SBDL-algoritme toe te passen met de samengestelde prior (enkele prior & groep prior), kan de essentiële subgraaf worden geëxtraheerd door zowel niet-structureel als structureel geconstrueerde priors over de modelparameters te gebruiken. Om na trimmen een samenhangende afgeleide graaf te garanderen, wordt in dit hoofdstuk ook rekening gehouden met de afhan-

kelijkheden van de verbindingen. Er moet worden opgemerkt dat, aangezien de structuur van MathONet vergelijkbaar is met een neuraal netwerk, de voorgestelde methode behoort tot de netwerkgebaseerde symbolische regressiebenadering. Het biedt een bemoedigende oplossing die eind-tot-eind kan worden getraind door terugpropagatie op een volledig ontwikkeld raamwerk voor diep leren (bijv. PyTorch en TensorFlow). De voorgestelde aanpak kan gewone differentiaalvergelijkingen (GDVs) of partiële differentiaalvergelijkingen (PDVs) identificeren uit waarnemingen voor verschillende lineaire en niet-lineaire dynamische systemen, dat wil zeggen, Kolmogorov-Petrovsky-Piskunov, Lotka-Volterra en chaotische Lorenz-systemen. De uitbreidingen van de voorgestelde Bayesiaanse benadering voor twee onderwerpen betreffende diep leren met hoog dimensionale datasets (d.w.z. neurale architectuur zoeken (NAS) en neuraal netwerk compressie) worden ook besproken in Hoofdstuk 6.

1

INTRODUCTION

The implementation of deep neural networks (DNNs) for system identification has regained research interest recently, thanks to the boom of deep learning. DNNs have shown impressive approximation ability in various fields, but there are still several challenges. First, DNNs are known to be too complex to overfit the training data easily. Second, DNNs are black-box models, which cannot be implemented for physical modelling tasks to uncover the underlying phenomenon of the dynamic system. We address these challenges through three aspects. First, we develop sparse Bayesian deep learning algorithms (SBDLs) to approximate the posterior distribution of model parameters. Second, the efficient and recursive Hessian calculation methods for the Fully-Connected (FC) layer, convolutional (Conv) layer, and recurrent layer are also proposed, respectively. Third, we design a DNN-like hierarchical structure composed of mathematical operations, termed as Mathematical Operation Network (MathONet), to learn the explicit physical model of a dynamic system.

This chapter illustrates the research background in Section. 1.1 and summaries the research contribution in Section 1.2. Finally, the thesis organization is presented in Section 1.3.

1

1.1. RESEARCH BACKGROUND

System identification (SYSID) has a long history in system and control engineering [32]. Its objective is to build the mathematical model (e.g., (partial) differential and difference dynamic relations) between the observed input and output data [1, 2, 8–10, 22, 31, 46, 56–58]. Whether the obtained model from SYSID can match the given dataset is decided by several aspects, including the data quality, identification criterion, optimization method, and model type selection. A detailed illustration of the influential factors for SYSID is in Fig. 1.1.

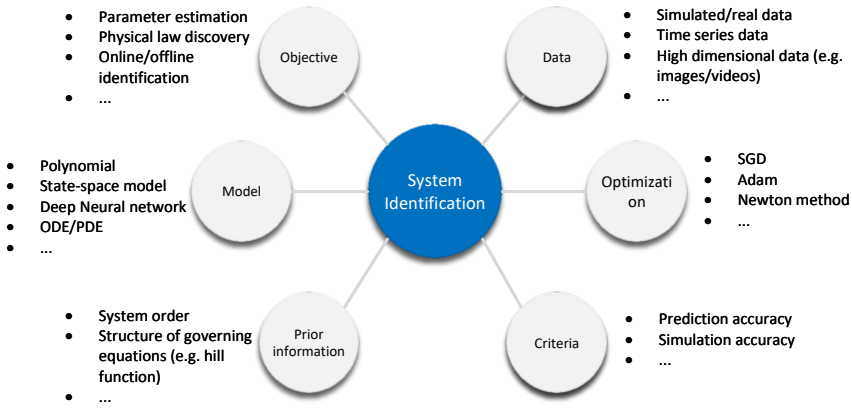


Figure 1.1: The paradigm for system identification.

Among these factors, the selection of model type is a fundamental one. The typical model types include linear model, state-space model, deep neural network model, etc. Different model types have their pros and cons and can adapt to diverse SYSID tasks. For example, a deep neural network model can provide a description for the health monitoring of a large chemical plant, which cannot be represented by mathematical expressions with a few terms. To reveal the underlying mechanism of the transcription and translation process between mRNAs and proteins, a white-box model taking account of the physical structure should be used to discover both the topology and parameters of a repressilator model. In recent years, with the prosperous development of deep learning [28], deep neural networks (DNNs) have attracted more and more attention in the SYSID community [2, 15, 33, 55]. Increasing attention has been put on DNNs for their inherent superiority, although they are normally regarded as prominent black-box models [33, 48, 49]. [21] stated that a feed-forward neural network with one hidden layer on a compact set has the universal approximation capabilities for any measurable function. Several works achieved competitive system identification results by using feed-forward neural networks [30] and recurrent neural networks (RNNs) [15, 45, 49] in dynamical systems. These achievements

motivate us to focus on the research of applying neural networks for system identification in this thesis.

However, DNNs also have disadvantages. First, the overfitting issue is a common problem for neural networks, which may reduce the model generalization ability. Second, DNNs are typically regarded as black-box models, which lack interpretability. The performance-oriented characteristic of a neural network makes it an appropriate choice for modelling tasks that aim to obtain a compact description of a system without considering the physical interpretability. However, every coin has two sides. This characteristic is also a disadvantage that limits the generalization of DNNs to white-box modelling tasks, which requires insight into the underlying phenomenon of the complex real-world system. In the next, we will briefly illustrate our solutions to address the above challenges.

For the overfitting issue, DNN compression techniques are promising solutions [14, 16, 29, 52]. As two typical compression methods, the ℓ_1 norm regularization [17, 50] and group ℓ_1 norm regularization [47, 54] can introduce the regularization term in the loss function to promote the sparsity of the network. However, our practical implementation showed that these two approaches are always non-sparse and time-consuming even when the hyper-parameters are extensively tuned. This motivates us to seek a Bayesian learning solution that can potentially result in sparser solutions and incorporate structural and nonstructural sparsity as priors.

The second issue is about building physical models from data, which can be classified as the symbolic regression (SR) problem. Previous works already propose several SR approaches, which can be divided into three categories, i.e., brute force methods [23, 51], genetic programming [6, 25], basis function methods [6, 20, 41, 43, 44]. These methods mainly suffer from the computational expense, sensitivity to initial conditions, and the nontrivial task of choosing appropriate basis functions. Recently, some works proposed to include a NN or NN-like model to benefit the discovery of the mathematical expressions. These approaches are called the network-based symbolic regression methods [39, 42]. The network-based SR method provides an encouraging solution that can be trained end-to-end through backpropagation with a fully developed deep learning framework (e.g., PyTorch and TensorFlow). It should be noted that the network-based SR method does not mean the mathematical expressions can be learned by simply training a classical neural network model, which lacks interpretability. In [42], a neural network is used to generate the mathematical operations in a reinforcement learning framework. And a modified NN-like structure is designed to denote the mathematical expressions in [39]. However, these previous network-based approaches cannot fit the constant term in the equation. Besides, they mainly use ℓ_1 norm regularization for training, which is inefficient, as illustrated before. In this thesis, we propose a novel type of network-based SR model termed as the Mathematical operation network (MathONet), which has a DNN-like *augmented hierarchical structure* composed of unary and binary operations. MathONet can be used to reformulate the governing equations, which are also characterized by basic mathematical operations, i.e., unary and binary ones. Besides, the governing equation discovery problem can also be treated as a sparse compression problem, which aims to search a subgraph from an over-parameterized MathONet graph with redundant mathematical operations. To this end, the compression of a DNN-like model MathONet is similar to the compression of a DNN model. Both the overfitting issue and physical modelling issue can be addressed

by sparse regression techniques. And the proposed Bayesian learning approach can also be used to learn and train the MathONet with structural and nonstructural sparsities.

In this thesis, we argue that the Bayesian approach is an effective sparse regression solution, where the redundant part of the model is pruned through employing sparsity-inducing priors on model parameters. The selection of prior represents the anticipation of the distribution of parameters and is an influential factor for the Bayesian method. The way to employ the prior on model parameters will affect both training and pruning results and should be adapted to different applications' specifics. Specifically, we mainly consider three kinds of priors. First, the prior is employed on a single weight. This can be used to prune the individual weight and estimate the parameters for a given model structure. In this thesis, we successfully employ the single prior to identify the closed form of Hill function for a repressilator model (see details in Chapter 3). Second, the prior is employed on a group of weights. This can be used to obtain structural sparsity. We successfully employ the group prior to select the input features for a dynamic system (see details in Chapter 4). Third, the priors are employed with both a single weight and a group of weights. Such priors are used to discover the governing equations of several dynamic systems, where both input feature selection and parameter estimation are required (see details in Chapter 5). To simplify the illustration, we use "single prior", "group-prior", and "fused prior" to represent these three ways of employing priors on model parameters.

The way to treat neural networks in a Bayesian manner is also known as Bayesian learning for neural networks [37, 40]. To approximate the posterior distribution, several approximate inference approaches have been proposed, *e.g.*, the Laplace approximation [37], Hamiltonian Monte Carlo [40], expectation propagation [18, 24], and variational inference [13, 19]. Among these methods, we adopt Laplace approximation, which is motivated by several reasons: 1) its easy implementation, especially using recent popular deep learning open-source software; 2) versatility of modern NN structures such as CNN and RNN as well as their modern variations; 3) close relationship between the computation of Hessian and network compression using Hessian metric [16, 29]; 4) acceleration effect to training convergence by second-order optimization algorithm [5] to which it is related; 5) scalability to deep neural networks. However, Laplace approximation requires the computation of the inverse Hessian of log-likelihood, which can be infeasible to compute for large networks. Although [5] has proposed an efficient Hessian approximation method for fully connected layers by calculating the diagonal blocks of the Hessian, it cannot be used for other parametric layers, such as the convolutional (Conv) layers and recurrent layers due to the indirect convolution operation and recurrent operation. This also motivates us to explore efficient Hessian calculation methods for the convolutional and recurrent layers in this thesis (see details in Chapter 2).

Furthermore, the objective of system identification is similar to training deep neural networks, which is to build a mathematical model uncovering the relationship between the input and output data. Therefore, the proposed Bayesian approaches, which aim to work for system identification tasks, should also have the potential to be implemented for the tasks related to deep learning. In this thesis, in addition to the typical SYSID topics, we also explore the generalization of the proposed approach to deep learning applications with high-dimensional datasets, including neural architecture search (NAS) and neural network compression (see details in Chapter 6).

1.2. RESEARCH CONTRIBUTIONS

The research contributions that we achieve in this thesis can be summarized as follow.

- **Propose efficient sparse Bayesian deep learning algorithms to address the overfitting issue of DNNs.** As a common problem in the training of DNNs, the overfitting issue is mainly caused by redundant structures. How to identify the model redundancy becomes the breakthrough to solve this problem. The ℓ_0 norm regularization of model parameters is a conceptually attractive approach by explicitly penalizing nonzero parameters with no further restrictions [34]. However, it is impractical to incorporate ℓ_0 norm regularization directly in the loss function due to its non-differentiability. To address this problem, ℓ_1 norm regularization [50] or group ℓ_1 norm regularization [47, 54] are usually adopted to replace the ℓ_0 norm regularization. They can shrink the actual value of the model parameters [50] and have already been widely used for topics such as deep neural network compression [7, 11, 17, 52], neural architecture search [12], multi-task regression [26], system identification [3, 7, 27], etc. However, the empirical evidence shows that ℓ_1 norm regularization methods cannot always effectively acquire sparse results, even when the hyper-parameters are extensively tuned. This motivates us to seek the Bayesian approach in this thesis, a more effective solution that can potentially result in sparser solutions. Specifically, we propose the sparse Bayesian deep learning (SBDL) approach, which uses the Laplace approximation to approximate the model evidence/marginal likelihood. An iterative regularized optimization procedure is derived as the identification algorithm. The effectiveness of the proposed Bayesian approach is demonstrated on various tasks, i.e., the parameter estimation and structure identification in Chapter 3, modelling on several linear and nonlinear SYSID benchmarks in Chapter 4, the governing equation discovery in Chapter 5.

- **The proposed Bayesian methods can be adapted to both black-box modelling and white-box modelling.** Typically, a model can be divided into three categories according to whether it takes a particular account of the physical structure, i.e., black-box model (e.g., DNN model), grey-box model, and white-box model (e.g., SR model). However, it is generally known that these model types have their respective optimization methods. For example, the DNN model can be trained by the stochastic gradient descent (SGD) method, where the gradients are computed via the back-propagation procedure. The SR model can be trained by genetic programming techniques, brute force, or sparse regression methods (See Section. 5.6 in Chapter 5 for details). It can be observed that these optimization methods are different and not compatible, which brings inconvenience for efficient training among different models. In this thesis, the DNNs (e.g., Fully-connected neural networks, convolutional neural networks, and recurrent neural networks) can be regarded as black-box models whose modelling task can be performed by the proposed SBDL approach. In addition, we design a novel model for the white-box modelling, termed mathematical operation network (MathONet), which has a DNN-like hierarchical structure consisting of basic mathematical operations, i.e., unary and binary ones (see more details in Section 5.2 of Chapter 5). With such a design, the white-box modelling problem can be treated as a subgraph search problem from an over-parameterized

MathONet graph. Since the structure of MathONet is similar to DNN, the proposed Bayesian approach, which works for DNNs, can also be used to train a MathONet. Thereafter, both the black-box and white-box models adopted in this thesis can be integrated with fully developed and advanced deep learning frameworks (e.g., PyTorch and TensorFlow), which can also accelerate the training process.

- **Efficient Hessian calculation methods for convolutional and recurrent layer are proposed to improve the calculation efficiency.** Bayesian approach has already proved to be an effective way to address the network compression problem [35], enabling a faster search through the parameter space [36]. However, a Bayesian approach always requires the updating of a probabilistic model, which means at least twice the parameters (e.g., mean and variance of each parameter) should be learned compared with the conventional methods. Therefore, higher computational efficiency is always a research direction worthy of effort. Simplifying related procedures or calculations within the Bayesian approach would be beneficial to the improvement of computational efficiency. Specifically, as we adopt the Laplace approximation method to update the posterior variance of parameters, the inverse Hessian of log-likelihood has to be calculated. However, as the dimension of Hessian is the square of the number of parameters, the calculation and storage of Hessian for large-scale neural networks are infeasible considering their millions of parameters [4, 5, 38]. To address this problem, efficient Hessian calculation techniques are required to be proposed for various networks, especially for convolutional neural networks and recurrent neural networks. In this thesis, inspired by a previous work related to the Hessian calculation method for the fully connected neural network [5], we present the efficient calculation/approximation methods of Hessian for both the convolutional layer and recurrent layer. By extracting the block-diagonal value of the Hessian, the proposed method can turn an intractable training/optimization procedure into a tractable one. The proposed Hessian computation can be calculated recursively along with the backpropagation process, potentially reducing the search time and being well extended to large-scale models. For example, compared to the Hessian calculation method proposed in [5], the required multiply-accumulate operation (MACs) is reduced from $n(2m^2+2n^2+4mn+3m-1)$ to $n(2+4m)$ with $W \in \mathbb{R}^{n\times m}$ (e.g., if $n = 100, m = 100$, the original method requires 107.97 MMACs compared with only 0.04 MMACs for the proposed method.). The detailed calculation procedures for a FC layer, Conv layer and recurrent layer are explained in Section 2.2.1, Section 2.2.2 and Section 2.2.3 of Chapter 2, respectively.

- **The proposed Bayesian methods can employ single prior, group prior and fused prior on model parameters for different applications.** In a Bayesian approach, the selection of priors is always imperfect [53]. In this thesis, we mainly consider three kinds of priors for different applications, i.e., single prior, group prior and fused prior. The selection of different priors will lead to changes in the update of the posterior distribution, loss function, and hyper-parameters (see Chapter. 2 for details). In the case of different modelling tasks, different priors are required to be employed on the model parameters. For example, a single prior would be proper for the parameter estimation with known structure (see Chapter. 3 for details) and the

determination of a CNN cell in NAS (see Chapter. 6 for details). The group prior is necessary for the input feature selection (see Chapter. 4 for details) and structural sparsity of a DNN compression (see Chapter. 6 for details). And the combination of a single prior and group prior can be used to identify governing equations for several dynamic systems (see details in Chapter 5). The corresponding Bayesian algorithms with these priors and their corresponding regularization update rules are presented in Section 2.1.2, Section 2.1.3 and Section 2.1.4 of Chapter 2, respectively. Besides,a practical calculation approach based on the Monte-Carlo integration method is also derived to quantify the uncertainty of the parameters and predictions (see details in Section 4.2.2 of Chapter 4). Furthermore, the dependencies between adjacent connections are also considered as the pruning criterium, which ensures a connected derived graph after pruning. The encoding strategy of dependencies by calculating the joint prior distribution is in Section 5.2.4 of Chapter 5.

- **The proposed Bayesian methods have the generalization ability for diverse topics in the deep learning research field with high-dimensional datasets.** In this thesis, the main objectives of SYSID tasks include building the mathematical model (e.g., (partial) differential and difference dynamic relations) between the observed input and output data, acquiring good and competitive prediction or simulation accuracy. These typical objectives are regarded as sparse regression problems and can be addressed with the proposed Bayesian approaches with diverse priors. In fact, some deep learning tasks, such as neural network compression and neural architecture search, can also be treated as sparse regression problems. However, we cannot directly apply the proposed Bayesian approaches to these deep learning tasks considering their peculiarities. For example, a search space should be designed in the first place for NAS, which requires the proposed Bayesian approach to adapt to the designed search space. And the dimensional consistency is essential to obtain the structured sparsity in a neural network compression task. In Chapter 6, we presented how to apply the proposed Bayesian approach for another two deep learning topics, i.e., neural network compression and neural architecture search. Specifically, the one-shot neural architecture search method is implemented on image classification tasks on CIFAR-10 and ImageNet datasets. The neural network compression is implemented for fully connected neural networks and convolutional neural networks.

1.3. THESIS ORGANIZATION

The remainder of the thesis is organized as follows.

Chapter 2 presents the proposed Bayesian approaches. The way to formulate the system identification in a Bayesian framework, the derivation procedures of the Laplace approximation, the update rules for hyper-parameters, and the algorithms enforcing single prior, group prior and fused prior on the model parameters are elaborated in detail. Besides, to address the challenge of Hessian calculation for deep neural networks, the efficient and recursive Hessian calculation methods for the DNN layers, including fully-connected, convolutional and recurrent layers are also presented.

Chapter 3 applys the sparse Bayesian deep learning algorithm with the single prior to

identify a repressilator model. The exact closed-form of the Hill function in a repressilator model is discovered by implementing the proposed Bayesian approach on a specially designed network structure. The designed network consists of two sub-networks. The first subnetwork is a linear network. The second subnetwork is a Fully-connected neural network that adopts the Hill function's general form as the activation function. Both the model structure and parameters can be identified accurately.

Chapter 4 applys the sparse Bayesian deep learning algorithm with the group prior to address several existing challenges of DNNs for system identification, including easily overfitting training datasets, input feature selection, and uncertainty quantification in estimated parameters and predictions. The effectiveness of the proposed method is demonstrated on several linear and nonlinear system identification benchmarks by achieving good and competitive simulation accuracy.

Chapter 5 applys the sparse Bayesian deep learning algorithm with the fused prior to the white-box modelling tasks. By presenting a new representation for governing equations with a DNN-like hierarchical network (i.e., MathONet), the developed Bayesian approach can be used to discover governing equations (ordinary differential equations (ODEs) or partial differential equations (PDEs)) from observations for several dynamic systems.

Chapter 6 presents the application of the proposed Bayesian approach to the neural architecture search (NAS) and neural network compression. For the NAS, the method can encode the dependency between connections and provide uncertainty as the pruning criteria, which are two typical issues associated with most one-shot NAS methods. The method is demonstrated by searching the architectures for the image classification tasks on CIFAR-10 and ImageNet datasets. For neural network compression, two classical DNN models (i.e., Fully-connected neural network and convolutional neural network) are compressed for their corresponding applications, including image classification on MNIST and Cifar10 datasets and the prediction of atrial fibrillation on a physiological dataset. It should be noted that the datasets used in this Chapter belong to high-dimensional datasets.

Chapter 7 concludes this thesis.

BIBLIOGRAPHY

[1] Roberto Battiti. "Using mutual information for selecting features in supervised neural net learning". In: *IEEE Transactions on neural networks* 5.4 (1994), pp. 537–550.

[2] Gerben Beintema, Roland Toth, and Maarten Schoukens. "Nonlinear state-space identification using deep encoder networks". In: *Learning for Dynamics and Control*. PMLR. 2021, pp. 241–250.

[3] M Bonin, V Seghezza, and Luigi Piroddi. "NARX model selection based on simulation error minimisation and LASSO". In: *IET control theory & applications* 4.7 (2010), pp. 1157–1168.

[4] Aleksandar Botev. "The Gauss-Newton matrix for Deep Learning models and its applications". PhD thesis. UCL (University College London), 2020.

[5] Aleksandar Botev, Hippolyt Ritter, and David Barber. "Practical Gauss-Newton Optimisation for Deep Learning". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 557–565.

[6] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proceedings of the national academy of sciences* 113.15 (2016), pp. 3932–3937.

[7] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proceedings of the National Academy of Sciences* 113.15 (2016), pp. 3932–3937. ISSN: 0027-8424. DOI: 10.1073/pnas.1517384113. eprint: https://www.pnas.org/content/113/15/3932.full.pdf.

[8] Giovanna Castellano and Anna Maria Fanelli. "Variable selection using neural-network models". In: *Neurocomputing* 31.1-4 (2000), pp. 1–13.

[9] Tianshi Chen et al. "System identification via sparse multiple kernel-based regularization using sequential convex optimization techniques". In: *IEEE Transactions on Automatic Control* 59.11 (2014), pp. 2933–2945.

[10] Alessandro Chiuso and Gianluigi Pillonetto. "A Bayesian approach to sparse dynamic network identification". In: *Automatica* 48.8 (2012), pp. 1553–1565. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2012.05.054. URL: http://www.sciencedirect.com/science/article/pii/S0005109812002270.

[11] Gari D Clifford et al. "AF classification from a short single lead ECG recording: The PhysioNet/computing in cardiology challenge 2017". In: *2017 Computing in Cardiology (CinC)*. IEEE. 2017, pp. 1–4.

[12] Ariel Gordon et al. “Morphnet: Fast & simple resource-constrained structure learning of deep networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1586–1595.

[13] Alex Graves. “Practical variational inference for neural networks”. In: *Advances in neural information processing systems*. 2011, pp. 2348–2356.

[14] Song Han, Huizi Mao, and William J. Dally. “Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding”. In: *International Conference on Learning Representations*. 2016.

[15] Joshua Hanson, Maxim Raginsky, and Eduardo Sontag. “Learning Recurrent Neural Net Models of Nonlinear Systems”. In: *Proceedings of Machine Learning Research* (2020).

[16] B. Hassibi, D. G. Stork, and G. J. Wolff. “Optimal Brain Surgeon and general network pruning”. In: *IEEE International Conference on Neural Networks*. Mar. 1993, 293–299 vol.1. DOI: 10.1109/ICNN.1993.298572.

[17] Yihui He, Xiangyu Zhang, and Jian Sun. “Channel pruning for accelerating very deep neural networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 1389–1397.

[18] José Miguel Hernández-Lobato and Ryan Adams. “Probabilistic backpropagation for scalable learning of bayesian neural networks”. In: *International Conference on Machine Learning*. PMLR. 2015, pp. 1861–1869.

[19] 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*. 1993, pp. 5–13.

[20] Moritz Hoffmann, Christoph Fröhner, and Frank Noé. “Reactive SINDy: Discovering governing reactions from concentration data”. In: *The Journal of chemical physics* 150.2 (2019), p. 025101.

[21] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: http://www.sciencedirect.com/science/article/pii/0893608089900208.

[22] William R Jacobs et al. “Sparse Bayesian nonlinear system identification using variational inference”. In: *IEEE Transactions on Automatic Control* 63.12 (2018), pp. 4172–4187.

[23] Ying Jin et al. “Bayesian symbolic regression”. In: *arXiv preprint arXiv:1910.08892* (2019).

[24] Pasi Jylänki, Aapo Nummenmaa, and Aki Vehtari. “Expectation propagation for neural networks with sparsity-promoting priors”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1849–1901.

[25] Samuel Kim et al. “Integration of neural network-based symbolic regression in deep learning for scientific discovery”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[26] Seyoung Kim and Eric P Xing. "Tree-guided group lasso for multi-task regression with structured sparsity". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. 2010, pp. 543–550.

[27] Sunil L Kukreja, Johan Löfberg, and Martin J Brenner. "A least absolute shrinkage and selection operator (LASSO) for nonlinear system identification". In: *IFAC proceedings volumes* 39.1 (2006), pp. 814–819.

[28] Yann Lecun, Yoshua Bengio, and Geoffrey E Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[29] Yann LeCun, John S Denker, and Sara A Solla. "Optimal brain damage". In: *Advances in neural information processing systems*. 1990, pp. 598–605.

[30] Moshe Leshno et al. "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function". In: *Neural Networks* 6.6 (1993), pp. 861–867. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(05)80131-5. URL: https://www.sciencedirect.com/science/article/pii/S0893608005801315.

[31] Martin Lindfors and Tianshi Chen. "Regularized LTI system identification in the presence of outliers: A variational EM approach". In: *Automatica* 121 (2020), p. 109152. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2020.109152. URL: https://www.sciencedirect.com/science/article/pii/S0005109820303502.

[32] Lennart Ljung. "System identification". In: *Wiley encyclopedia of electrical and electronics engineering* (1999), pp. 1–19.

[33] Lennart Ljung et al. "Deep Learning and System Identification". In: *IFAC-PapersOnLine* 53.2 (2020). 21th IFAC World Congress, pp. 1175–1181. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2020.12.1329. URL: https://www.sciencedirect.com/science/article/pii/S2405896320317353.

[34] C Louizos, M Welling, and DP Kingma. "Learning sparse neural networks through L0 regularization." In: *Sith International Conference on Learning Representations, Vancouver Canada, Monday April 30-Thursday May 03, 2018*. 2018.

[35] Christos Louizos, Karen Ullrich, and Max Welling. "Bayesian compression for deep learning". In: *arXiv preprint arXiv:1705.08665* (2017).

[36] Xingchen Ma et al. "A bayesian optimization framework for neural network compression". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 10274–10283.

[37] David JC MacKay. "A practical Bayesian framework for backpropagation networks". In: *Neural computation* 4.3 (1992), pp. 448–472.

[38] James Martens and Roger Grosse. "Optimizing neural networks with kronecker-factored approximate curvature". In: *International conference on machine learning*. PMLR. 2015, pp. 2408–2417.

[39] Georg Martius and Christoph H Lampert. "Extrapolation and learning equations". In: *arXiv preprint arXiv:1610.02995* (2016).

[40] Radford M Neal. “Bayesian learning for neural networks”. In: (1995).

[41] Wei Pan et al. “Reconstruction of arbitrary biochemical reaction networks: A compressive sensing approach”. In: *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE. 2012, pp. 2334–2339.

[42] Brenden K Petersen et al. “Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients”. In: *arXiv preprint arXiv:1912.04871* (2019).

[43] Markus Quade et al. “Sparse identification of nonlinear dynamics for rapid model recovery”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 28.6 (2018), p. 063116.

[44] Samuel H Rudy et al. “Data-driven discovery of partial differential equations”. In: *Science Advances* 3.4 (2017), e1602614.

[45] Anton Maximilian Schäfer and Hans Georg Zimmermann. “Recurrent Neural Networks Are Universal Approximators”. In: *Proceedings of the 16th International Conference on Artificial Neural Networks - Volume Part I*. ICANN’06. Athens, Greece: Springer-Verlag, 2006, pp. 632–640. ISBN: 3540386254. DOI: 10.1007/11840817_66. URL: https://doi.org/10.1007/11840817_66.

[46] Maarten Schoukens and Fritjof Griesing Scheiwe. “Modeling nonlinear systems using a volterra feedback model”. In: *Workshop on nonlinear system identification benchmarks*. 2016.

[47] Noah Simon et al. “A sparse-group lasso”. In: *Journal of computational and graphical statistics* 22.2 (2013), pp. 231–245.

[48] Jonas Sjöberg, Håkan Hjalmarsson, and Lennart Ljung. “Neural networks in system identification”. In: *IFAC Proceedings Volumes* 27.8 (1994), pp. 359–382.

[49] Eduardo D Sontag. “A learning result for continuous-time recurrent neural networks”. In: *Systems & Control Letters* 34.3 (1998), pp. 151–158.

[50] Robert Tibshirani. “Regression shrinkage and selection via the lasso”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288.

[51] Silviu-Marian Udrescu and Max Tegmark. “AI Feynman: A physics-inspired method for symbolic regression”. In: *Science Advances* 6.16 (2020), eaay2631.

[52] Wei Wen et al. “Learning structured sparsity in deep neural networks”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 2074–2082.

[53] Andrew Gordon Wilson. *The Case for Bayesian Deep Learning*. 2020. arXiv: 2001.10995 [cs.LG].

[54] Ming Yuan and Yi Lin. “Model selection and estimation in regression with grouped variables”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.1 (2006), pp. 49–67.

[55] Luca Zancato and Alessandro Chiuso. “A novel Deep Neural Network architecture for non-linear system identification”. In: *arXiv preprint arXiv:2106.03078* (2021).

[56] Wenxiao Zhao, Erik Weyer, and George Yin. "A General Framework for Nonparametric Identification of Nonlinear Stochastic Systems". In: *IEEE Transactions on Automatic Control* 66.6 (2021), pp. 2449–2464. DOI: 10.1109/TAC.2020.3007569.

[57] Wenxiao Zhao, George Yin, and Er-Wei Bai. "Sparse system identification for stochastic systems with general observation sequences". In: *Automatica* 121 (2020), p. 109162. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2020.109162. URL: https://www.sciencedirect.com/science/article/pii/S0005109820303605.

[58] Wei-Xing Zheng and Chun-Bo Feng. "Identification of stochastic time lag systems in the presence of colored noise". In: *Automatica* 26.4 (1990), pp. 769–779. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(90)90052-J. URL: https://www.sciencedirect.com/science/article/pii/000510989090052J.

2

METHOD

This chapter illustrates two main theoretical contributions of this thesis. First, three sparse Bayesian deep learning (SBDL) algorithms are presented that can enforce diverse priors (e.g., single prior, group prior and fused prior) on model parameters and obtain different regularization effect (non-structural, structural and combined regularization). The regularization update rules for the hyper-parameters and loss function with different priors are also concluded. Second, we derived efficient and recursive Hessian calculation methods for different DNN layers, including fully-connected, convolutional and recurrent layers. By extracting the block-diagonal value of the Hessian, the proposed method can turn an intractable training/optimization procedure into a tractable one. For example, compared to a previous Hessian calculation approach, the required multiply accumulate operation (MACs) with the proposed Hessian calculation method could be reduced from $n(2m^2+2n^2+4mn+3m-1)$ *to* $n(2+4m)$ *with* $W \in \mathbb{R}^{n\times m}$ *(e.g., if* $n=100, m=100$*, the original method requires* 107.97 *MMACs compared with only* 0.04 *MMACs for the approximate method.).*

In this Chapter, we formulate the DNN identification problem in a Bayesian framework in Section. 2.1.1. Three corresponding Bayesian deep learning algorithms with these priors are presented in Section. 2.1.2, Section. 2.1.3, and Section. 2.1.4, respectively. The efficient and recursive Hessian calculation methods for fully-connected, convolutional and recurrent layers are in Section. 2.2.

Parts of this chapter have been published in the International Conference on Machine Learning (2019) [30] and arXiv preprint arXiv:2107.12910(2021) [31].

2.1. SPARSE BAYESIAN DEEP LEARNING ALGORITHMS

2.1.1. BAYESIAN NEURAL NETWORK

In this thesis, we mainly consider the system identification problems using DNN models or the designed DNN-like models. The typical DNN models include fully-connected neural networks, convolutional neural networks and recurrent neural networks. To simplify the explanation, we use a fully-connected neural network to explain how to formulate the system identification problem from a Bayesian viewpoint.

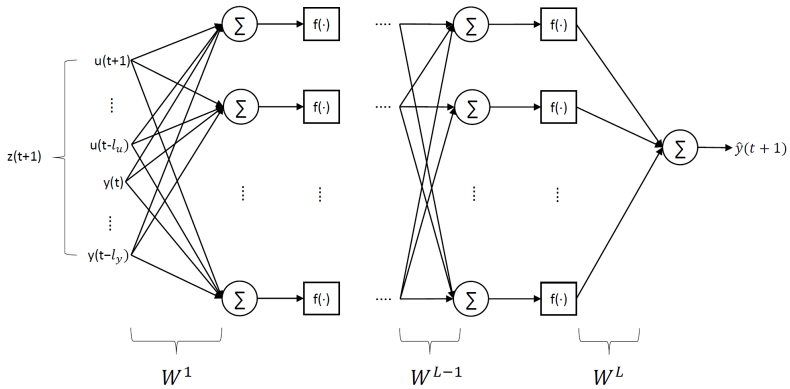


Figure 2.1: Fully-connected neural networks with L layers.

Given the network model as shown in Fig. 2.1, its structure is the map generated by training the network by $\texttt{Net}(W,z)$, where $W=\{W^l\}, l=[1,2,\cdots,L]$ represents the set of weights in the network and z represents the input regressors of size $1\times(l_y+l_u+1)$. $f(\cdot)$ stands for the activation function. The prediction model is defined as:

$$\hat{y}(t+1)=\texttt{Net}(W,z(t+1))+\epsilon \qquad \epsilon\sim\mathcal{N}(0,\sigma^2) \tag{2.1}$$

The noise term ϵ is assumed normally distributed with mean zero and known variance σ^2. The input regressors of this model are defined as a combination of lagged elements of the system inputs u and outputs y. The input lag is denoted as l_u, and output lag l_y, resulting in the expression $z(t+1)=[u(t+1),u(t),\cdots,u(t-l_u),y(t),y(t-1),\cdots,y(t-l_y)]^\top$.

Given a dataset $\mathscr{D}$ of T observations within a Bayesian framework, the posterior estimation for network weights W is given by Bayes' rule:

$$p(W|\mathscr{D},\mathscr{H})=\frac{p(\mathscr{D}|W,\mathscr{H})\,p(W,\mathscr{H})}{p(\mathscr{D}|\mathscr{H})} \tag{2.2}$$

$p(\mathscr{D}|W,\mathscr{H})$ designates the likelihood function, $p(W,\mathscr{H})$ the prior over the weights W and $p(\mathscr{D}|\mathscr{H})$ is the evidence of the hypothesis $\mathscr{H}$ given $\mathscr{D}$. The hypothesis generally incorporates model and inference assumptions. For simplicity of notations, the hypothesis term

is dropped in the rest of the paper. With variance σ^2, the likelihood function is given by:

$$p(\mathscr{D}|W,\sigma^2) = \prod_{t=1}^{\top} \mathcal{N}(y(t)|\texttt{Net}(W,z(t+1)),\sigma^2)$$
$$= (2\pi\sigma^2)^{-\frac{T}{2}} \exp\{-\mathbf{E}(W,\sigma^2)\} \tag{2.3}$$

Assume the likelihood function belongs to a Gaussian distribution, $\mathbf{E}(W,\sigma^2)$ denotes an energy loss function of the neural network in the form of a sum of squared errors $\mathbf{E}(W,\sigma^2) = \frac{1}{2\sigma^2}\sum_{t=1}^{\top}(y(t) - \text{Net}(W,z(t+1)))^2$.

The prior probability $p(W)$ takes a Gaussian relaxed variational form $p(W) \geq p(W,\psi) = \mathcal{N}(W|0,\Psi)\ \phi(\psi)$, where $\phi(\psi)$ represents the hyperprior probability of $\psi \triangleq [\psi_1,\cdots,\psi_n,\cdots,\psi_N]$ and $\Psi \triangleq \text{diag}(\psi)$. N denotes the number of unknown parameters. For instance, suppose $W^l \in \mathbb{R}^{n^{l-1}\times n^l}$, $N = \sum_{l=1}^{L} n^{l-1} \times n^l$ where $n^0 = l_y + l_u + 1$. With the principle of minimizing misaligned probability mass [22], the hyper-parameter ψ can be obtained by:

$$\hat{\psi} = \underset{\psi\geq 0}{\text{argmin}} \int p(\mathscr{D}|W,\sigma^2)|p(W) - p(W,\psi)|\ dW \tag{2.4}$$

$$= \underset{\psi\geq 0}{\text{argmax}} \int p(\mathscr{D}|W,\sigma^2)p(W,\psi)\ dW \tag{2.5}$$

The resulting problem is known as a type II maximum likelihood [26]. The intractable integral can be approximated by approximate inference methods. We refer specifically to the Laplace approximation.

It should be noted that we employ different priors to address diverse SYSID tasks in this thesis. Specifically, three kinds of priors are included, i.e., the single prior for non-structural sparsity used for parameter estimation, the group prior for structural sparsity used for input feature selection and the fused prior (single prior & group prior) for combined (non-structural & structural) sparsity used for governing equation identification. Fig. 2.2 shows how to employ priors on model parameters. With the Laplace approximation method, the detailed derivation about the hyper-parameters under different priors is also different and will be elaborated as follows.

2.1.2. SPARSE BAYESIAN DEEP LEARNING ALGORITHM WITH SINGLE PRIOR

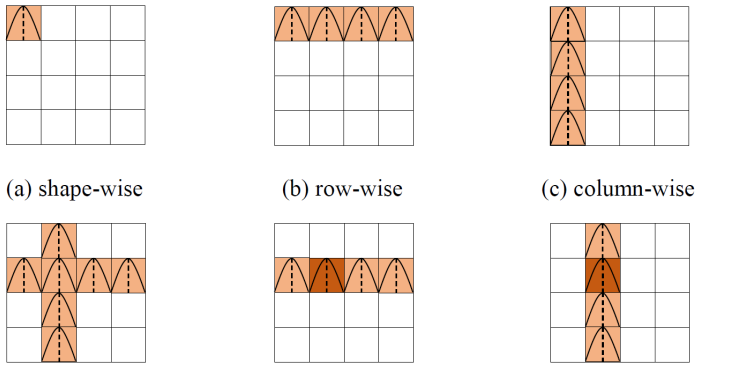
For the single prior, the independent prior distribution will be enforced on each parameter as shown in Fig. 2.2(a). Taking the weight matrix W^l of layer l as the optimization target, we assume that the prior probability for $p(W^l)$ is a Gaussian relaxed variational form

$$p(W^l) \geq p(W^l,\psi^l) = \mathcal{N}(W^l|0,\psi^l)\ \phi(\psi^l) \tag{2.6}$$

where $\phi(\psi^l)$ represents the hyperprior probability of $\psi^l \triangleq [\psi^l_1,\cdots,\psi^l_m,\cdots,\psi^l_M]$. M denotes the number of unknown parameters. ψ^l_m is independent from each other.

THE LAPLACE APPROXIMATION

In this section, a detailed mathematical description of the adopted Laplace approximation method is given. To compute the intractable integral in Eq. (2.5), the Laplace approximation is taken on the likelihood function as Eq. (2.3). The energy function $\mathbf{E}(W^l,\sigma^2)$ is the



(a) shape-wise (b) row-wise (c) column-wise

(d) row & column-wise (e) shape & column-wise (f) shape & column-wise

Figure 2.2: Priors for structured sparsity of weight matrices. (a): the regularization is enforced on a single parameter (coloured square), which is called shape-wise regularization; (b)-(d): the regularization is enforced on a group of parameters (coloured squares). The group can be a row, column, row & column of a 2D matrix, which are called row-wise, column-wise, row & column-wise regularization, respectively; (e)-(f): a fused prior (a single prior & group prior) is enforced on model parameters, which are marked as dark brown.

loss of the network given the data $\mathscr{D}$. It is given by:

$$\mathbf{E}(W^l, \sigma^2) = \frac{1}{2\sigma^2} \sum_{t=1}^{\top} (y(t) - \mathrm{Net}(W^l, z(t)))^2 \tag{2.7}$$

The expression $\mathtt{Net}(\cdot)$ in Eq. (2.7) is the resulting network non-linear map. To compute the intractable integral for the evidence, the energy function can be expanded to a second-order Taylor series expansion around W^{l*}.

$$\mathbf{E}(W^{l*}, \sigma^2) \approx \mathbf{E}(W^{l*}, \sigma^2) + (W^l - W^{l*})^\top \mathbf{g}(W^{l*}, \sigma^2) + \frac{1}{2}(W^l - W^{l*})^\top \mathbf{H}(W^{l*}, \sigma^2)(W^l - W^{l*}) \tag{2.8}$$

where $\mathbf{g}(W^{l*}, \sigma^2) = \nabla \mathbf{E}(W^l, \sigma^2)|_{W^{l*}}$ and $\mathbf{H}^l(W^{l*}, \sigma^2) = \nabla\nabla \mathbf{E}(W^l, \sigma^2)|_{W^{l*}}$. To save space, we use $\mathbf{g}^l$, $\mathbf{H}^l$ and $\mathbf{E}$ to denote $\mathbf{g}(W^{l*}, \sigma^2)$, $\mathbf{H}^l(W^{l*}, \sigma^2)$ and $\mathbf{E}(W^{l*}, \sigma^2)$ in the following, respectively. The quadratic expression is also adopted among Trust-Region methods. A region is defined around the current iterate connection weights W^l, and the expansion in Eq. (2.8) is considered a reasonable local representation of the loss function [21]. With this

expansion, the likelihood function becomes:

$$
\begin{aligned}
& p(\mathscr{D}|W^l,\sigma^2) \\
\approx & (2\pi\sigma^2)^{-\frac{T}{2}} \exp\left\{-\left(\frac{1}{2}(W^l - W^{l^*})^\top \mathbf{H}^l (W^l - W^{l^*}) + (W^l - W^{l^*})^\top \mathbf{g}^l + \mathbf{E}\right)\right\} && (2.9) \\
= & (2\pi\sigma^2)^{-\frac{T}{2}} \exp\left\{-\left(\frac{1}{2}(W^l)^\top \mathbf{H}^l W^l + (W^l)^\top \left(\mathbf{g}^l - \mathbf{H} W^{l^*}\right)\right)\right\} \\
& \cdot \exp\left\{-\left(\frac{1}{2}(W^{l^*})^\top \mathbf{H}^l W^{l^*} - (W^{l^*})^\top \mathbf{g}^l + \mathbf{E}\right)\right\} && (2.10) \\
= & \mathbf{A} \cdot \exp\left\{-\left(\frac{1}{2}(W^l)^\top \mathbf{H}^l W^l + (W^l)^\top \hat{\mathbf{g}}^l\right)\right\} && (2.11)
\end{aligned}
$$

with,

$$
\begin{aligned}
\hat{\mathbf{g}}^l &= \mathbf{g}^l - \mathbf{H} W^{l^*} && (2.12) \\
\mathbf{A} &= (2\pi\sigma^2)^{-\frac{T}{2}} \cdot \exp\left\{-\left(\frac{1}{2}(W^{l^*})^\top \mathbf{H}^l W^{l^*} - (W^{l^*})^\top \mathbf{g}^l + \mathbf{E}\right)\right\} && (2.13)
\end{aligned}
$$

A Gaussian form can be easily recuperated from Eq. (2.11) by completing the square in the exponent. Before that, we define the following quantities: $\mathbf{B} = \exp\left\{\frac{1}{2}(\hat{g}^l)^\top \mathbf{H}^l \hat{\mathbf{g}}^l\right\}, \mathbf{C} = (2\pi)^{\frac{T}{2}} |\mathbf{H}|^{\frac{1}{2}}$

$$
\begin{aligned}
& p(\mathscr{D}|W^l,\sigma^2) \\
\approx & \mathbf{A} \cdot \exp\left\{-\left(\frac{1}{2}(W^l)^\top \mathbf{H}^l W^l + (W^l)^\top \hat{\mathbf{g}}^l\right)\right\} \cdot \exp\left\{\frac{1}{2}(\hat{\mathbf{g}}^\mathbf{l})^\top \mathbf{H}^l \hat{\mathbf{g}}^l - \frac{1}{2}(\hat{\mathbf{g}}^\mathbf{l})^\top \mathbf{H}^l \hat{\mathbf{g}}^l\right\} && (2.14) \\
= & \mathbf{A} \cdot \mathbf{B} \cdot \exp\left\{-\left(\frac{1}{2}(W^l)^\top \mathbf{H}^l W^l + (W^l)^\top \hat{\mathbf{g}}^l + \frac{1}{2}(\hat{\mathbf{g}}^\mathbf{l})^\top \mathbf{H}^l \hat{\mathbf{g}}^l\right)\right\} && (2.15) \\
= & \mathbf{A} \cdot \mathbf{B} \cdot \mathbf{C} \cdot \mathscr{N}(W^l | \hat{W}^l, (\mathbf{H}^l)^{-1}) && (2.16)
\end{aligned}
$$

where $\hat{W}^l = -(\mathbf{H}^l)^{-1} \hat{\mathbf{g}}^l$.

Given such a Gaussian likelihood and a Gaussian prior, the posterior is also Gaussian $\mathscr{N}(\mu_{W^l}, \Sigma_{W^l})$ by the effect of the conjugacy rule:

$$
\mu_{W^l} = \left[\mathbf{H}^l + (\psi^l)^{-1}\right]^{-1} \hat{\mathbf{g}}^l \qquad \Sigma_{W^l} = \left[\mathbf{H}^l + (\psi^l)^{-1}\right]^{-1} \tag{2.17}
$$

EVIDENCE MAXIMIZATION

In this section, we will derive the objective function, which is achieved by maximizing the evidence and can be used to update the model parameters. The evidence in Eq. (2.5) attempts to find the volume of the product $p(\mathscr{D}|W^l,\sigma^2)p(W^l,\psi^l)$, which is Gaussian and proportional to the posterior. Thus, one can approximate the evidence as to the volume around the most probable value (here the posterior μ_{W^l}).

$$
\begin{aligned}
\hat{\psi^l} &= \underset{\psi^l \geq 0}{\operatorname{argmax}} \int p(\mathscr{D}|W^l,\sigma^2) p(W^l|\psi^l) p(\psi^l) dW^l && (2.18) \\
&\approx \underset{\psi^l \geq 0}{\operatorname{argmax}} \underbrace{p(\mathscr{D}|\mu_{W^l},\sigma^2)}_{\text{Best Fit Likelihood}} \underbrace{p(\mu_{W^l}|\psi^l)|\Sigma_{W^l}|^{\frac{1}{2}}}_{\text{Occam Factor}} && (2.19)
\end{aligned}
$$

In David Mackay's words, the evidence is approximated by the product of the data likelihood given the most probable weights and the Occam factor [15]. It can also be interpreted as a Riemann approximation of the evidence, where the best-fit likelihood represents the peak of the evidence. And the Occam's factor is the Gaussian curvature around the peak [8].

By realizing that the posterior mean μ_{W^l} maximizes $p(\mathscr{D}|W^l,\sigma^2)p(W^l|\psi^l)$, the likelihood and prior is replaced by their expressions:

$$\int p(\mathscr{D}|W^l,\sigma^2)p(W^l|\psi^l)p(\psi^l)\,dW^l \tag{2.20}$$

$$= \int \mathbf{A}\cdot\exp\Big\{-\Big(\frac{1}{2}(W^l)^\top \mathbf{H}^l W^l + (W^l)^\top \hat{\mathbf{g}}^l\Big)\Big\}\cdot\mathscr{N}(W^l|0,\psi^l)\cdot\phi(\psi^l)\,dW^l \tag{2.21}$$

$$= \frac{\mathbf{A}}{(2\pi)^{T/2}|\psi^l|^{\frac{1}{2}}}\cdot\int \exp\Big\{-\Big(\frac{1}{2}(W^l)^\top \mathbf{H}^l W^l + (W^l)^\top \hat{\mathbf{g}}^l\Big)\Big\}$$

$$\cdot\exp\Big\{-\Big(\frac{1}{2}(W^l)^\top(\psi^{\mathbf{l}})^{-\mathbf{1}}W^l\Big)\Big\}\,dW^l\cdot\prod_{i=1}^{M}\phi(\psi_i^l) \tag{2.22}$$

$$= \frac{\mathbf{A}}{(2\pi)^{T/2}|\psi^l|^{\frac{1}{2}}}\cdot\int \exp\Big\{-E(W^l,\sigma^2)\Big\}\,dW^l\cdot\prod_{i=1}^{M}\phi(\psi_i^l) \tag{2.23}$$

where,

$$E(W^l,\sigma^2) = \frac{1}{2}(W^l)^\top \mathbf{H}(W^{l^*},\sigma^2)W^l + (W^l)^\top\hat{\mathbf{g}}^l + \frac{1}{2}(W^l)^\top(\psi^{\mathbf{l}})^{-\mathbf{1}}W^l \tag{2.24}$$

The integral in Eq. (2.20) is the integral of the product $p(\mathscr{D}|W^l,\sigma^2)p(W^l|\psi^l)$, which is proportional to the posterior $p(W^l|\mathscr{D},\psi^l)$. In most applications, the posterior peaks with respect to the prior, and the evidence can be approximated by the posterior volume. This approximation is analogous to using the Laplace approximation of the posterior in David MacKay's Bayesian framework [15].

$$\int p(\mathscr{D}|W^l,\sigma^2)p(W^l|\psi^l)\,dW^l \approx p(\mathscr{D}|\mu_{W^l},\sigma^2)p(\mu_{W^l}|\psi^l)\cdot|\Sigma_{W^l}|^{\frac{1}{2}}\cdot(2\pi)^{T/2} \tag{2.25}$$

$$\iff \int \exp\Big\{-E(W^l,\sigma^2)\Big\}\,dW^l \approx \exp\Big\{-E(\mu_{W^l},\sigma^2)\Big\}\cdot|\Sigma_{W^l}|^{\frac{1}{2}}\cdot(2\pi)^{T/2} \tag{2.26}$$

where,

$$E(\mu_{W^l},\sigma^2) = \frac{1}{2}(\mu_{W^l})^\top \mathbf{H}(W^{l^*},\sigma^2)\mu_{W^l} + (\mu_{W^l})^\top\hat{\mathbf{g}}^l + \frac{1}{2}(\mu_{W^l})^\top(\psi^{\mathbf{l}})^{-\mathbf{1}}\mu_{W^l} \tag{2.27}$$

$$= \min_{W^l}\frac{1}{2}(W^l)^\top \mathbf{H}(W^{l^*},\sigma^2)W^l + (W^l)^\top\hat{\mathbf{g}}^l + \frac{1}{2}(W^l)^\top(\psi^{\mathbf{l}})^{-\mathbf{1}}W^l \tag{2.28}$$

Hence the maximization of the evidence becomes the maximization in Eq. (2.29):

$$\psi^l = \underset{\psi^l>0}{\operatorname{argmax}}\frac{\mathbf{A}}{(2\pi)^{T/2}|\psi^l|^{\frac{1}{2}}}\cdot\exp\big\{-E(\mu_{W^l},\sigma^2)\big\}\cdot|\Sigma_{W^l}|^{\frac{1}{2}}\cdot\prod_{i=1}^{M}\phi(\psi_i^l) \tag{2.29}$$

By applying a $-2\log(\cdot)$ operation and using Eq. (2.28), one obtains

$$\psi^l = \underset{\psi^l>0}{\operatorname{argmin}} -2\log\left[\frac{\mathbf{A}}{(2\pi)^{T/2}|\psi^l|^{\frac{1}{2}}}\cdot\exp\left\{-E(\mu_{W^l},\sigma^2)\right\}\cdot|\Sigma_{W^l}|^{\frac{1}{2}}\cdot\prod_{i=1}^{M}\phi(\psi_i^l)\right] \tag{2.30}$$

$$= \underset{\psi^l>0}{\operatorname{argmin}} -2\log(\mathbf{A}) + E(\mu_{W^l},\sigma^2) + \log|\psi^l| - \log|\Sigma_{W^l}| - 2\prod_{i=1}^{M}\log\Big(\phi(\psi_i^l)\Big) \tag{2.31}$$

$$W^l,\psi^l = \underset{W^l,\psi^l>0}{\operatorname{argmin}} \frac{1}{2}(W^l)^\top\mathbf{H}(W^{l^*},\sigma^2)W^l + (W^l)^\top\hat{\mathbf{g}}^l + \frac{1}{2}(W^l)^\top(\psi^{\mathbf{l}})^{-\mathbf{1}}W^l + \log|\psi^l|$$
$$+\log|\mathbf{H}(W^{l^*},\sigma^2) + (\psi^l)^{-1}| - 2\log(\mathbf{A}) - 2\prod_{i=1}^{M}\log\Big(\phi(\psi_i^l)\Big)$$

Since the hyperprior $\phi(\psi^l)$ is a non-informative hyper-prior, the final objective function is given by:

$$\mathscr{L}(W^l,\psi^l,\sigma^2) = (W^l)^\top\mathbf{H}^lW^l + 2(W^l)^\top\hat{\mathbf{g}}^l + (W^l)^\top(\psi^l)^{-1}W^l + \log|\psi^l| + \log|\mathbf{H}^l + (\psi^l)^{-1}|$$
$$- T\log\left(2\pi\sigma^2\right) \tag{2.32}$$

This objective function will be revisited in the next section to update the parameters W^l,ψ^l.

REGULARIZATION UPDATE RULES

In this section, we will derive how to update the parameters W^l,ψ^l using a convex-concave procedure (CCCP). This is also a contribution of the thesis. As an optimization objective, Eq. (2.32) has two parameters W^l and ψ^l which are required 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. The objective function in Eq. (2.32) can be seen as a sum of convex u and concave v functions in ψ^l shown in Eq. (2.33)- (2.34).

$$u(W^l,\psi^l) = (W^l)^\top\mathbf{H}^lW^l + 2(W^l)^\top\hat{\mathbf{g}}^l + (W^l)^\top(\psi^l)^{-1}W^l \tag{2.33}$$

$$v(\psi^l) = \log|\psi^l| + \log|\mathbf{H}^l + (\psi^l)^{-1}| \tag{2.34}$$

$(W^l)^\top(\psi^l)^{-1}W^l$ is positive definite, since $\psi^l > 0$, thus u is convex in ψ^l. v can be reformulated as a log-determinant of an affine function of ψ^l. By using the Schur complement determinant identities,

$$|\psi^l||\mathbf{H}^l + (\psi^l)^{-1}| = \begin{vmatrix}\mathbf{H}^l & \\ & -\psi^l\end{vmatrix} = |\mathbf{H}^l||(\mathbf{H}^l)^{-1} + \psi^l| \tag{2.35}$$

and taking the log of Eq. (2.35),

$$\log|\psi^l| + \log|\mathbf{H}^l + (\psi^l)^{-1}| = \log|\mathbf{H}^l| + \log|(\mathbf{H}^l)^{-1} + \psi^l| \tag{2.36}$$

Since the log-determinant is concave (see page 74 in [4]), v is concave in ψ^l (Eq. (2.36)). The minimization problem can therefore be reformulated as a convex-concave procedure

(CCCP) [28]. W^l and ψ^l are obtained with iterative minimization of Eq. (2.37)-(2.38).

$$W^l(k+1) = \underset{W^l(k)}{\operatorname{argmin}}\, u\big(W^l(k), \psi^l(k)\big) \tag{2.37}$$

$$\psi^l(k+1) = \underset{\psi^l \geq 0}{\operatorname{argmin}}\, u\big(W^l(k+1), \psi^l(k)\big) + \alpha^l(k) \cdot \psi^l(k) \tag{2.38}$$

where $\alpha^l(k) = \nabla^l_\psi v\big(\psi^l(k)\big)^\top$ is the gradient of v evaluated at the current iterate $\psi^l(k)$. Using the chain rule, its analytic form is given by:

$$\begin{aligned}\alpha^l(k) =& \nabla^l_\psi \Big(\log|\psi^l(k)| + \log|\mathbf{H}^l + \Big(\psi^l(k)\Big)^{-1}| \Big)\Big|_{\psi^l=\psi^l(k)} \qquad (2.39)\\ =& -\operatorname{diag}\Big(\Big(\psi^l(k)\Big)^{-1}\Big) \circ \operatorname{diag}\Big(\big(\mathbf{H}^l + \Big(\psi^l(k)\Big)^{-1}\big)^{-1}\Big) \\ & \circ \operatorname{diag}\Big(\Big(\psi^l(k)\Big)^{-1}\Big) + \operatorname{diag}\Big(\Big(\psi^l(k)\Big)^{-1}\Big) \qquad (2.40)\end{aligned}$$

$\circ$ is the point-wise Hadamard product. Since ψ^l is a diagonal matrix, Eq. (2.38) can be expressed per connection independently. With $\Sigma_{W^l}(k)$ the connection weight posterior variance, the analytical form for α^l is given by

$$\Sigma_{W^l(k)} = \big(\mathbf{H}^l(k) + \psi^l(k)^{-1}\big)^{-1} \tag{2.41}$$

$$\alpha^l_i(k) = -\frac{\Sigma_{W^l_i(k)}}{\big(\psi^l_i(k)\big)^2} + \frac{1}{\psi^l_i(k)} \tag{2.42}$$

The optimization step in Eq. (2.38) for ψ^l_i becomes

$$\psi^l_i(k+1) = \underset{\psi^l \geq 0}{\operatorname{argmin}} \frac{\big(W^l_i(k+1)\big)^2}{\psi^l_i(k)} + \alpha^l_i(k) \cdot \psi^l(k) \tag{2.43}$$

By noting that

$$\frac{(W^l_i)^2}{\psi^l_i} + \alpha^l_i \cdot \psi^l_i \geq 2\Big|\sqrt{\alpha^l_i} \cdot W_i\Big| \tag{2.44}$$

The analytical solution is given by

$$\psi^l_i(k+1) = \frac{|W^l_i(k+1)|}{\omega^l_i(k)} \tag{2.45}$$

$$\omega^l_i(k) = \sqrt{\alpha^l_i(k)} \tag{2.46}$$

For the second part, finding W^l can be done with stochastic gradient descent on Eq. (2.37), which can be reformulated as a regularized neural network loss function.

$$\begin{aligned}W^l(k+1) &= \underset{W^l}{\operatorname{argmin}} \Big(W^l(k)\Big)^\top \mathbf{H}^l W^l(k) + 2\Big(W^l(k)\Big)^\top \hat{\mathbf{g}}^l + \sum_{i=1}^{M} ||\omega^l_i \cdot W^l_i(k)||_{\ell_1} \\ &\approx \underset{W^l}{\operatorname{argmin}}\, \mathbf{E}(\cdot) + \frac{1}{2}\sum_{i=1}^{M} ||\omega^l_i \cdot W^l_i(k)||_{\ell_1} \qquad (2.47)\end{aligned}$$

E(·) designates the energy loss function defined in Eq. (2.3). Till now, W^l can be calculated according to (2.47), and the hyperparameter ψ^l can be updated correspondingly according to (2.45). It is worth mentioning that W^l obtained at time t is just the W^{l^*} at time $t+1$ for the reason that the updated W^l and ψ^l can be viewed as a temporary local optimum.

ALGORITHM

In this section, we summarize the iterative procedure derived in Section. 2.1.2.3. The pseudo-code for illustrating the process of Bayesian learning is shown in algorithm 1. By employing the single prior on model parameters, the proposed algorithm can lead to a shape-wise regularization, which drives individual connection weights to 0. A hyperparameter λ is also introduced to tune the regularization. L denotes the number of hidden layers. After the training process, the redundant parameters should be compressed. In this chapter, we adopt the ψ^l and the magnitude of W^l as the pruning criteria. ψ_i^l denotes the uncertainty for the weight W_i^l. A small variance ψ_i^l means high confidence that the corresponding weight W_i^l has a high probability of being zero and does not contribute to the final result. At the same time, the parameter with its magnitude smaller than a threshold is also regarded as a redundant parameter. We define $\kappa_\psi, \kappa_w \in \mathbb{R}^+$ as these two thresholds.

Algorithm 1 Sparse Bayesian Deep Learning Algorithm with Single Prior

Initialize: hyper-parameters $\omega^l, \psi^l = I, l = [1,2,\cdots,L]$; threshold for pruning $\kappa_\psi, \kappa_w \in \mathbb{R}^+$; regularization tuning parameter $\lambda \in \mathbb{R}^+$; $C_{\max} \in \mathbb{N}^+$ denotes the maximum cycles; $E_{\max} \in \mathbb{N}^+$ denotes the number of epochs in each cycle.

for $i = 1$ to $C_{\max}$

  for $j = 1$ to $E_{\max}$

    (1) Update the weight W^l by applying the gradient decent with loss function as

$$\arg\min_{W^l} E(W^l, \sigma^2) + \lambda \sum_{l=1}^{L} ||\omega^l \cdot W^l||_{\ell_1}$$

  end for

  for $l = 1$ to L

    (2) Update ψ^l as Eq. (2.45).

    (3) Update ω^l as Eq. (2.46).

  end for

end for

(4) One-shot prune W^l if $|W^l| < \kappa_w$ and $\psi^l < \kappa_\psi$

Remark 1 *We now give some clarifications on the definition of cycle and epoch in Algorithm 1. One identification "cycle" has E_{max} epochs. One "epoch" refers to that the entire dataset is processed forward and backward by the NN for one time. In the first identification cycle, regularization is conventional ($\omega(0) = 1$). That is, the first obtained model is a sparse model corresponding to the conventional sparse ℓ_1 norm regularization method,*

and sparser models are expected to result from the next identification cycles. The selection of C_{max} will impact the training effect. A C_{max} that is too large will consume more time and lead to overfitting, while a C_{max} that is too small will result in underfitting.

2

2.1.3. SPARSE BAYESIAN DEEP LEARNING ALGORITHM WITH GROUP PRIOR

With the assumption on the independence and non-stationarity of connection weights, the Bayesian approach derived in Section. 2.1.2 can be used to achieve the non-structural (shape-wise) regularization by driving some individual weights to be 0. However, one may want to enforce structured sparsity in some applications. This can be realized by enforcing group priors on a group of weight and re-expressing the regularization term as a function of these groups [27]. Fig. 2.2(b-c) shows the structured regularization of rows and columns. The benefit of such an approach, specific to this thesis, is obtaining compact sparse models and the suppression of input features that are deemed less pertinent without loss of accuracy. An example about input feature selection is in Figure. 2.3. The relevant experiment on system identification is in Chapter 4.

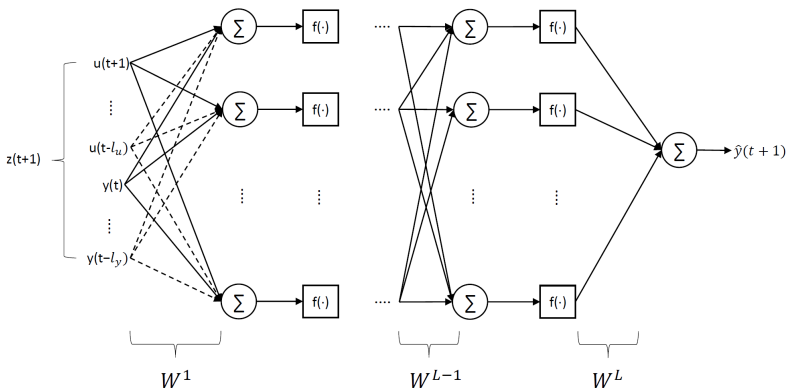


Figure 2.3: An example about selecting input features by employing group prior on model parameters. The outgoing connections of some input features are identified as redundant (dotted line).

In order to distinguish it from the symbols used in Section. 2.1.2, we take the weight matrix W_g^l of layer l as the optimization target in this section. Assuming that the prior probabilities for $p(W_g^l)$ is still a Gaussian relaxed variational form

$$p(W_g^l) \geq p(W_g^l, \psi_g^l) = \mathcal{N}(W_g^l|0, \psi_g^l)\, \phi(\psi_g^l) \tag{2.48}$$

$$\psi_g^l \triangleq [\psi_{g1}^l, \cdots, \psi_{gm}^l, \cdots, \psi_{gM}^l] = \Big[\underbrace{\psi_{g1}^l, \ldots, \psi_{g\aleph_1}^l}_{\aleph_1 \textbf{ elements}} \Big| \cdots \Big| \underbrace{\psi_{gM-\aleph_O+1}^l, \ldots, \psi_{gM}^l}_{\aleph_O \textbf{ elements}} \Big] \tag{2.49}$$

$$W_g^l \triangleq [W_{g1}^l, \cdots, W_{gm}^l, \cdots, W_{gM}^l] = \Big[\underbrace{W_{g1}^l, \ldots, W_{g\aleph_1}^l}_{\aleph_1 \textbf{ elements}} \Big| \cdots \Big| \underbrace{W_{gM-\aleph_O+1}^l, \ldots, W_{gM}^l}_{\aleph_O \textbf{ elements}} \Big] \tag{2.50}$$

where M denotes the number of unknown parameters. $\phi(\psi_g^l)$ represents the hyper-prior probability of Ψ^l and $\psi^l \triangleq \text{diag}(\psi^l)$. It can be found from Eq. (2.49) that the un-

known parameters can be divided into $\aleph_O$ groups, where $\sum_{i=1}^{O} \aleph_i = M$. The ψ^l within each group shares the same value. Therefore, if we use the ψ^l as the network pruning criteria, a group of weights can be retained or compressed simultaneously. Typically, for a 2-D matrix, the weights in the same row or column can be regarded as a group, which means O is equal to the number of rows or columns. It should be noted that these groups are considered independent, but the weight of each connection in a specific group share the same prior Gaussian relaxation (see Fig. 2.2(b-c)). This results in a slightly different iterative update rule for the identification algorithm.

REGULARIZATION UPDATE RULES

With the group prior defined in Eq. (2.48) (2.49), the way to formulate the system identification in a Bayesian framework and the derivation of the loss function is similar to Section. 2.1.2. The difference is with the optimization step for ψ^l. Parameters in the same row share the same prior uncertainty $\psi^l_{a:}$ and the same column the prior uncertainty $\psi^l_{:b}$. According to the optimization step in Eq. (2.43) for ψ^l_{ab}, the prior shared among the weights in the same column becomes

$$\psi^l_{:b}(k+1) = \underset{\psi \geq 0}{\operatorname{argmin}} \sum_{b=1}^{n^l} \frac{W^l_{:b}(k+1)^T W^l_{:b}(k+1)}{\psi^l_{:b}(k)} + \alpha^l_{:b}(k) \cdot \psi^l_{:b}(k) \tag{2.51}$$

where $\alpha^l_{:b} = \sum_{a=1}^{n^{l-1}} \alpha^l_{ab}(k)$. By noting that

$$\sum_{b=1}^{n^l} \frac{W^{l\,T}_{:b} W^l_{:b}}{\psi^l_{:b}} + \alpha^l_{:b} \cdot \psi^l_{:b} \geq 2 \left\| \sqrt{\alpha^l_{:b}} \cdot W^l_{:b} \right\|_{l_2} \tag{2.52}$$

the analytical solution is given by

$$\psi^l_{:b}(k+1) = \frac{||W^l_{:b}(k+1)||_2}{\omega^l_{:b}(k)} \tag{2.53}$$

where

$$\omega^l_{:b}(k) = \sqrt{\alpha^l_{:b}(k)} = \sqrt{\sum_{a=1}^{n^{l-1}} \alpha^l_{ab}(k)} \tag{2.54}$$

The row-wise regularization can be analogously derived. Note that the update rules for α^l_{ab} remains similar to Eq. (2.41) and (2.42).

ALGORITHM

Taking the column-wise regularization as the example. A pseudo-code for the algorithm with the group prior is given by Algorithm 2. Similarly, the row-wise regularization item, the update rules for ψ, ω are in Table 2.1.

Remark 2 *In the first identification cycle, regularization is conventional ($\omega(0) = \mathbb{1}$). That is, the first obtained model is a sparse model corresponding to the conventional sparse group lasso regularization method, and sparser models result from the subsequent identification cycles.*

Algorithm 2 Sparse Bayesian Deep Learning Algorithm with Group Prior

Initialize: hyper-parameters $\omega^l, \psi^l = I, l = [1, 2, \cdots, L]$; threshold for pruning $\kappa_\psi, \kappa_w \in \mathbb{R}^+$; regularization tuning parameter $\lambda \in \mathbb{R}^+$; $C_{\max} \in \mathbb{N}^+$ denotes the maximum cycles; $E_{\max} \in \mathbb{N}^+$ denotes the number of epochs in each cycle.

for $i = 1$ to $C_{\max}$

for $j = 1$ to $E_{\max}$

(1) Update the weight W^l by applying the gradient decent with loss function:

$$\arg\min_{W^l} E(W^l, \sigma^2) + \lambda \sum_{l=1}^{L} \sum_{b=1}^{n^l} ||\omega^l_{:b} \cdot W^l_{:b}||_{\ell_2}$$

end for

for $l = 1$ to L

(2) Update ψ^l as Eq. (2.53).

(3) Update ω^l as Eq. (2.54)

end for end for

(4) One-shot prune W^l if $|W^l| < \kappa_w$ and $\psi^l < \kappa_\psi$

Remark 3 *The algorithm does not exhibit global convergence properties. It shares the local convergence properties (local minima, saddle point) of the adopted stochastic gradient descent method. This is because the Laplace approximation is a local approximation to the energy function $E(\cdot)$ and includes an assumption on the uni-modality of the posterior. However, the introduced pruning and the regularization techniques are heuristics that help speed up the algorithm and improve convergence and optimality. Nonetheless, the identification experiments are run multiple times randomly initialized, and the generated model with the best simulation validation performance is chosen.*

2.1.4. SPARSE BAYESIAN DEEP LEARNING ALGORITHM WITH FUSED PRIOR

As shown in Fig. 2.2(d), the way to enforce fused prior on weight matrix means enforcing both single prior and group prior on the same weight matrix. Taking the weight matrix W^l of layer l as the optimization target, we define the single prior $p(W^l)$ and group prior $p(W^l_g)$ as:

$$p(W^l) \geq p(W^l, \psi^l) = \mathcal{N}(W^l|0, \psi^l)\ \phi(\psi^l) \quad p(W^l_g) \geq p(W^l_g, \psi^l_g) = \mathcal{N}(W^l_g|\mathbf{0}, \psi^l_g) \tag{2.55}$$

where $P(W^l)$ is imposed to regularize each weight. And a group prior $P(W^l_g)$ is imposed to regularize a group of weights with

$$\psi^l \triangleq [\psi^l_1, \cdots, \psi^l_m, \cdots, \psi^l_M]$$

$$\psi^l_g \triangleq [\psi^l_{g1}, \cdots, \psi^l_{gm}, \cdots, \psi^l_{gM}] = \Big[\underbrace{\psi^l_{g1}, \ldots, \psi^l_{g\aleph_1}}_{\aleph_1 \textbf{ elements}} \Big| \cdots \Big| \underbrace{\psi^l_{gM-\aleph_O+1}, \ldots, \psi^l_{gM}}_{\aleph_O \textbf{ elements}} \Big].$$

$$W_g^l \triangleq [W_{g1}^l, \cdots, W_{gm}^l, \cdots, W_{gM}^l] = \Big[\underbrace{W_{g1}^l, \ldots, W_{g\aleph_1}^l}_{\aleph_1 \textbf{ elements}} \Big| \cdots \Big| \underbrace{W_{gM-\aleph_O+1}^l, \ldots, W_{gM}^l}_{\aleph_O \textbf{ elements}} \Big].$$

where M denotes the number of unknown parameters. $\phi(\psi^l)$ and $\phi(\psi_g^l)$ represent the hyperprior probability of ψ^l and ψ_g^l, respectively. For the group regularization, the unknown parameters can be divided into $\aleph_O$ groups, where $\sum_{i=1}^{O} \aleph_i = M$. The ψ^l within each group shares the same value. However, when using both single prior and group prior, there would be a slightly different iterative update rule for the identification algorithm.

REGULARIZATION UPDATE RULES

The difference is with the optimization step for ψ. When using both shape-wise and group-wise regularization, the posterior is updated according to a combined prior given by:

$$\hat{\psi}_i(k) = \frac{1}{(\frac{1}{\psi_i(k)} + \frac{1}{\psi_g(k)})} \tag{2.56}$$

The last row in Table 2.1 summarizes the update rules for ψ, ω, α according to the category of regularization techniques adopted. $R(\cdot)$ in Table 2.1 represents the corresponding regularization items. Note that the update rules for ψ_i remain similar to Eq. (2.45) of Section. 2.1.2, and the update rules for ψ_g remain similar to Eq. (2.53) of Section. 2.1.2.

ALGORITHM

If both the shape-wise and column-wise regularization are adopted, a pseudo-code for the algorithm is given by Algorithm 3.

A complete summary of the calculation of hyper-parameters (i.e., ψ, ω, α) and the corresponding regularization item $R(\cdot)$ is in Table. 2.1. It should be noted that we use different terms in Table. 2.1 to represent the regularization based on different priors. Specifically, the "shape-wise" in the first row represents the single prior leading to shape-wise regularization (Fig. 2.2(a)). The "row-wise" (Fig. 2.2(b)) and "column-wise" (Fig. 2.2(c)) in the second and third row represent the group prior leading to row-wise or column-wise regularization. The "row-wise + column-wise" (Fig. 2.2(d)), "shape-wise + row-wise" (Fig. 2.2(e)) and "shape-wise + column-wise" (Fig. 2.2(f)) in last three rows represent the fused priors which is combination of different priors.

As explained in Section. 2.1.2.1, the Laplace approximation methods require the Hessian calculation for weight matrices. In the next section, we propose efficient Hessian calculation methods for several typical parametric layers, i.e., Fully-connected, convolutional and recurrent layers.

2.2. HESSIAN CALCULATION

As explained in Section. 2.1.2.1, the Hessian of weight matrices is also a necessity for the Laplace approximation method. We have to find effective calculation methods to compute the Hessian of the weight matrices in a neural network model. This section

Table 2.1: Hyper-parameter update rule based on regularization technique.

Category	Prior Formulation	$R(\omega^l, W^l)$	ω^l	ψ^l
(a) Shape-wise	$\prod_{a=1}^{n_{l-1}} \prod_{b=1}^{n_l} p(W^l_{ab}, \psi^l_{ab})$	$\sum_{a=1}^{n_{l-1}} \sum_{b=1}^{n_l} \|\omega^l_{ab}(k) \cdot W^l_{ab}(k)\|_{l_1}$	$\omega^l_{ab}(k) = \sqrt{\alpha^l_{ab}(k)}$	$\psi^l_{ab}(k) = \frac{\|W^l_{ab}(k)\|}{\omega^l_{ab}(k-1)}$
(b) Row-wise	$\prod_{a=1}^{n_{l-1}} p(W^l_{a:}, \psi^l_{a:})$	$\sum_{a=1}^{n_{l-1}} \|\|\omega^l_{a:}(k) \cdot W^l_{a:}(k)\|\|_{l_2}$	$\omega^l_{a:}(k) = \sqrt{\sum_{b=1}^{n_l} \alpha^l_{ab}(k)}$	$\psi^l_{a:}(k) = \frac{\|\|W^l_{a:}(k)\|\|_2}{\omega^l_{a:}(k-1)}$
(c) Column-wise	$\prod_{b=1}^{n_l} p(W^l_{:b}, \psi^l_{:b})$	$\sum_{b=1}^{n_l} \|\|\omega^l_{:b}(k) \cdot W^l_{:b}(k)\|\|_{l_2}$	$\omega^l_{:b}(k) = \sqrt{\sum_{a=1}^{n_{l-1}} \alpha^l_{ab}(k)}$	$\psi^l_{:b}(k) = \frac{\|\|W^l_{:b}(k)\|\|_2}{\omega^l_{:b}(k-1)}$
(b) Row-wise + (c) Column-wise	$\prod_{a=1}^{n_{l-1}} p(W^l_{a:}, \psi^l_{a:}) \times \prod_{b=1}^{n_l} p(W^l_{:b}, \psi^l_{:b})$	$\sum_{a=1}^{n_{l-1}} \|\omega^l_{a:}(k) \cdot W^l_{a:}(k)\|\|_{l_2} + \sum_{b=1}^{n_l} \|\omega^l_{:b}(k) \cdot W^l_{:b}(k)\|\|_{l_2}$	$\omega^l_{a:}(k) = \sqrt{\sum_{b=1}^{n_l} \alpha^l_{ab}(k)}$ $\omega^l_{:b}(k) = \sqrt{\sum_{a=1}^{n_{l-1}} \alpha^l_{ab}(k)}$	$\psi^l_{a:}(k) = \frac{\|\|W^l_{a:}(k)\|\|_2}{\omega^l_{a:}(k-1)}$ $\psi^l_{:b}(k) = \frac{\|\|W^l_{:b}(k)\|\|_2}{\omega^l_{b:}(k-1)}$ $\psi^l_{ab}(k) = 1/(\frac{1}{\psi^l_{a:}(k)} + \frac{1}{\psi^l_{:b}(k)})$
(a) Shape-wise + (b) Row-wise	$\prod_{a=1}^{n_{l-1}} \prod_{b=1}^{n_l} p(W^l_{ab}, \psi^l_{ab}) \times \prod_{a=1}^{n_{l-1}} p(W^l_{a:}, \psi^l_{a:})$	$\sum_{a=1}^{n_{l-1}} \sum_{b=1}^{n_l} \|\omega^l_{ab}(k) \cdot W^l_{ab}(k)\|_{l_1} + \sum_{a=1}^{n_{l-1}} \|\omega^l_{a:}(k) \cdot W^l_{a:}(k)\|\|_{l_2}$	$\omega^l_{ab}(k) = \sqrt{\alpha^l_{ab}(k)}$ $\omega^l_{a:}(k) = \sqrt{\sum_{b=1}^{n_l} \alpha^l_{ab}(k)}$	$\psi^l_{ab}(k) = \frac{\|W^l_{ab}(k)\|}{\omega^l_{ab}(k-1)}$ $\psi^l_{a:}(k) = \frac{\|\|W^l_{a:}(k)\|\|_2}{\omega^l_{a:}(k-1)}$ $\psi^l_{ab}(k) = 1/(\frac{1}{\psi^l_{ab}(k)} + \frac{1}{\psi^l_{a:}(k)})$
(a) Shape-wise + (c) Column-wise	$\prod_{a=1}^{n_{l-1}} \prod_{b=1}^{n_l} p(W^l_{ab}, \psi^l_{ab}) \times \prod_{b=1}^{n_l} p(W^l_{:b}, \psi^l_{:b})$	$\sum_{a=1}^{n_{l-1}} \sum_{b=1}^{n_l} \|\omega^l_{ab}(k) \cdot W^l_{ab}(k)\|_{l_1} + \sum_{b=1}^{n_l} \|\omega^l_{:b}(k) \cdot W^l_{:b}(k)\|\|_{l_2}$	$\omega^l_{ab}(k) = \sqrt{\alpha^l_{ab}(k)}$ $\omega^l_{:b}(k) = \sqrt{\sum_{a=1}^{n_{l-1}} \alpha^l_{ab}(k)}$	$\psi^l_{ab}(k) = \frac{\|W^l_{ab}(k)\|}{\omega^l_{ab}(k-1)}$ $\psi^l_{:b}(k) = \frac{\|\|W^l_{:b}(k)\|\|_2}{\omega^l_{b:}(k-1)}$ $\psi^l_{ab}(k) = 1/(\frac{1}{\psi^l_{ab}(k)} + \frac{1}{\psi^l_{:b}(k)})$

Algorithm 3 Sparse Bayesian Deep Learning Algorithm with Fused Prior

Initialize: hyper-parameters $\omega^l, \psi^l = I, l = [1, 2, \cdots, L]$; threshold for pruning $\kappa_\psi, \kappa_w \in \mathbb{R}^+$; regularization tuning parameter $\lambda \in \mathbb{R}^+$; $C_{\max} \in \mathbb{N}^+$ denotes the maximum cycles; $E_{\max} \in \mathbb{N}^+$ denotes the number of epochs in each cycle.

for $i = 1$ to $C_{\max}$

for $j = 1$ to $E_{\max}$

(1) Update the weight W^l by applying the gradient decent with loss function as

$$\arg\min_{W^l} E(W^l, \sigma^2) + \lambda \sum_{l=1}^{L} ||\omega^l \cdot W^l||_{\ell_1} + \lambda_g \sum_{l=1}^{L} \sum_{b=1}^{n^l} ||\omega^l_{:b} \cdot W^l_{:b}||_{\ell_2}$$

end for

for $l = 1$ to L

(2) Update ψ^l and $\psi^l_{:,b}$ as Eq. (2.45) and Eq. (2.53), respectively.

(3) Update $\hat{\psi}^l$ as Eq. (2.56).

(4) Update ω^l and $\omega^l_{:,b}$ as Eq. (2.46) and Eq. (2.54), respectively.

end for

end for

(4) One-shot prune W^l if $|W^l| < \kappa_w$ and $\hat{\psi}^l < \kappa_\psi$

will present the proposed efficient and recursive Hessian calculation methods for fully-connected, convolutional and recurrent layers. In the beginning, we will explain the mathematical definition and properties of Hessian. The Hessian of the weight matrix $W \in \mathbb{R}^{m \times n}$ is a square matrix of the second-order partial derivatives of the loss function, which includes the local curvature information and can be formulated as:

$$\mathbf{H}_{\mathscr{L}} = \begin{bmatrix} \frac{\partial^2 \mathscr{L}}{\partial \vec{W}_1^2} & \frac{\partial^2 \mathscr{L}}{\partial \vec{W}_1 \partial \vec{W}_2} & \cdots & \frac{\partial^2 \mathscr{L}}{\partial \vec{W}_1 \partial \vec{W}_{mn}} \\ \frac{\partial^2 \mathscr{L}}{\partial \vec{W}_2 \partial \vec{W}_1} & \frac{\partial^2 \mathscr{L}}{\partial \vec{W}_2^2} & \cdots & \frac{\partial^2 \mathscr{L}}{\partial \vec{W}_2 \partial \vec{W}_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 \mathscr{L}}{\partial \vec{W}_{mn} \partial \vec{W}_1} & \frac{\partial^2 \mathscr{L}}{\partial \vec{W}_{mn} \partial \vec{W}_2} & \cdots & \frac{\partial^2 \mathscr{L}}{\partial \vec{W}_{mn,mn}^2} \end{bmatrix} \tag{2.57}$$

where the (i, j) element of the $\mathbf{H}_{\mathscr{L}}$ can be represented as:

$$[\mathbf{H}_{\mathscr{L}}]_{ij} = \frac{\partial^2 \mathscr{L}}{\partial \vec{W}_i \partial \vec{W}_j} \tag{2.58}$$

It can be found that the dimension of $\mathbf{H}_{\mathscr{L}}$ is the square of the number of parameters, e.g., $\mathbf{H}_{\mathscr{L}} \in \mathbb{R}^{mn \times mn}$ if $W \in \mathbb{R}^{m \times n}$. Therefore, the Hessian calculation and storage for large-scale neural networks are infeasible due to their millions of parameters [2]. Therefore, we need to find the practical and effective calculation method to relieve the computation burden. It should be noted that the weight $\vec{W} \in \mathbb{R}^{mn}$ here is the vectorization of the original

multi-dimension weight matrix $W \in \mathbb{R}^{m\times n}$. The vectorization is performed by the defined vectorization operator. The definition of vectorization operators for the $2D$ matrix and $3D$ matrix are in definition 1 and definition 2, respectively. The Hessian matrix of W to the loss function is also known as the jacobian matrix of the gradient of W to the loss function. Since the dimension of the Hessian is decided by the amount of parameters, it would be convenient to explain the Hessian calculation by treating the matrix as a vector. Therefore, we define the vectorization operators for $2D$ and $3D$ matrix in definition 1 and definition 2, respectively. It should be noted that the $2D$ matrix is the typical matrix in a Fully-connected neural network and recurrent neural network. In contrast, the $3D$ matrix is the typical matrix in the convolutional neural network.

Definition 1 *The vectorization operator for a $2D$ matrix $A \in \mathbb{R}^{m\times n}$ is defined as stacking the elements of A into a vector $\vec{A} \in \mathbb{R}^{mn}$ by assembling the columns of A sequentially. Formally, the vectorization process $\mathbb{V}_{2D} : \mathbb{R}^{m\times n} \to \mathbb{R}^{mn}$ is:*

$$\mathbb{V}_{2D}(A) = \begin{bmatrix} A_{:,1} \\ A_{:,2} \\ \cdots \\ A_{:,n} \end{bmatrix} \tag{2.59}$$

where $A_{:,i} = \begin{bmatrix} A_{1,i} A_{2,i} \cdots A_{m,i} \end{bmatrix}^\top$ The operator $\mathbb{V}_{2D}$ represents the identity map if the input A is a vector itself.

Definition 2 *The vectorization operator for a $3D$ matrix $A \in \mathbb{R}^{m\times n\times o}$ is defined as stacking the elements of A into a vector $\vec{A} \in \mathbb{R}^{mno}$ by assembling the vectorization of the $2D$ matrix $A_{i,:,:}, i = [1, 2, \cdots, m]$ with the $2D$ vectorization operator 1 sequentially. Formally, the vectorization process $\mathbb{V}_{3D} : \mathbb{R}^{m\times n\times o} \to \mathbb{R}^{mno}$ is:*

$$\mathbb{V}_{3D}(A) = \begin{bmatrix} \mathbb{V}_{2D}(A_{1,:,:}) \\ \mathbb{V}_{2D}(A_{2,:,:}) \\ \cdots \\ \mathbb{V}_{2D}(A_{m-1,:,:}) \\ \mathbb{V}_{2D}(A_{m,:,:}) \end{bmatrix} \tag{2.60}$$

It should be noted that these vectorization operators represent an isomorphic map. Besides the vectorization operators, we also propose the definition of de-vectorization operators, which will be useful for the Hessian calculation of Conv layers in Section. 2.2.2.2.

Definition 3 *The 2D de-vectorization operator to reshape a vector $\vec{A} \in \mathbb{R}^{mn}$ to a $2D$ matrix $A \in \mathbb{R}^{m\times n}$ is defined as truncating $\vec{A}$ into m equal segments, which are stacked sequentially as a $2D$ matrix shaped as $m \times n$. Formally, the de-vectorization process $\mathbb{DV}_{2D} : \mathbb{R}^{mn} \to \mathbb{R}^{m\times n}$ is:*

$$\mathbb{DV}_{2D}(\vec{A}) = \begin{bmatrix} \vec{A}[1:n] \\ \vec{A}[n+1:2n] \\ \cdots \\ \vec{A}_{[}(m-1)n+1:mn] \end{bmatrix} \tag{2.61}$$

Definition 4 *The 3D de-vectorization operator to reshape a vector $\vec{A} \in \mathbb{R}^{mno}$ to a 3D matrix $A \in \mathbb{R}^{m\times n\times o}$ is implemented by two steps. First, truncate the $\vec{A}$ into m equal segments, where each segment is denoted as $\vec{A}_{(i-1)no+1:ino} \in \mathbb{R}^{no}, i == [1,2,\cdots,m]$. Second, de-vectorize each segment $A_{(i-1)no+1:ino}$ into a 2D matrix shaped as $n \times o$ with 2D de-vectorization operator $\mathbb{DV}_{2D}(\vec{A})$. The de-vectorized 2D matrix is stacked sequentially as a 3D matrix shaped as $m \times n \times o$. Formally, the de-vectorization process $\mathbb{DV}_{3D} : \mathbb{R}^{mno} \rightarrow \mathbb{R}^{m\times n\times o}$ is:*

$$\mathbb{DV}_{3D}(\vec{A}) = \begin{bmatrix} \mathbb{DV}_{2D}(\vec{A}_{1:no}) \\ \mathbb{DV}_{2D}(\vec{A}_{no+1:2no}) \\ \cdots \\ \mathbb{DV}_{2D}(\vec{A}_{(m-1)no+1:mno}) \end{bmatrix} \tag{2.62}$$

The definition of $\mathbb{DV}_{2D}$ is in definition 2.61.

The Hessian of weight matrices can benefit the DNN training from two aspects. First, it can accelerate optimization by adopting the Hessian in second-order optimization algorithms, e.g., the Quasi-Newton methods [4, 20]. First-order optimization methods (e.g., gradient descent, gradient descent with momentum [23]) are the mainstream for deep learning by leveraging their advantages, such as easily implementation with mature deep learning frameworks (e.g., PyTorch and TensorFlow) and scalability to large models and datasets [11, 29]. However, they also suffer the shortages such as heavy tuning effort for hyper-parameters (e.g., learning rate), sensitivity to weight initialization and easily fall into a local minimum. On the other hand, the second-order optimization methods have shown their competence in rapid convergence without much tuning work [2, 3]. [17] also demonstrated that the second-order optimization method could perform better than the first-order approaches on problems such as pathological curvature, where the first-order method often falls into the "valleys" with large varying curvatures because of their lack of ability to capture the curvature information [5, 19]. However, the second-order approaches escape such pitfalls by leveraging the Hessian information, also known as the curvature matrix. The use of curvature information can speed up the search process per step. Second, the Hessian of weight matrices is also a necessity for the Laplace approximation method (see details in Section. 2.1.2.1). The Hessian is used to calculate the posterior distribution of weight parameters as in Eq. 2.17 and incorporated to update the loss function in each cycle. [16] has successfully adopted the curvature matrix as the precision matrix of the approximated Gaussian distribution in some small scale neural networks.

Since the Hessian is diagonal dominant practically [19], [2, 3] presented an efficient Hessian calculation method for fully-connected (FC) layer. The proposed approach focused on the diagonal blocks of the Hessian. Each block represents the diagonal entries of the Hessian in each layer and can be calculated recursively along with the back-propagation process using Kronecker products. However, this approach can only be used to approximate the diagonal blocks of the Hessian for fully connected layers. It cannot be used for other parametric layers, such as the convolutional (Conv) and recurrent layers. It can be observed that the difficulty of Hessian calculation for the Conv layer shall increase due to the indirect convolution operation, as well as for the recurrent layer due to the repeated multiplications of the same weights [18]. This motivates us to explore the efficient Hessian calculation/approximation methods for the Conv and recurrent layer in

this thesis. Inspired by the Hessian calculation methods for FC layers [3], we develop the block-diagonal approximation methods for the Hessian of Conv and recurrent layers, respectively. Specifically, the Hessian calculation methods for these three typical deep neural networks are in the following sections.

2

2.2.1. COMPUTE THE HESSIAN OF FULLY-CONNECTED LAYER

FULLY-CONNECTED NEURAL NETWORK

Fully-connected (FC) neural network is a feed-forward network constructed by arranging perceptron type neurons in different layers. Fig. 2.4 is a schematic representation of a simple FC neural network with two hidden layers. As a FC network, every neuron within

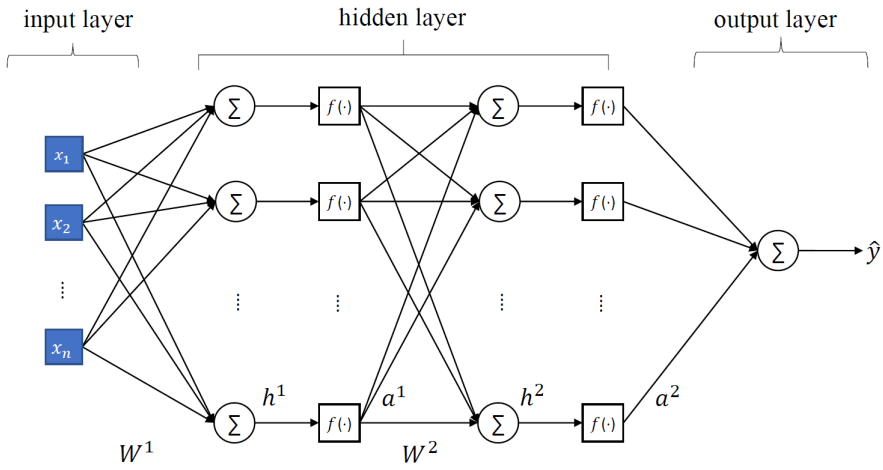


Figure 2.4: A Fully connected neural network with two hidden layers.

the model is connected to other neurons in the adjacent layers. As shown in Fig. 2.4, a^2, the output of the *2-nd* hidden layer can be computed as:

$$h^2 = W^2 a^1 + b^2 \tag{2.63a}$$

$$a^2 = f(h^2) \tag{2.63b}$$

where the superscript of W, h, b, a denotes the layer index. Eq. (2.63a) stands for the linear combination, computed as the sum of product between the weight matrix W^2 and input vector a^1. W^2_{ij} is the weighted scalar that determines the strength of the connection between the *i-th* neuron in the input layer and *j-th* neuron in the hidden layer. b is the bias. The nonlinear behaviour of the neural network model is decided by $f(\cdot)$, which is named the nonlinear activation function.

HESSIAN CALCULATION

In this section, we will first recap the derivation process of the Hessian calculation methods for the fully-connected layer proposed by [3]. And then, we present our simplified Hessian calculation method, which can save more computation effort.

Lemma 1 *For a Fully-connected layer, given the activation function f, the activation value a^l, a^{l-1} and the pre-activation value h^l, the Hessian of the weight matrix W^l can be calculated recursively:*

$$\mathbf{H}^l = a^{l-1} \cdot (a^{l-1})^\top \otimes H^l \tag{2.64}$$

where $\otimes$ stands for Kronecker product. H^l is the pre-activation Hessian and can be updated as:

$$H^l = B^l (W^{l+1})^\top H^{l+1} W^{l+1} B^l + D^l \tag{2.65}$$

with two diagonal matrices B^l and D^l defined as:

$$B^l = \text{diag}(f'(h^l)), \quad D^l = \text{diag}(f''(h^l) \circ \frac{\partial L}{\partial a^l}) \tag{2.66}$$

where $\circ$ represents the element-wise multiplication. The initialised H^l is the second-order derivative of the loss function to the output of the neural network.

Proof 1 *Since the Hessian is diagonally dominant, we mainly consider how to obtain the diagonal value of the Hessian matrix. Suppose W^l is a 2D matrix with $W^l \in \mathbb{R}^{m \times n}$, the diagonal value of $\mathbf{H}^l$ can be computed as the jacobian matrix of the gradient of W^l concerning the vectorization of W^l:*

$$\begin{aligned}
&\frac{\partial^2 \mathscr{L}}{\partial \mathbb{V}_{2D}(W^l)^2} = \frac{\partial}{\partial \mathbb{V}_{2D}(W^l)} \left(\frac{\partial \mathscr{L}}{\partial \mathbb{V}_{2D}(W^l)} \right) = \frac{\partial}{\partial W^l} \left(\frac{\partial \mathscr{L}}{\partial h^l} \frac{\partial h^l}{W^l} \right) = \frac{\partial}{\partial W^l} \left(\left(1 \otimes \frac{\partial \mathscr{L}}{\partial h^l} \right) \left(\frac{\partial h^l}{W^l} \otimes \mathbf{I} \right) \right) \\
=&\frac{\partial}{\partial W^l} \left((\mathbf{a}^{l-1})^\top \otimes \frac{\partial \mathscr{L}}{\partial \mathbf{h}^l} \right) = (\mathbf{a}^{l-1})^\top \otimes \frac{\partial^2 \mathscr{L}}{\partial W^l \partial h^l} = (\mathbf{a}^{l-1})^\top \otimes \left(\frac{\partial h_l^\top}{\partial W_l} \frac{\partial^2 \mathscr{L}}{\partial (h^l)^2} \right) \\
=&(\mathbf{a}^{l-1})^\top \otimes \left(\left((a^{l-1})^\top \otimes \mathbf{I} \right)^\top \frac{\partial^2 \mathscr{L}}{\partial (h^l)^2} \right) = (a^{l-1})^\top \otimes \left(a^{l-1} \otimes \frac{\partial^2 \mathscr{L}}{\partial (h^l)^2} \right) \\
=&(a^{l-1})^\top \otimes a^{l-1} \otimes \frac{\partial^2 \mathscr{L}}{\partial (h^l)^2} \\
=&\left(a^{l-1} (a^{l-1})^\top \right) \otimes \frac{\partial^2 \mathscr{L}}{\partial (h^l)^2}
\end{aligned} \tag{2.67}$$

where $\frac{\partial^2 \mathscr{L}}{\partial (h^l)^2}$ is defined as the pre-activation Hessian H^l, which can be calculated as follows:

$$\begin{aligned}
H^l = \frac{\partial^2 \mathscr{L}}{(\partial h^l)^2} &= \frac{\partial}{\partial h^l} \left(\frac{\partial \mathscr{L}}{\partial h^l} \right) = \frac{\partial}{\partial h^l} \left(\frac{\partial \mathscr{L}}{\partial h^{l+1}} \frac{\partial h^{l+1}}{\partial a^l} \frac{\partial a^l}{\partial h^l} \right) \\
=&\frac{\partial^2 \mathscr{L}}{\partial h^l \partial h^{l+1}} W^{l+1} \frac{\partial a^l}{\partial h^l} + \frac{\partial \mathscr{L}}{\partial h^{l+1}} \frac{\partial h^{l+1}}{\partial a^l} \frac{\partial}{\partial h^l} \left(\frac{\partial a^l}{\partial h^l} \right) \\
=&\frac{\partial (a^l)^\top}{\partial h^l} (W^{l+1})^\top H^{l+1} W^{l+1} \frac{\partial a^l}{\partial h^l} + \sum_k \frac{\partial \mathscr{L}}{\partial a_k^l} \frac{\partial^2 a_k^l}{\partial (h^l)^2}
\end{aligned} \tag{2.68}$$

Define the diagonal matrices B^l and D^l as:

$$B^l = \text{diag}(\frac{\partial (a^l)^\top}{\partial h^l}) = \text{diag}(f'(h^l)), \quad D^l = \text{diag}(\sum_k \frac{\partial \mathscr{L}}{\partial a_k^l} \frac{\partial^2 a_k^l}{\partial (h^l)^2}) = \text{diag}(f''(h^l) \circ \frac{\partial L}{\partial a^l}) \tag{2.69}$$

Eq. (2.68) *can be reformulated as:*

$$H^l = B^l (W^{l+1})^\top H^{l+1} W^{l+1} B^l + D^l \tag{2.70}$$

where diag($\vec{A}$) *in Eq.* (2.69) *refers to generate a diagonal matrix whose diagonal value is extracted from the vector* $\vec{A}$.

In this thesis, in order to reduce computation complexity, we make a further simplification by extracting the diagonal values of the pre-activation Hessian H in Eq. (2.68) and Hessian $\mathbf{H}$ Eq. (2.67) for recursive computation. Thus the matrix multiplication could be reduced to vector multiplication. The Hessian calculation process can be approximated as:

$$\hat{\mathbf{H}}^l = \text{diag}((a^{l-1})^2 \otimes \hat{H}^l) \tag{2.71}$$

$$\hat{H}^l = \hat{B}^2 \circ \left(\left((W^{l+1})^\top \right)^2 \hat{H}^{l+1} \right) + \hat{D}^l, \quad \hat{B}^l = f'(h^l), \quad \hat{D}^l = f''(h^l) \circ \frac{\partial L}{\partial a^l} \tag{2.72}$$

If we compute Hessian with the approximate method as Eq. (2.71)-(2.72), the multiply accumulate operation (MACs) for the pre-activation Hessian H and Hessian $\mathbf{H}$ could be reduced from $n(2m^2+2n^2+4mn+3m-1)$ to $n(2+4m)$ with $W \in \mathbb{R}^{n \times m}$. (e.g., if $n = 100, m = 100$, the original method requires 107.97 MMACs compared with only 0.04 MMACs for the approximate method.)

Based on the above result, we conclude the simplified Hessian calculation method in the following lemma:

Lemma 2 *For a Fully-connected layer, given the activation function* f*, the activation value* a^l, a^{l-1} *and the pre-activation value* h^l*, the Hessian of the weight matrix* W^l *can be calculated recursively:*

$$\hat{\mathbf{H}}^l = \text{diag}((a^l)^2 \otimes \hat{H}^l) \tag{2.73}$$

where $\otimes$ *stands for Kronecker product.* $\hat{H}^l$ *is the pre-activation Hessian and can be updated as:*

$$\hat{H}^l = \hat{B}^2 \circ \left(\left((W^{l+1})^\top \right)^2 \hat{H}^{l+1} \right) + \hat{D}^l \tag{2.74}$$

with $\hat{B}^l$ *and* $\hat{D}^l$ *are defined as:*

$$\hat{B}^l = f'(h^l), \quad \hat{D}^l = f''(h^l) \circ \frac{\partial L}{\partial a^l} \tag{2.75}$$

where $\circ$ *represents the element-wise multiplication. The initialised* $\hat{H}^l$ *is the second-order derivative of the loss function to the output of the neural network.*

It can be noted that there is a gap between the diagonal matrix (lemma 2) to the full Hessian matrix (lemma 1), which means that our proposed method inevitably introduces errors. In other words, we are trading the computational accuracy for improved computational efficiency. The gap is indeed a shortage of our method. Although we cannot theoretically assessed the influence of this gap, we found several previous works on Hessian

inversion are very related and support the rationality of our method. Specifically, with diagonal or low-rank approximations to the curvature matrix, [19] proposed the Kronecker-factored Approximate Curvature (K-FAC) method, which can approximate natural gradient descent and work very well in highly stochastic optimisation regimes. The experiment results in [1] presented that diagonal approximation of the Hessian by finite differences can perform better even than the well-established algorithms, e.g., steepest descent and Broyden–Fletcher–Goldfarb–Shanno (BFGS). The variational approach is adopted in [32] to determine the diagonal updating matrices to satisfy the quasi-Cauchy (QC) relation. Since the Hessian of the minimising function is diagonally dominant (a common assumption in nonlinear optimisation) [1], the above works have demonstrated that the approximation of the diagonal elements of Hessian is an effective approximated and often used solution to calculate the inverse Hessian. However, few of these works addressed how to approximate the diagonal elements of the Hessian for deep neural networks, which is nontrivial as it can promote the application of second-order optimisation methods from machine learning (e.g., the Quasi-Newton methods [4, 20]) on the training of DNNs. And it is especially challenging for "deep" models and vary from time to time on different network architectures, e.g., LSTM.

It should also be noted that lemma 1 and lemma 2 illustrate the detailed procedures to calculate the Hessian referred to a single data point (i.e., $b = 1$). If the number of data points is more than 1 (i.e., $b > 1$), the Hessian is calculated as the average of the Hessian of an individual data point. The lemma 2 is a modification of the Hessian calculation methods proposed by [3], which is also the inspiration of our Hessian calculation approach for the Conv layer and recurrent layer. We will revisit this lemma repeatedly in the next sections.

2.2.2. COMPUTE THE HESSIAN OF CONVOLUTIONAL LAYER

CONVOLUTIONAL NEURAL NETWORK

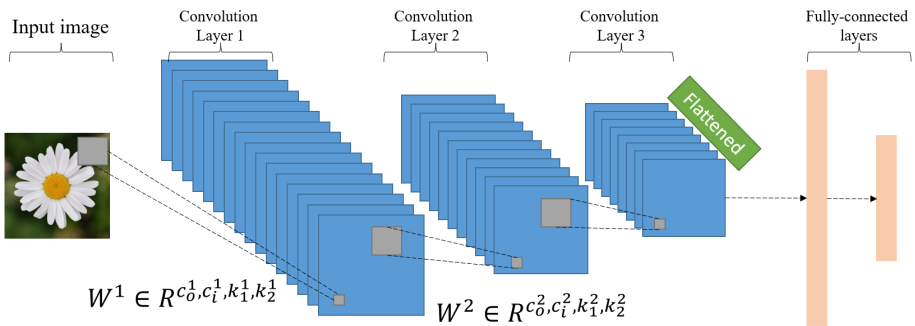


Figure 2.5: A convlutional neural network with standard architecture.

The convolutional neural network (CNN) is typically developed to perform computer vision tasks, including but not limited to the image classification tasks [9, 10, 12] and object detection tasks [6, 7, 24]. Different to the FC neural network, which extracts the information via a direct linear map, the convolutional layer enables the DNN to integrate the

information through convolutional operations, which includes a bank of small filters and can perform the spatial convolution of the input features. The assembly of the filters in each Conv layer is also called a convolutional kernel.

As shown in Fig. 2.5, the standard architecture of a CNN typically includes the convolutional layers and fully connected layers. The input information will be processed by several convolutional layers in the first place, followed by the several fully-connected layers. It should be noted that the 3D input feature will be flattened into a vector before being imparted to the fully-connected layers. The nonlinear activation functions are normally applied after the convolutional layers.

HESSIAN CALCULATION

Unlike the FC layer, the difficulty of the Hessian calculation method for a Conv layer comes from the indirect convolution operation. In this section, this challenge is addressed by converting a Conv layer to its equivalent FC layer, then the Hessian calculation method for a FC layer as in Section. 2.2.1.2 can be used to calculate the Hessian for a Conv layer. A recursive and efficient approach to compute the block diagonal of the Hessian for Conv layers is developed. Before continuing the following, we first give a lemma and two definitions, which are used to explain how to convert a Conv layer to its equivalent FC layer [14]. This lemma will be revisited repeatedly for the proposed Hessian calculation approach.

Suppose the input vector, weight and output vector of a Conv layer are denoted as $B_i^l \in \mathbb{R}^{b\times C_i^l \times H_i^l \times W_i^l}$, $W^l \in \mathbb{R}^{C_o^l \times C_i^l \times k_1^l \times k_2^l}$ and $B_o^l \in \mathbb{R}^{b\times C_o^l \times H_o^l \times W_o^l}$ respectively, where b is the batch size, C_i^l, H_i^l, W_i^l of B_i^l are the size of input channel, height and width; C_o^l, H_o^l, W_o^l are the size of output channel, height and width; $k_1^l \times k_2^l$ is the kernel dimension. With these variables, the lemma and definitions about converting convolutional operation to matrix multiplication are given as follows:

Definition 5 *Assume the input vector and weight of a spatial convolutional operator $f_{conv}(\cdot)$ are $B_i \in \mathbb{R}^{b\times C_i \times H_i \times W_i}$ and $W \in \mathbb{R}^{C_o \times C_i \times k_1 \times k_2}$, respectively, then the spatial convolutional operator $f_{conv}(\cdot) : \mathbb{R}^{b\times C_i \times H_i \times W_i} \rightarrow \mathbb{R}^{b\times C_o \times H_o \times W_o}$ is defined as:*

$$B_o = f_{conv}(B_i, W) \tag{2.76}$$

where $B_o \in \mathbb{R}^{b\times C_o \times H_o \times W_o}$ with

$$H_o = \frac{H_i + 2P - k_1}{s} + 1 \qquad W_o = \frac{W_i + 2P - k_2}{s} + 1 \tag{2.77}$$

where the number of rows or columns padded to each side of the input is denoted as P. s represent the stride of the kernel, which is usually the same for both width and height direction.

Definition 6 *Assume the input vector and weight of a matrix multiplication operator $g_{fc}(\cdot)$ is $M_i \in \mathbb{R}^{m\times n}$ and $W \in \mathbb{R}^{n\times k}$, respectively, then the matrix multiplication operator $g_{fc}(\cdot) : \mathbb{R}^{m\times n} \rightarrow \mathbb{R}^{m\times k}$ is defined as:*

$$M_o = g_{fc}(M_i, W) = M_i \times W \tag{2.78}$$

where $M_o \in \mathbb{R}^{m\times k}$.

Besides, it should be noted that the input, output and kernel of a convolutional layer are usually $4D$ matrices with the form $A \in \mathbb{R}^{m,n,o,p}$. The dimension reduction of a $4D$ matrix to a $2D$ matrix will be used for the Hessian calculation of a Conv layer (see more details in lemma 3). We give the definitions of the dimensionality reduction operator and the dimensionality addition operator as follows:

Definition 7 *Given a $4D$ matrix $A \in \mathbb{R}^{d_1,d_2,d_3,d_4}$ with four axes, the dimensionality reduction operator for reducing the dimension of A to a $2D$ matrix shaped as $(d_j, \prod_{i=1,i\neq j}^{4} d_i)$ along the j-th ($j = [1,2,\cdots,4]$) axis is implemented by two steps. Taking $j = 2$ as the example, the first step is the sequential vectorization of the $3D$ matrix $A_{:,k,:,:}, k = [1,\cdots,d_2]$. The second step is to stack these vectors into a $2D$ matrix. Formally, the dimensionality reduction from $\underset{4D\to 2D}{\mathbb{DR}} : \mathbb{R}^{d_1,d_2,d_3,d_4} \to \mathbb{R}^{d_j,\prod_{i=1,i\neq j}^{4} d_i}$ is:*

$$\underset{4D\to 2D}{\mathbb{DR}}(A) = \begin{bmatrix} \mathbb{V}_{3D}(A_{:,1,:,:}) \\ \cdots \\ \mathbb{V}_{3D}(A_{:,2,:,:}) \\ \cdots \\ \mathbb{V}_{3D}(A_{:,d_2-1,:,:}) \\ \cdots \\ \mathbb{V}_{3D}(A_{:,d_2,:,:}) \end{bmatrix} \tag{2.79}$$

The definition of $\mathbb{V}_{3D}$ is in definition 2.

Definition 8 *Given a $2D$ matrix $A_M \in \mathbb{R}^{d_j,\prod_{i=1,i\neq j}^{4} d_i}$ generated by applying the dimensionality reduction operator (see definition 7) to a $4D$ matrix $A \in \mathbb{R}^{d_1,d_2,d_3,d_4}$, the dimensionality addition operator can recover the original $4D$ matrix by reshaping the A_M by stacking the $3D$ matrix generated by applying the $3D$ de-vectorization operator along the row of A_M sequentially. Formally, the dimensionality addition operator from $\underset{2D\to 4D}{\mathbb{DA}} : \mathbb{R}^{d_j,\prod_{i=1,i\neq j}^{4} d_i} \to \mathbb{R}^{d_1,d_2,d_3,d_4}$ is:*

$$\underset{2D\to 4D}{\mathbb{DA}}(A_M) = \begin{bmatrix} \mathbb{DV}_{3D}(A_{M1,:}) \\ \mathbb{DV}_{3D}(A_{M2,:}) \\ \cdots \\ \mathbb{DV}_{3D}(A_{Md_j-1,:}) \\ \mathbb{DV}_{3D}(A_{Md_j,:}) \end{bmatrix} \tag{2.80}$$

The definition of $\mathbb{DV}_{3D}$ is in definition 4.

Lemma 3 *For a spatial convolutional operator $f_{conv}(\cdot)$ with its input vector, weight and output vector denoted as $B_i^l \in \mathbb{R}^{b\times C_i^l \times H_i^l \times W_i^l}$, $W^l \in \mathbb{R}^{C_o^l \times C_i^l \times k_1^l \times k_2^l}$ and $B_o^l \in \mathbb{R}^{b\times C_o^l \times H_o^l \times W_o^l}$ respectively, it can be converted to a matrix multiplication $g_{fc}(\cdot)$ by converting the original input vector, output vector and weight to corresponding 2D matrices denoted as $M_i^l \in \mathbb{R}^{(bH_o^l W_o^l)\times(C_i^l k_1^l k_2^l)}$, $M_o^l \in (bH_o^l W_o^l) \times C_o^l$ and $W_M^l \in \mathbb{R}^{(C_i^l k_1^l k_2^l)\times C_o^l}$. Formally, the conversion operator $\underset{conv\to fc}{\mathbb{C}} : f_{conv}(\cdot) \to g_{fc}(\cdot)$ is:*

$$\underset{conv\leftrightarrow fc}{\mathbb{C}} \left(f_{conv}(B_i^l, W^l) \right) = g_{fc}(M_i^l, W_M^l) \tag{2.81}$$

Fig. 2.6 is the explanation about how to generate M_i^l, M_o^l, W^l. The M_i^l is generated by two steps. First, flatten each input patch of the convolutional operator to a vector (as in Fig. 2.6(c)). Second, stack all the flattened vectors (as in Fig. 2.6(c)(d)). Similarly, the M_o^l is also generated by two steps. First, flatten the 3D matrix along the output channel into a vector (as in Fig. 2.6(c)). Second, stack all the flattened vectors sequentially(as in Fig. 2.6(c)(d)). For the W_M^l, it is a 2D matrix that is generated by stacking the flattened filters (as in Fig. 2.6(c)). The definitions of $f_{conv}(\cdot)$ and $g_{fc}(\cdot)$ are in definition 5 and definition 6, respectively.

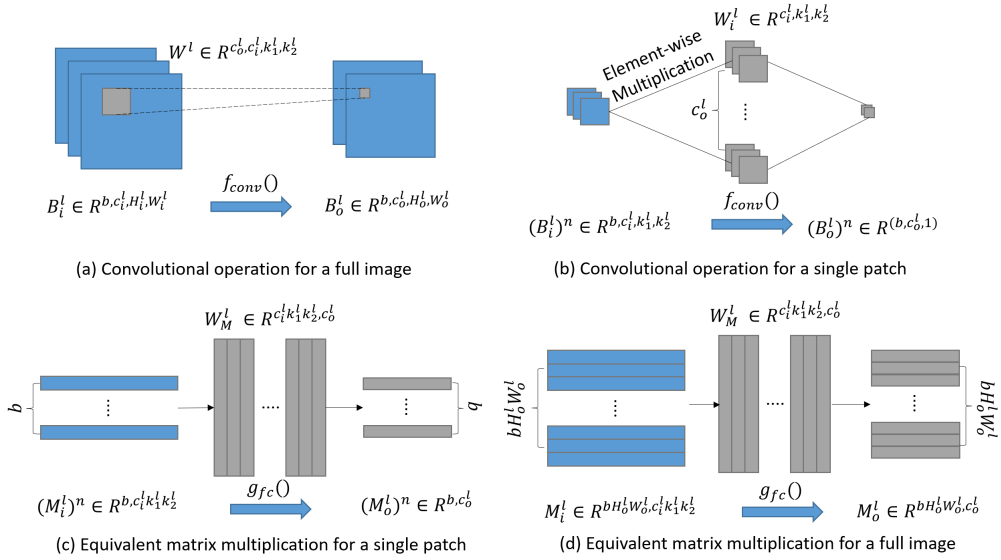


Figure 2.6: Converting a convolutional operation to its equivalent matrix multiplication operation.

Proof 2 *Fig. 2.6 illustrates the process of the conversion from $f_{conv}(\cdot)$ to $g_{fc}(\cdot)$. First of all, as shown in Fig. 2.6(a), the convolutional operation for a full image is implemented by moving the kernels along with the width & height directions with stride s. In each step, the element-wise multiplication will be carried out between a filter and an input patch as shown in Fig. 2.6(b).*

If we flatten the pair of the multiplier of the element-wise multiplication (i.e., a filter and an input patch) as vectors, these vectors can be stacked together as 2D matrix as shown in Fig. 2.6(c). And it can be observed that the convolutional operator $f_{conv}(\cdot)$ for an input patch in Fig. 2.6(b) can be converted to the matrix multiplication operator $g_{fc}(\cdot)$ in Fig. 2.6(c).

Besides, as such a convolution operation on an input patch will be executed $H_o^l \times W_o^l$ times along with the kernel's scrutiny, the convolutional operation for a full image (Fig. 2.6(a)) can be regarded as implementing the equivalent matrix multiplication for $H_o^l \times W_o^l$ times (Fig. 2.6(c)). By stacking all the input and output vectors (Fig. 2.6(d)), the convolutional operation for the full image can be regarded as implementing one-shot matrix multiplication with the input matrix $M_i^l \in \mathbb{R}^{(bH_o^l W_o^l) \times (C_i^l k_1^l k_2^l)}$, output matrix $M_o^l \in (bH_o^l W_o^l) \times C_o^l$ and

weight matrix $W_M^l \in \mathbb{R}^{(C_i^l k_1^l k_2^l) \times C_o^l}$. It should also be noted that the reshaping of the kernel and output vector can also be regarded as applying the dimensionality reduction operator (see definition 7) on W^l and H_o^l along the output channel, respectively. Besides, the 2D output matrix M_o^l can also be easily reshaped as (b, C_o^l, H_o^l, W_o^l) by using the dimensionality addition operator as in definition 8.

It should be noted that, by converting the spatial convolution operation to the matrix multiplication, which is the main operation in the FC layer, the Hessian calculation method for the FC layer in lemma 2 can be adopted to compute the Hessian of the Conv layer. The detailed calculation procedures are in lemma 4.

Lemma 4 *For a convolutional layer, given the activation function f, the input value B_i^l and the output value of after activation function B_o^l, the Hessian of the weight matrix W^l can be calculated recursively as following steps:*

$$\mathbf{H}^l = \underset{2D \to 4D}{\mathbb{DA}}(\hat{\mathbf{H}}^l) \tag{2.82}$$

$\hat{\mathbf{H}}^l$ denotes the Hessian of the equivalent FC layer, which can be calculated as lemma 2:

$$\hat{\mathbf{H}}^l = \text{diag}\left((M_o^l)^2 \otimes \hat{H}^l\right) \tag{2.83}$$

where the pre-activation Hessian $\hat{H}^l$ can be updated iteratively:

$$\hat{H}^l = (\hat{B}^l)^2 \circ \left(\left((W_M^{l+1})^\top\right)^2 \hat{H}^{l+1}\right) + \hat{D}^l \tag{2.84}$$

with $\hat{B}^l$ and $\hat{D}^l$ are defined as:

$$\hat{B}^l = f'(M_o^l), \quad \hat{D}^l = f''(M_o^l) \circ \frac{\partial L}{\partial a_M^{l+1}} \tag{2.85}$$

where $\circ$ represents the element-wise multiplication. $M_i^l, M_o^l, W_M^l, a_M^{l+1}$ are the input, output, weight, activation value of the equivalent FC layer by applying the conversion operator $\underset{conv \to fc}{\mathbb{C}} : f_{conv}(\cdot) \to g_{fc}(\cdot)$ as in lemma 3

The equivalent pre-activation Hessian $\hat{H}^l$ can also be converted to the Conv type by applying the dimensionality addition operator $\underset{2D \to 4D}{\mathbb{DA}}$:

$$H^l = \underset{2D \to 4D}{\mathbb{DA}}(\hat{H}^l) \tag{2.86}$$

The pre-activation Hessian H^l will be imparted along with the back-propagation process and initialized as the second-order derivative of the loss function to the output of the neural network.

Proof 3 *As explained before, one main challenge for the Hessian calculation of a Conv layer is the indirect convolution operation. By converting the spatial convolution operation to the matrix multiplication, the Hessian calculation method for FC layer in lemma 2 can be used to calculate the Hessian of the Conv layer. The procedures are summarized as follows:*

1. *Convert the convolutional operation to its equivalent matrix multiplication operation by applying the conversion operator* $\underset{conv\to fc}{\mathbb{C}} : f_{conv}(\cdot) \to g_{fc}(\cdot)$ *as in lemma 3. The generated equivalent input, output, weight and activation values are denoted as* $M_i^l, M_o^l, W_M^l, a_M^{l+1}$*, respectively.*

2. *Calculate the Hessian of the equivalent FC layer according to lemma 2:*

$$\hat{\mathbf{H}}^l = \text{diag}\left((M_o^l)^2 \otimes \hat{H}^l\right) \tag{2.87}$$

where $\otimes$ *stands for Kronecker product.* $\hat{H}^l$ *is the pre-activation Hessian and can be updated as:*

$$\hat{H}^l = (\hat{B}^l)^2 \circ \left(\left((W_M^{l+1})^\top\right)^2 \hat{H}^{l+1}\right) + \hat{D}^l \tag{2.88}$$

with $\hat{B}^l$ *and* $\hat{D}^l$ *are defined as:*

$$\hat{B}^l = f'(M_i^l), \quad \hat{D}^l = f''(M_i^l) \circ \frac{\partial L}{\partial a_M^{l+1}} \tag{2.89}$$

where $\circ$ *represents the element-wise multiplication.*

3. *Reshape the 2D matrices Hessian* $\hat{\mathbf{H}}^l$ *and pre-activation Hessian* $\hat{H}^l$ *to the Conv type by applying the dimensionality addition operator* $\underset{2D\to 4D}{\mathbb{DA}}$ *as in definition 8:*

$$\mathbf{H}^l = \underset{2D\to 4D}{\mathbb{DA}}(\hat{\mathbf{H}}^l), \quad H^l = \underset{2D\to 4D}{\mathbb{DA}}(\hat{H}^l) \tag{2.90}$$

Another challenge of Hessian calculation is the flattened operation for the input features of the fully-connected layers at the tail of the networks (see Fig. 2.5). It is necessary to convert the pre-activation Hessian information H_{fc}^l *from the FC layer to its adjacent Conv layer* H^l*. The dimensionality addition operator* $\underset{2D\to 4D}{\mathbb{DA}}$ *should be enforced on* H_{fc}^l*:*

$$H^l = \underset{2D\to 4D}{\mathbb{DA}}(H_{fc}^l) \tag{2.91}$$

then the H^l *can be used as the pre-activation Hessian of the Conv layer.*

It should be noted that lemma 4 illustrates the detailed procedures to calculate the Hessian referred to a single data point (i.e., $b = 1$). If the number of data points is more than 1 (i.e., $b > 1$), the Hessian is calculated as the average of the Hessian of an individual data point.

2.2.3. COMPUTE THE HESSIAN OF RECURRENT LAYER

RECURRENT NEURAL NETWORK

The recurrent neural network (RNN) is mainly used to process sequential data and memorize information by including the recurrent connections in the model. The recurrent

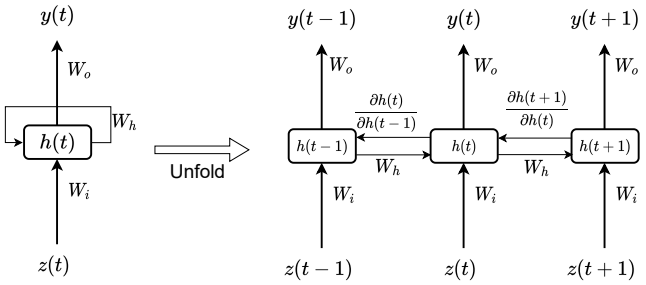


Figure 2.7: An unrolled RNN layer.

connection is embodied by the feedback loop of each hidden neuron, which will be cycled over a sequence (time). The function of memorizing information of the recurrent connection makes the RNN applicable to handwriting recognition tasks [25] and speech recognition tasks [13].

As shown in the left part of Fig. 2.7, an unrolled RNN layer consists of three basic components, the input weight matrix W_i, output weight matrix W_o and hidden weight matrix W_h. Given $f(\cdot), g(\cdot)$ as the activation functions, $z(t)$, $h(t)$ and $y(t)$ as the input, hidden state and output of the time step t, The behaviour of this RNN layer can be described by:

$$h(t) = f\left(\bar{h}(t)\right) = f(W_i z(t) + W_h h(t-1)) \tag{2.92}$$

$$y(t) = g(W_o h(t)) \tag{2.93}$$

Eq. (2.92) reveals that the output of the hidden layer is updated based on both the input vector $z(t)$ and the previous hidden states $h(t-1)$. A RNN is trained by the backpropagation through time (BPTT) method, where the network is unfolded in time and weights are updated based on an accumulation of gradients across time steps. The unfolded process is shown in Fig. 2.7.

HESSIAN CALCULATION

The challenge of the Hessian calculation for a RNN layer comes from the recurrent operation, where the W_i, W_h, W_o will be revisited iteratively through time. This diffres from the FC layer and Conv layer, where the weight matrices only participate once-through operation in a forward propagation process. As the unrolled RNN layer can be unfolded as a fully connected neural network, the Hessian calculation for a RNN layer can be regarded as the calculation of its equivalent FC neural network. Inspired by lemma 2, we propose a recursive and efficient method to compute the Hessian of a recurrent layer as follows:

Lemma 5 *For a recurrent layer, given $f(\cdot)$ representing the activation function, τ representing backward propagation time horizon, T representing the number of data samples, $z(t)$, $h(t)$ and $y(t)$ representing the input, hidden state and output at the time step t, W_i, W_h and W_o representing the weight matrix of the input layer, hidden layer and output layer, the Hessian of W_i, W_h, W_o within the RNN layer can be calculated as follows:*

The Hessian for W_o is:

$$\mathbf{H}_o = \frac{1}{T}\sum_{t=1}^{T}\mathbf{H}_o^{\top}, \quad \mathbf{H}_o^{\top} = (h(t))^2 \otimes H_o^{\top} \tag{2.94}$$

where $H_o^{\top}$ is the pre-activation Hessian. $\otimes$ stands for Kronecker product.

The Hessian for W_h is:

$$\mathbf{H}_h = \mathbb{E}\left(\sum_{t=1}^{T}\sum_{j=\max(1,t-\tau+1)}^{t}\mathbf{H}_h^{t,j}\right) \tag{2.95}$$

$$\mathbf{H}_h^{t,j} = (h(t-1))^2 \otimes H_h^{t,j}, \quad H_h^{t,j} = B_h^2 \circ \left(\left((W_h)^{\top}\right)^2 H_h^{t,j+1}\right) + D_h \tag{2.96}$$

where $B_h = f'(\bar{h}(j)), D_h = f''(\bar{h}(j)) \circ \frac{\partial L}{\partial h(j)}$.

The Hessian for W_i is:

$$\mathbf{H}_i = \mathbb{E}\left(\sum_{t=1}^{T}\sum_{k=\max(1,t-\tau+1)}^{t}\mathbf{H}_i^{t,k}\right) \tag{2.97}$$

$$\mathbf{H}_i^{t,k} = (z(k))^2 \otimes H_i^{t,k} \quad H_i^{t,k} = \prod_{j=k+1}^{t} B_i^2 \circ \left(\left((W_i)^{\top}\right)^2 H_i^{j-1,j}\right) \tag{2.98}$$

where $B_i = f'(\bar{h}(j))$ The above Hessian process can be calculated through a backward propagation through time (BPTT) process.

Proof 4 *As explained before, a RNN layer is normally consist of three matrices, i.e., W_i, W_o, W_h, which will be revisited many times for a complete recurrent operation. The Hessian calculation for a RNN layer can be divided into three parts to calculate the Hessian of W_i, W_o, W_h, respectively. The procedures are summarized as follows:*

1. *Extend the RNN layer to its equivalent FC layers through sequence t (see Fig. 2.7).*
2. *Calculate the Hessian for W_o: If a RNN layer is unfolded through time, then W_o can*

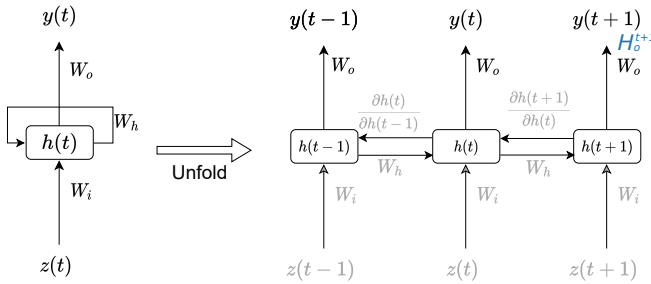


Figure 2.8: The equivalent FC layers about output weight.

be regarded as the weight matrix for a FC layer, whose input is $h(t)$ and output is $y(t)$. As explained in Eq. (2.93), the output $y(t)$ is computed by applying the activation

function $g(\cdot)$ on the matrix multiplication between $h(t)$ and W_o. Such matrix multiplication is implemented T times for a complete recurrent operation (as shown in the unblurred part in Fig. 2.8). According to the Hessian calculation method of the FC layers as in lemma 2, the Hessian for W_o is:

$$\mathbf{H}_o = \frac{1}{T}\sum_{t=1}^{T}\mathbf{H}_o^{\top}, \quad \mathbf{H}_o^{\top} = (h(t))^2 \otimes H_o^{\top} \tag{2.99}$$

where $H_o^{\top}$ is the initialized pre-activation Hessian of W_o. It should be noted that $H_o^{\top}$ will be updated along the BPTT process and be used as the initialized pre-activation Hessian for W_h and W_i.

$$H_h^{t,t} = H_i^{t,t} = (B)^2 \circ \left(\left((W_o)^{\top}\right)^2 H_o^t\right) + D \tag{2.100}$$

with B and D are defined as:

$$B = g'(M_i), \quad D = g''(M_i) \circ \frac{\partial L}{\partial y(t)} \tag{2.101}$$

3. *Calculate the Hessian for W_h: As shown in Fig. 2.9, the W_h can be regarded as the*

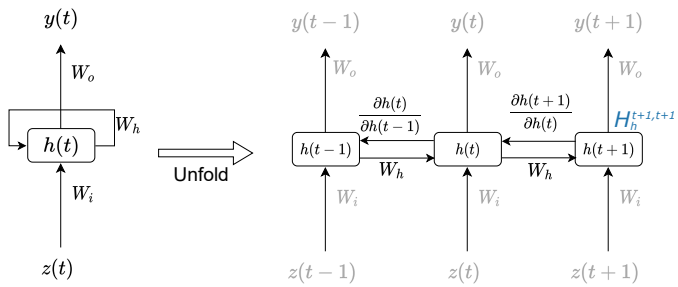


Figure 2.9: The equivalent FC layers about hidden weight.

weight matrix for a FC layer, whose input is $h(t-1)$ and output is $h(t)$. As explained in Eq. (2.92), one component of $h(t)$ is the matrix multiplication between $h(t-1)$ and W_h. Such matrix multiplication is implemented $\min(t,\tau)$ times for the time step t, where τ is the backward propagation time horizon, (as shown in the unblurred part in Fig. 2.9). Therefore, if we only calculate the Hessian of W_h refer to a single data sample at time t, then the Hessian can be calculated by averaging τ individual Hessians as follows:

$$\mathbf{H}_h = \mathbb{E}\left(\sum_{j=\max(1,t-\tau+1)}^{t}\mathbf{H}_h^{t,j}\right) \tag{2.102}$$

If we consider the total time steps T, the Hessian for W_h is:

$$\mathbf{H}_h = \mathbb{E}\left(\sum_{t=1}^{T}\sum_{j=\max(1,t-\tau+1)}^{t}\mathbf{H}_h^{t,j}\right) \tag{2.103}$$

where

$$\mathbf{H}_h^{t,j} = (h(t-1))^2 \otimes H_h^{t,j}, \quad H_h^{t,j} = B_h^2 \circ \left(\left((W_h)^\top \right)^2 H_h^{t,j+1} \right) + D_h \tag{2.104}$$

where $B_h = f'(\bar{h}(t-1)), D_h = f''(\bar{h}(t-1)) \circ \frac{\partial L}{\partial h(t)}$. $H_h^{t,j}$ *is the pre-activation Hessian whose initialized value is calculated by Eq.* (2.100).

4. *Calculate the Hessian for* W_i*: As shown in Fig. 2.10, the* W_i *can be regarded as the*

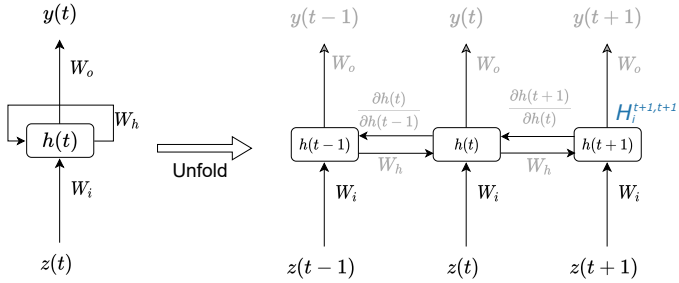


Figure 2.10: The equivalent FC layers about input weight.

weight matrix for a FC layer, whose input is $z(t)$ *and output is* $h(t)$*. Similar to the calculation method for* W_h*, the Hessian of* W_i *can be calculated by averaging* $\min(t,\tau) \times T$ *individual Hessians as follows:*

$$\mathbf{H}_i = \mathbb{E}\left(\sum_{t=1}^{T} \sum_{k=\max(1,t-\tau+1)}^{t} \mathbf{H}_i^{t,k} \right) \tag{2.105}$$

where the individual Hessian is updated as:

$$\mathbf{H}_i^{t,k} = (z(k))^2 \otimes H_i^{t,k} \quad H_i^{t,k} = \prod_{j=k+1}^{t} B_i^2 \circ \left(\left((W_i)^\top \right)^2 H_i^{j-1,j} \right) \tag{2.106}$$

where $B_i = f'(\bar{h}(j))$ *It should be noted that the initialized pre-activation Hessian* $H_i^{t,t}$ *is calculated by Eq.* (2.100).

2.3. CONCLUSION

In this chapter, the Bayesian treatment of the system identification problem is presented. To extend the generalization ability, the Bayesian approach, we propose three different sparse Bayesian deep learning algorithms, which can employ single prior, group prior and fused prior on model parameters. By extracting the diagonal values of the Hessian matrix, we present the detailed derivation of the Hessian calculation methods, which can address the indirect operations within the DNNs effectively, such as convolutional operation and recurrent operation. In Chapter 3 4 5, we will explore how to apply the presented SBDL algorithms and the Hessian calculation methods for different system identification tasks. The extension to high dimensional data is in Chapter 6.

BIBLIOGRAPHY

[1] Neculai Andrei. "Diagonal Approximation of the Hessian by Finite Differences for Unconstrained Optimization." In: *Journal of Optimization Theory & Applications* 185.3 (2020).

[2] Aleksandar Botev. "The Gauss-Newton matrix for Deep Learning models and its applications". PhD thesis. UCL (University College London), 2020.

[3] Aleksandar Botev, Hippolyt Ritter, and David Barber. "Practical Gauss-Newton Optimisation for Deep Learning". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 557–565.

[4] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[5] Yann Dauphin et al. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: *arXiv preprint arXiv:1406.2572* (2014).

[6] Ross B. Girshick. "Fast R-CNN". In: *CoRR* abs/1504.08083 (2015). arXiv: 1504.08083. URL: http://arxiv.org/abs/1504.08083.

[7] Ross B. Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *CoRR* abs/1311.2524 (2013). arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524.

[8] Ethan Goan and Clinton Fookes. "Bayesian Neural Networks: An Introduction and Survey". In: *Case Studies in Applied Bayesian Data Science: CIRM Jean-Morlet Chair, Fall 2018*. Cham: Springer International Publishing, 2020, pp. 45–87. ISBN: 978-3-030-42553-1. DOI: 10.1007/978-3-030-42553-1_3. URL: https://doi.org/10.1007/978-3-030-42553-1_3.

[9] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[10] Gao Huang et al. "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[11] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105. URL: http://dl.acm.org/citation.cfm?id=2999134.2999257.

[13] Xiangang Li and Xihong Wu. “Constructing Long Short-Term Memory based Deep Recurrent Neural Networks for Large Vocabulary Speech Recognition”. In: *CoRR* abs/1410.4281 (2014). arXiv: 1410.4281. URL: http://arxiv.org/abs/1410.4281.

[14] Wei Ma and Jun Lu. *An Equivalence of Fully Connected Layer and Convolutional Layer*. 2017. arXiv: 1712.01252 [cs.LG].

[15] David J. C. MacKay. “Bayesian Interpolation”. In: *Neural Computation* 4.3 (1992), pp. 415–447. DOI: 10.1162/neco.1992.4.3.415. URL: https://doi.org/10.1162/neco.1992.4.3.415.

[16] David JC MacKay. “A practical Bayesian framework for backpropagation networks”. In: *Neural computation* 4.3 (1992), pp. 448–472.

[17] James Martens et al. “Deep learning via hessian-free optimization.” In: *ICML*. Vol. 27. 2010, pp. 735–742.

[18] James Martens, Jimmy Ba, and Matt Johnson. “Kronecker-factored curvature approximations for recurrent neural networks”. In: *International Conference on Learning Representations*. 2018.

[19] James Martens and Roger Grosse. “Optimizing neural networks with kronecker-factored approximate curvature”. In: *International conference on machine learning*. PMLR. 2015, pp. 2408–2417.

[20] Jorge Nocedal. “Updating quasi-Newton matrices with limited storage”. In: *Mathematics of computation* 35.151 (1980), pp. 773–782.

[21] Jorge Nocedal and Stephen J. Wright. “Trust-Region Methods”. In: *Numerical Optimization*. New York, NY: Springer New York, 2006, pp. 66–100. ISBN: 978-0-387-40065-5. DOI: 10.1007/978-0-387-40065-5_4. URL: https://doi.org/10.1007/978-0-387-40065-5_4.

[22] Jason Palmer, Bhaskar D. Rao, and David P. Wipf. “Perspectives on Sparse Bayesian Learning”. In: *Advances in Neural Information Processing Systems 16*. Ed. by S. Thrun, L. K. Saul, and B. Schölkopf. MIT Press, 2004, pp. 249–256. URL: http://papers.nips.cc/paper/2393-perspectives-on-sparse-bayesian-learning.pdf.

[23] Boris T Polyak. “Some methods of speeding up the convergence of iteration methods”. In: *Ussr computational mathematics and mathematical physics* 4.5 (1964), pp. 1–17.

[24] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[25] Hasim Sak, Andrew W. Senior, and Françoise Beaufays. “Long short-term memory recurrent neural network architectures for large scale acoustic modeling”. In: *INTERSPEECH*. 2014.

[26] Michael E. Tipping. “Bayesian Inference: An Introduction to Principles and Practice in Machine Learning”. In: *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 41–62. ISBN: 978-3-540-28650-9. DOI: 10.1007/978-3-540-28650-9_3. URL: https://doi.org/10.1007/978-3-540-28650-9_3.

[27] Wei Wen et al. “Learning structured sparsity in deep neural networks”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 2074–2082.

[28] Alan L. Yuille. and Anand Rangarajan. “The Concave-Convex Procedure (CCCP)”. In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. NIPS’01. Vancouver, British Columbia, Canada: MIT Press, 2001, pp. 1033–1040.

[29] Matthew D Zeiler. “Adadelta: an adaptive learning rate method”. In: *arXiv preprint arXiv:1212.5701* (2012).

[30] Hongpeng Zhou et al. “Bayesnas: A bayesian approach for neural architecture search”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 7603–7613.

[31] Hongpeng Zhou et al. “Sparse Bayesian Deep Learning for Dynamic System Identification”. In: *arXiv preprint arXiv:2107.12910* (2021).

[32] Mingfa Zhu, John Lawrence Nazareth, and Henry Wolkowicz. “The quasi-Cauchy relation and diagonal updating”. In: *SIAM Journal on Optimization* 9.4 (1999), pp. 1192–1204.

3

THE ART OF PRIOR I: SINGLE PRIOR

This chapter implements the proposed sparse Bayesian deep learning algorithm with single prior on the identification of a gene regulatory network. Identifying the gene regulatory network is a central topic for systems biology as it captures the interaction between genes and other cell substances (e.g., protein). This chapter presents how to use the proposed Bayesian approach to identify both the topology and parameters of a repressilator model, which is described by nonlinear ordinary difference equations involving polynomial and rational functions. We designed a novel model structure consisting of a nonlinear Fully-connected neural network and a linear neural network. The activation function of the nonlinear neural network is replaced by the Hill function with a typical structure, which is assumed as the prior information. With such model design, an initialized model can be regarded as a super-graph with a redundant structure, whose sub-graph can represent the underlying mathematical model. The sparse Bayesian deep learning algorithm with the single prior is implemented to identify the redundancy. The proposed approach is demonstrated by the accurate identification of both model structure and parameters.

3.1. INTRODUCTION

The identification of both topology and parameters of a biology network is an important research topic in synthetic biology. As a classical synthetic oscillatory network, the repressilator model is a classical nonlinear dynamical system to represent mRNA transcription and and protein translation dynamics. The repressilator model can be described by nonlinear ordinary differential equations (ODEs) involving polynomial and rational functions. The identification of the topology and parameters of the repressilator model is a typical research problem that has attracted a lot of attention. The sustained oscillation of messenger RNAs (mRNA) and proteins is decided by several important parameters [3] (e.g., the translation rate, the transcription rate of repressor concentration, and the decay rates of the protein and mRNA).

The identification of the repressilator, especially the parameters of the Hill function, which decides the repression curve describing the transcriptional repression and is a special case of a rectangular hyperbola with its typical form expressed as Eq. (3.7). Several previous works have presented approach to identify the Hill function. To name a few examples, [1] focused on identifying the interconnection topology of biological and biochemical systems. It presented a network determination algorithm that can determine the interaction topology of a gene regulatory network by treating the model descriptions with polynomial and rational functions. Based on the convex relaxations of the sparsity and stability constraints, [7, 20] developed an algorithm to infer stable genetic networks from the steady-state measurements around the equilibrium state. With the linear programming formulation of the identification task, the proposed algorithm is verified by successful parameter selection on both simulated data and experimental data. By assuming that the dynamics of the system can be described as a linear combination of a finite set of candidate dictionary functions, [13–15] cast the identification problem as a sparse linear regression problem that could be addressed from a Bayesian viewpoint. Although the proposed method could identify both nonlinear structure and kinematic parameters of genetic regulatory networks, its effectiveness relied on the proper selection of basis functions. The modelling process is more like "selecting" the relevant terms from pre-defined alternative functions instead of "learning" from data It cannot identify the accurate physical models if the correct nonlinear/linear items are not included in the set of basis functions. [13] also stated that only an approximated model could be obtained if the true function is not included in the considered set of dictionary functions.

Following these works, the main focus of this chapter is to identify the model topology and parameters from data without a pre-defined set of basis functions. The proposed method includes two parts. First, a combined neural network is designed to approximate the underlying repressilator model. To design a proper model structure, some prior knowledge is required to be known. First, it is assumed to be known that the repressilator model includes both nonlinear and linear functions. Second, the typical structure of the Hill function is supposed to be known (see Eq. (3.7)). Based on the prior information, the model is designed to combine a linear sub-network and a nonlinear sub-network. And the Hill function is adopted as the activation function in the nonlinear sub-network. Second, with such model design, the identification problem can be cast as a sparse optimization problem which can be addressed with the proposed sparse Bayesian deep learning (SBDL) algorithm. An initialized combined neural network can be regarded as an over-

parametrized model which includes extra input and a more complex Hill function. The underlying model can be represented by removing the redundancy from the super-graph. Besides, as the designed model structure is the same as a deep neural network, the sparse regression techniques [4, 9, 18] for DNNs can also be implemented in this chapter. One classical approach is the ℓ_1 norm regularization method [5, 16], which adds $\|W\|_{\ell_1}$ as the penalty item to promote the model sparsity. However, the practical implementation showed that this method is always non-sparse which requires a lot of tuning effort. Inspired by [13–15], we propose to treat such a sparse regression problem from a Bayesian viewpoint. The Bayesian approach has already been proved to be a practical ways to address the network compression problem [10], enabling a faster search through the parameter space [11]. In this work, the proposed Bayesian approach can incorporate the non-structural sparsity as priors and result in a sparser solution. The experiment result in Section 3.3 verifies that the proposed approach can effectively identify the topology and parameters of the repressilator model.

The organization of this chapter is as follows. Section 3.2 introduces the identification method. Section 3.3 presents the identification results on the simulated dataset. Finally, section 3.4 concludes the chapter.

3.2. IDENTIFICATION METHOD

3.2.1. MODEL DESIGN

The discrete-time mathematical description of the kinetics of the repressilator can be represented by following six coupled difference equations:

$$m_1(t+1) = m_1(t) + dt \cdot \left[-\gamma_1 m_1(t) + \frac{a_1}{K_1 + p_6^n(t)} \right], \tag{3.1}$$

$$m_2(t+1) = m_2(t) + dt \cdot \left[-\gamma_2 m_2(t) + \frac{a_2}{K_2 + p_4^n(t)} \right], \tag{3.2}$$

$$m_3(t+1) = m_3(t) + dt \cdot \left[-\gamma_3 m_3(t) + \frac{a_3}{K_3 + p_5^n(t)} \right], \tag{3.3}$$

$$p_4(t+1) = p_4(t) + dt \cdot \left[-c_1 p_4(t) + \beta_1 m_1(t) \right], \tag{3.4}$$

$$p_5(t+1) = p_5(t) + dt \cdot \left[-c_2 p_5(k) + \beta_2 m_2(t) \right], \tag{3.5}$$

$$p_6(t+1) = p_6(t) + dt \cdot \left[-c_3 p_6(t) + \beta_3 m_3(t) \right]. \tag{3.6}$$

where Eq. (3.1)- (3.3) denote the transcription dynamics and Eq. (3.4)- (3.6) stand for the translation dynamics. m_1, m_2, m_3 denote the mRNA concentrations. p_4, p_5, p_6 are the protein concentrations. The nonlinear terms in Eq. (3.1)- (3.3) are the Hill functions which decide the repression curve and describe the transcriptional repression. n is the Hill coefficient, which is a measure of the sleep of the response curve. $\gamma_1, \gamma_2, \gamma_3$ denote the mRNA decay rates. K_1, K_2, K_3 are the apparent dissociation constant derived from the law of mass action. a_1, a_2, a_3 denote the maximum promoter strength of the gene. In Eq. (3.4)- (3.6), the ratio of protein production rate caused by mRNA decay rate are represented by $\beta_1, \beta_2, \beta_3$. c_1, c_2, c_3 denote the protein decay rates. dt is the discretization time step. From these equations, it can be found that the mRNA concentrations are rescaled by translation efficiency. Typically, the repressilator model is a simplification of the transcriptional regu-

lation. The oscillation behaviour can be observed and obtained by numerical integration with predefined parameters (e.g.,$\forall i:\ K_i = 1, a_i = 3.2, \gamma_i = 0.31, \beta_i = 1.51, c_i = 0.31, dt = 1$ in this chapter).

The six equations in the repressilator model can be split as nonlinear (first three equations) and linear (the other three equations) parts. Therefore, the trained network model can be designed as a combination of linear sub-network and nonlinear sub-network as shown in Fig. 3.1, where the nonlinear sub-network consists of hidden layers, and the linear part has no hidden layer. It should be noted that the activation function of the nonlinear sub-network is the typical form of the Hill function as in Eq. (3.7).

$$\text{hill}\ (x_{i,t},\ k_1,\ k_2,\ n) \triangleq \frac{k_1}{k_2 + (\sum_{j=1}^{N} \sum_{i=1}^{6} W_{ji} x_{i,t})^n} \tag{3.7}$$

where W_{ji} denotes the weight of the connection between i-th input feature and j-th hidden neuron. N represents the amount of hidden neurons in the hidden layer. The objective is to estimate the parameters k_1, k_2, n, W_{ji}, in which n is the most important model parameter which has a critical impact on the dynamics of the transcription process. For this task, the typical structure of the Hill function is assumed to be known as prior knowledge. To simplify the notation, we use $x_1, x_2, x_3, x_4, x_5, x_6$ to represent $m_1, m_2, m_3, p_1, p_2, p_3$. The input of the model is the vector $X = [x_1, \cdots, x_6]$. The output target is defined as $y_i(k+1) = x_i(k+1) - x_i(k)$, where $i = 1, 2, \cdots, 6$.

3.2.2. LEARNING IN A BAYESIAN FRAMEWORK

Using the designed model to approximate the given dataset can be denoted as training $\texttt{Net}(W, X)$, where W represents an array of all inferred parameters in the network, including the weight matrices in these two sub-networks and the parameters within the Hill function k_1, k_2, n. The prediction model is defined as $\hat{y}_i(t) = \texttt{Net}(W, X(t))$. The way to formulate the system identification problem using a Bayesian framework can be referred to Section. 2.1.1 of Chapter 2. The algorithm adopted in this chapter is the proposed SBDL approach with the single prior, as explained in Section. 2.1.2 of Chapter 2.

3.3. EXPERIMENT

3.3.1. EXPERIMENT SETTING

According to Eq. (3.1)- (3.6), the simulated dataset is generated under the specific parameters setting, i.e., $\forall i:\ K_i = 1, a_i = 3.2, \gamma_i = 0.31, \beta_i = 1.51, c_i = 0.31, dt = 1$. The mathematical model was simulated two times with different initialized x_i. Both the training and test dataset contains 200 samples. A snapshot of the persistent oscillatory behaviour can be observed in Fig. 3.2.

As explained in Section. 3.2.1, there exists six different models to describe the mRNA transcription and protein translation process. In this example, we also trained six different network models to learn the six mathematical models as in Eq. (3.1)- (3.6). The network structure is defined as a nonlinear part with structure $[6, 10, 1]$ and linear part with structure $[6, 1]$. The 6 is the number of input features. 1 is the number of the output features. 10 is the number of hidden neurons. It should be noted that the number

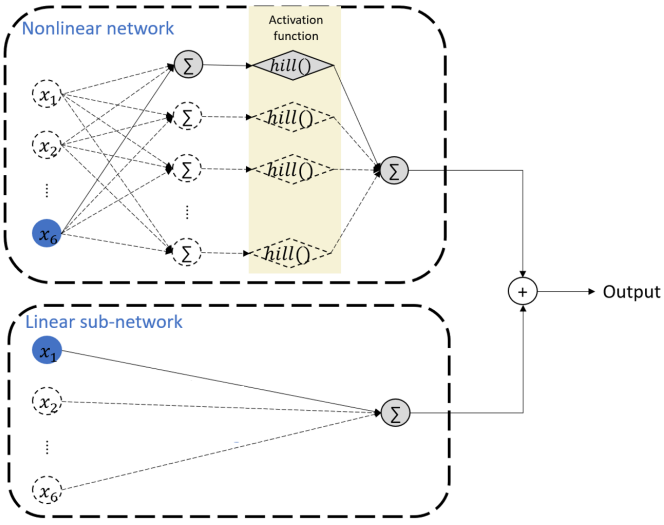


Figure 3.1: The designed combined network architecture consists of a nonlinear sub-network and a linear sub-network. These two sub-networks have the same input features, denoted as yellow circles. The blue circles in the nonlinear sub-network represent the remained neurons after compression. Identified redundant input features and hidden neurons are denoted as dotted circles. The summation of the two sub-networks forms the output of the designed model.

of hidden neurons should be a positive integer greater than 1 with no upper limit. The difficulty of the identification task increases with the increase of the number of hidden neurons. We did not choose 1 in case the identification task is too simple. But limited by computing resources, we also did not choose other larger numbers. We select 10 in this experiment, an experimental setting with moderate identification difficulty. The shape-wise regularization is applied to the model parameters. 10 cycles with 200 epochs in each cycle are implemented during the training process. The pruning strategy consists of two parts: a) dynamic pruning based on gamma, the weight with the magnitude of gamma smaller than 0.058 is forced to be zero every 200 epochs; b) one-shot weight pruning, the weight with magnitude smaller than 0.01 is removed after the whole training process. It should be noted that the threshold $\kappa_\psi = 0.058$ is decided according to the properties of differential entropy for Gaussian distribution. As the prior distribution is assumed to be Gaussian distribution with zero mean value, its differential entropy is $\frac{1}{2}(1 + ln(2\pi\psi_i))$ (see page 54 in Section 1.6 of [2]). It should be noted that the differential entropy can be negative and a distribution with larger entropy is typically regarded with a proper one [17]. Therefore, in this thesis, we only retain the weight whose ψ can make its differential entropy positive, i.e., κ_ψ is set as when $\frac{1}{2}(1 + ln(2\pi\psi_i)) \leq 0$, ($\psi_i \leq 0.0585$.) ψ_i represents the uncertainty of the prior distribution as illustrated in Eq. 2.45 and will be used as a pruning criteria as presented in Algorithm 1 The threshold $\kappa_W = 0.01$ is an empirical value. After the network pruning, the retraining process with several epochs is executed

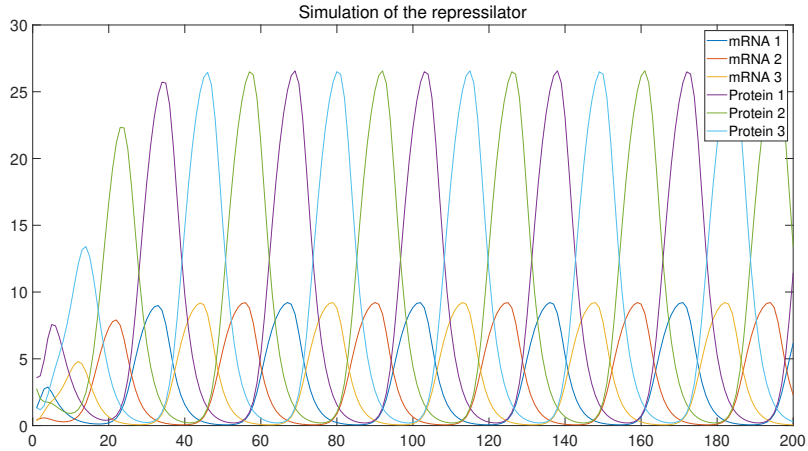


Figure 3.2: A snapshot of persistent oscillatory behavior of a repressilator consisting of 3 genes. The X-axis denotes time, and Y-axis denotes the value/concentration of each state.

to fine-tune the remained model parameters. As explained in Algorithm 1, the λ is also a tuning hyper-parameter that can influence the regularization effect. A larger λ means a stronger regularization on model parameters. It should be noted that λ is an empirical parameter, which cannot be learned automatically in the training process. Therefore, we have to define the value of λ before training. In this experiment, 10 different λ settings ($\lambda = \{1e^{-1}, 1e^{-2}, 1e^{-3}, 1e^{-4}, 1e^{-5}, 1e^{-6}, 1e^{-7}, 1e^{-8}, 1e^{-9}, 1e^{-10}\}$) were simulated. For each λ, the identification procedure is repeated 200 times with differing weight initialization.

3.3.2. EXPERIMENT RESULT

The result of the identified structure is shown in Table. 3.1. First of all, the influential input features for both the linear and nonlinear sub-networks can be identified. For example, the identified result of M_1 is shown in the first column of Table. 3.1 with only x_6 retained for the nonlinear sub-network and x_1 retained for the linear sub-network. This is consistent with the ground truth model of M_1 that is illustrated in Eq. (3.1). The redundant input features for other models can also be identified and removed after compression. Second, it is also worth noting that the Hill coefficient n for model M_1, M_2, M_3 can be identified accurately by implementing fine-tune procedure after the compression. The fine-tune procedure is a typical and conventional training procedure for neural network training tasks, especially network compression [12, 19, 22] and neural architecture search [6, 21]. The “fine-tune" means the identified/trained network will be fine-tuned again for several epochs on the training data using the standard back-propagation process [8]. The typical objective of “fine-tune" is to improve the prediction accuracy by optimizing the model parameters within the identified/learned structure. In this experiment, the fine-tune procedure is also implemented after the compression of the combined neural network. The simulated output of each identified network is almost the same as the ground truth, which

Table 3.1: Identified result for the synthetic gene network

Model	M_1	M_2	M_3	M_4	M_5	M_6
Identified hill function before fine-tune	$\frac{3.1816}{0.9955+x_6^{1.9918}}$	$\frac{3.0480}{0.9542+x_4^{1.9592}}$	$\frac{3.1395}{0.9841+x_5^{1.9775}}$	-	-	-
Identified hill function after fine-tune	$\frac{3.2}{1+x_6^2}$	$\frac{3.2}{1+x_4^2}$	$\frac{3.2}{1+x_5^2}$	-	-	-
Identified linear component	-0.31x_1	-0.31x_2	-0.31x_3	1.51x_1-0.51x_4	1.51x_2-0.51x_5	1.51x_3-0.51x_6

proves the identified structure could represent the intrinsic features of the synthetic gene network.

3.4. CONCLUSION

In this chapter, we present how to apply the sparse Bayesian deep learning algorithm with the single prior to identify the topology and parameters of a repressilator model. With the known prior information about the typical structure of the Hill function, a combined model structure is designed to approximate the underlying governing equations. The non-structured sparsity regularization is implemented on the designed model structure by enforcing the single prior on each parameter. The identified model structures are identical to the true model structures. And the parameters can be learned accurately by retraining the identified structure after the network pruning. The identified model can recover the translation and transcription dynamics precisely in the free-run simulation setting.

BIBLIOGRAPHY

[1] Elias August and Antonis Papachristodoulou. “Efficient, sparse biological network determination”. In: *BMC Systems Biology* 3.1 (2009), pp. 1–13.

[2] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[3] Michael B Elowitz and Stanislas Leibler. “A synthetic oscillatory network of transcriptional regulators”. In: *Nature* 403.6767 (2000), p. 335.

[4] Song Han, Huizi Mao, and William J. Dally. “Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding”. In: *International Conference on Learning Representations*. 2016.

[5] Yihui He, Xiangyu Zhang, and Jian Sun. “Channel pruning for accelerating very deep neural networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 1389–1397.

[6] Hengyuan Hu et al. “Network trimming: A data-driven neuron pruning approach towards efficient deep architectures”. In: *arXiv preprint arXiv:1607.03250* (2016).

[7] Agung Julius et al. “Genetic network identification using convex programming”. In: *IET systems biology* 3.3 (2009), pp. 155–166.

[8] Vadim Lebedev et al. “Speeding-up convolutional neural networks using fine-tuned cp-decomposition”. In: *arXiv preprint arXiv:1412.6553* (2014).

[9] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. “Snip: Single-shot network pruning based on connection sensitivity”. In: *arXiv preprint arXiv:1810.02340* (2018).

[10] Christos Louizos, Karen Ullrich, and Max Welling. “Bayesian compression for deep learning”. In: *arXiv preprint arXiv:1705.08665* (2017).

[11] Xingchen Ma et al. “A bayesian optimization framework for neural network compression”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 10274–10283.

[12] Wei Pan, Hao Dong, and Yike Guo. “Dropneuron: Simplifying the structure of deep neural networks”. In: *arXiv preprint arXiv:1606.07326* (2016).

[13] Wei Pan, Filippo Menolascina, and Guy-Bart Stan. “Online model selection for synthetic gene networks”. In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE. 2016, pp. 776–782.

[14] Wei Pan et al. “A sparse Bayesian approach to the identification of nonlinear state-space systems”. In: *IEEE Transactions on Automatic Control* 61.1 (2015), pp. 182–187.

[15] Wei Pan et al. “Reconstruction of arbitrary biochemical reaction networks: A compressive sensing approach”. In: *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE. 2012, pp. 2334–2339.

[16] Robert Tibshirani. “Regression shrinkage and selection via the lasso”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288.

[17] Alan Watkins. *Maximum-Entropy and Bayesian Methods in Science and Engineering*. 1989.

[18] Wei Wen et al. “Learning structured sparsity in deep neural networks”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 2074–2082.

[19] Zhonghui You et al. “Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks”. In: *arXiv preprint arXiv:1909.08174* (2019).

[20] Michael M Zavlanos et al. “Inferring stable genetic networks from steady-state data”. In: *Automatica* 47.6 (2011), pp. 1113–1122.

[21] Hongpeng Zhou et al. “Bayesnas: A bayesian approach for neural architecture search”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 7603–7613.

[22] Zhuangwei Zhuang et al. “Discrimination-aware channel pruning for deep neural networks”. In: *arXiv preprint arXiv:1810.11809* (2018).

4

THE ART OF PRIOR II: GROUP PRIOR

This chapter implements the proposed sparse Bayesian deep learning algorithm with group prior on the system identification using deep neural networks (DNNs). Although DNNs show impressive approximation ability in various fields, several challenges still exist for system identification problems. First, DNNs are known to be too complex that they can easily overfit the training data. Second, the selection of the input regressors for system identification is nontrivial. Third, uncertainty quantification of the model parameters and predictions are necessary. The proposed Bayesian approach offers a principled way to alleviate the above challenges by marginal likelihood/model evidence approximation and structured group sparsity-inducing priors construction. The identification algorithm is derived as an iterative regularized optimization procedure that can be solved as efficiently as training typical DNNs. Besides, a practical calculation approach based on the Monte-Carlo integration method is derived to quantify the uncertainty of the parameters and predictions. The effectiveness of the proposed Bayesian approach is demonstrated on several linear and nonlinear system identification benchmarks by achieving good and competitive simulation accuracy. The one-step-ahead prediction experiments on a small ratio of datasets are also implemented.

Parts of this chapter have been published in arXiv preprint:1911.06847(2019) [55] and arXiv preprint arXiv:2107.12910(2021) [57].

4.1. INTRODUCTION

System identification (SYSID) has a long history in natural and social sciences [28]. Various approaches have been proposed for both linear or nonlinear systems and static or dynamical processes [1, 5, 8, 9]. Among these, deep neural networks (DNNs) are prominent black-box models and recently regain research interest in the SYSID community [3, 12, 14, 30], thanks to the boom of deep learning.

The deep neural network (DNN) models have their advantages and disadvantages. An early paper on feed-forward NNs proved the universal approximation capabilities of any measurable function, using one hidden layer on a compact set [16]. Several works also achieved competitive results by using feed-forward NNs [26] and recurrent neural networks (RNNs) [10, 48] in the context of dynamical systems. However, it is not easy to design a proper NN structure. First of all, the trade-off between the model complexity and (simulation) prediction accuracy should be considered. An over-simplified model cannot reveal the underlying relation between input data and output data. On the other hand, an over-complex model may overfit the training data, thus reducing its generalization ability. Besides, the inevitable (non-Gaussian and non-additive) noise and non-smooth characteristics of some nonlinear processes may also cause the overfitting problem. Furthermore, NNs can also be underspecified by the data and constitute a large space of hypotheses for high-performing models [53]. Another challenging problem for SYSID is input regressor selection, which is defined as follows: given a set of available input features, select the most relevant ones which can explain the intrinsic phenomenon of the system [7]. An effective feature selection can improve prediction performance, generalization ability and reduce computational [2, 7].

For these challenges, the sparse Bayesian deep learning (SBDL) method offers a principled way to tackle them simultaneously: a) A more efficient exploration of the hypothesis space (corresponding to saddle points) of NN models is possible [53, 56]; b) Over-fitting can be alleviated, and model redundancies can be eliminated through marginalization and by choice of sparsity inducing prior distribution over parameters [31]; c) Important input variables can be selected automatically by imposing structured sparsity on the NN; d) Model parameters and prediction uncertainties can be quantified, which is particularly useful in decision making and safety-critical applications such as autonomous driving and structural health monitoring [17].

Diverse Bayesian SYSID methods have been developed in the last decades. To name a few, a practical sparse Bayesian approach to state-space identification of nonlinear systems was proposed in [38] in the context of biochemical networks. A Bayesian identification algorithm of nonlinear autoregressive exogenous (NARX) models using variational inference with a demonstration on the electroactive polymer was introduced in [22]. A framework for identifying the governing interactions and transition logics of subsystems in cyber-physical systems was presented in [54] by using Bayesian inference and predefined basis functions. A variational expectation maximization approach to SYSID when the data includes outliers was developed in [27]. Two approaches to SYSID using Bayesian networks were proposed in [9]. The first one combines kernel-based stable spline and group Least Angle Regression while the other combines stable splines with the hyper-prior definition in a fully Bayesian model. However, this work did not discuss how to apply the Bayesian approach to the NN model. Another typical probabilistic nonparametric mod-

elling method is the Gaussian process (GP), which can perform excellently for linear and nonlinear SYSID tasks, but suffers from the high computational burden for large datasets and cannot conduct input regressor selection efficiently. Overall, specific to the use of NNs as a model form, little attention has been given to the identification of dynamic systems in a Bayesian framework.

The intractability of exact inference forces a trade-off between accuracy and computational needs [46]. There exists multiple approaches to approximate Bayesian inference, i.e. Laplace approximation [32], expectation propagation [24], variational inference [15]. As explained in Chapter. 2, the Laplace approximation method is adopted in this thesis and a Bayesian iterative algorithm is derived and used repeatedly from different random initial conditions. The efficient Hessian calculation methods for fully connection and recurrent layers are developed, allowing a more efficient neural network exploration.

In this chapter, multi-layer perceptron (MLP) and long short term memory networks (LSTM) are used as model forms. As two typical neural networks for system identification, MLP is mainly can be used to model both static and dynamic systems [34, 35, 39] and, while LSTM is mainly used to model the dynamic systems [10, 11, 23, 39]. To address the challenges such as overfitting the training data and the selection of input regressors, group priors are introduced over network parameters to induce structured sparsity. The simulation error is adopted as the evaluation metric, which is a more challenging criterion compared with one-step-ahead prediction. The simulation error is equivalent to the N-step-ahead prediction error with N denoting a user-defined temporal horizon. The main contributions of this chapter are:

- A practical iterative algorithm using Bayesian deep learning is proposed for SYSID. The first identification cycle of the algorithm is equivalent to the conventional sparse group lasso regularization method. This algorithm can be used with both MLP and LSTM networks for linear and nonlinear processes.
- The structured sparsity is incorporated in the Bayesian formulation of the identification problem to alleviate the overfitting issue and perform the selection of the input regressor. As a consequence, the number of hidden neurons in both MLP and LSTM networks can be significantly reduced.
- The proposed algorithm achieves good and competitive simulation accuracy on five benchmark datasets. The datasets of three linear processes are provided in the MATLAB System Identification Toolbox[1], including the Hairdryer, Heat exchanger, and the Glass Tube manufacturing process. The datasets of two nonlinear processes are provided in the Nonlinear System Identification Benchmarks website[2], including the Cascaded Tanks [45] and Coupled Electric Drives [51].

The organization of this chapter is as follows. Section 4.2 explains the input feature selection, gives a practical Monte-Carlo sampling method to estimate the predictive uncertainty, and illustrates the implementation information. Section 4.3 reports the identification results on benchmark data of linear and nonlinear processes. Finally, Section 4.4 is a discussion of the results, and Section 4.5 concludes the paper.

[1] https://nl.mathworks.com/help/ident/examples.html
[2] https://sites.google.com/view/nonlinear-benchmark/

4.2. IDENTIFICATION METHOD

4.2.1. INPUT FEATURE SELECTION

The objective of this chapter is the identification of dynamic systems. Hence the regressors used as input to these models will be defined as a combination of lagged elements of the system input u and outputs y. The input lag is denoted l_u and output lag l_y, resulting in the expression $z(t+1) = [u(t+1), u(t), \cdots, u(t-l_u), y(t), y(t-1), \cdots, y(t-l_y)]^\top$. The feature selection means to identify and remove the redundant features from $z(t+1)$. The proposed SBDL algorithm can select the input features automatically by imposing sparsity on the neural network. Specifically, the derived iterative procedure in Section. 2.1.3 of Chapter 2 includes an assumption on the independence and non-stationarity of connection weights resulting in a shape-wise regularization as shown in Fig. 2.2(a). This drives the individual connection weight to 0. In some applications, one may want to enforce more structured sparsity by pre-defined groups and re-expressing the regularization term as a function of these groups [49]. This chapter uses a structured regularization of rows and columns (Fig. 2.2(b-c)). The benefit of such an approach, specific to this chapter, is not only obtaining compact sparse models, but also, the suppression of input nodes in z that are deemed less pertinent without loss of accuracy. The reduction in the dimensionality of the input vector z represents the selection of input features.

To extend this approach to the Bayesian framework, one has to revisit the prior formulation. The prior of a weight matrix is formulated based on the designated group of weight matrices (row or column or both). These groups are considered independent, but the connection weights of a specific group share the same prior Gaussian relaxation (see Fig. 2.2(b-c)). For each of the cases shown in Fig. 2.2, the update rules for ψ, ω and the regularization function ρ are given in Table. 2.1.

4.2.2. MAKING UNCERTAIN PREDICTIONS

In the Bayesian procedure, predictions are made using the posterior predictive distribution. It is given by Eq.(4.1):

$$p(\hat{y}|z,\mathscr{D}) = \int p(\hat{y}|W,z)\; p(W|\mathscr{D}) dW \tag{4.1}$$

The first term of the integral is the likelihood of the prediction conditional on the network parameters. The second term is the inferred posterior distribution over the weights W given the training data $\mathscr{D}$.

To find the expected value of the prediction, the expression of the predictive distribution in Eq. (4.1) is used, and given that the likelihood is defined as a Gaussian function one obtains:

$$\mathbb{E}[\hat{y}] = \int \hat{y}\; p(\hat{y}|z,\mathscr{D}) d\hat{y} \tag{4.2}$$

$$= \int \Big(\int \hat{y}\; p(\hat{y}|W,z) d\hat{y} \Big)\; p(W|\mathscr{D}) dW \tag{4.3}$$

$$= \int \texttt{Net}(W,z)\; p(W|\mathscr{D}) dW \tag{4.4}$$

Using the inferred posterior distribution over the weights, one can approximate this integral by Monte-Carlo sampling methods [13, 36]. An unbiased estimate of the prediction is given by the average predictions using W sampled by the posterior M times.

$$\mu_{\hat{\mathbf{y}}} \approx \frac{1}{M} \sum_{m=1}^{M} \texttt{Net}(W(m), z) \tag{4.5}$$

In an analogous way, to estimate the variance in the posterior predictive distribution, the expected value $\mathbb{E}\left[\hat{y}^T \hat{y}\right]$ is analytically derived as follows in Eq. (4.6)- (4.8):

$$\mathbb{E}[\hat{y}^T \hat{y}] = \int \hat{y}^T \hat{y}\ p(\hat{y}|z, \mathscr{D}) d\hat{y} \tag{4.6}$$

$$= \int \Big(\int \hat{y}^T \hat{y}\ p(\hat{y}|W, z) d\hat{y} \Big)\ p(W|\mathscr{D}) dW \tag{4.7}$$

$$= \int \left(\sigma^2 + \texttt{Net}(W, z)^2\right)\ p(W|\mathscr{D}) dW \tag{4.8}$$

An unbiased estimate of the variance is given by Monte-Carlo integration methods [13, 36], with M samples from the inferred posterior distribution of W.

$$\Sigma_{\hat{\mathbf{y}}} \approx \sigma^2 + \frac{1}{M} \sum_{m=1}^{M} \texttt{Net}(W(m), z)^2 - {\mu_{\hat{\mathbf{y}}}}^T \mu_{\hat{\mathbf{y}}} \tag{4.9}$$

This variance (Eq. (4.9)) represents the model uncertainty in prediction. It is approximated by the sum of an aleatoric uncertainty and an epistemic uncertainty. The aleatoric uncertainty is generally known to be irreducible corresponding to the noise covariance of the measurement and is generally incorporated in the likelihood form [13]. The epistemic uncertainty corresponds to the model's uncertainty in a prediction that is often called reducible uncertainty [13] and grows when moving away from the training data [53].

4.2.3. VALIDATION AND SOFTWARE IMPLEMENTATION

The identification experiments are implemented using the PyTorch library on Python. The MLP models are randomly initialized and trained based on a one-step-ahead prediction approach and validated based on a free run simulation setting. The stochastic gradient descent method adopted is the ADAM optimizer, and the learning rate is scheduled using Cosine Annealing for each identification experiment.

For the evaluation metric, it should be noted that prediction and simulation error are two typical evaluation metrics for SYSID. Given the current and past measurements of the system input and output, the prediction means predicting the system response to the future k steps, where k denotes the prediction horizon. Simulation is to predict the system response based only on the input data and initial conditions. Therefore, the simulation error is a more challenging evaluation metric and is used in this paper. The figure of merit used is given by the root mean square error (RMSE) of the simulation experiment:

$$RMSE(\hat{y}, y) = \sqrt{\frac{1}{T} \sum_{i=1}^{T} (y_i - \hat{y}_i)^2} \tag{4.10}$$

Besides, the model sparsity refers to the number of zero-valued parameters divided by the total number of parameters.

4.2.4. ALGORITHM

In this chapter, we introduce a dynamic pruning strategy, which is different from the standard one-shot pruning strategy in Algorithm 2. By comparing the magnitude of weight and uncertainty of weight, the network will be pruned in each cycle, which can achieve more efficient pruning effects empirically. Specifically, the pruning criterium is based on not only the magnitude of weight $\|W_{ab}^l\|$ but also the uncertainty ψ_{ab}^l. Typically, a decrease in ψ_{ab}^l means less regularization on corresponding weight W_{ab}^l. Based on this, the binary matrices C^l are generated as the *masks* of W, which denotes the connection redundancy. C^l has the same dimension as W^l and will be optimized during the training process. The update of C is decided by:

$$C = \begin{cases} 0, & \psi_{ab}^l(k) \geq \kappa_\psi, |W_{ab}^l(k)| \geq \kappa_w \\ 1, & \text{others} \end{cases} \tag{4.11}$$

where 1 denotes the redundancy, and 0 means the weight should be retained. It should be noted that the *masks* C is implemented updated at the last epoch of each cycle. A pseudocode for the iterative procedure is given by Algorithm 4.

4.3. EXPERIMENT

4.3.1. SIMULATION EXPERIMENT

This section summarizes the identification experiments of three linear processes and two nonlinear processes using the proposed algorithm. For linear systems, the identification procedure is repeated $M = 20$ times with $K_{max} = 6$ identification cycles. For nonlinear systems, the identification is also repeated $M = 20$ times but with $K_{max} = 10$ identification cycles each. In Table 4.1, stands a summary of the model structure used for identification as well as the mean, standard deviation and minimum validation RMSE of the M best-generated models, the percentage of sparse parameters in the best-generated model. The benchmarks are described more thoroughly, and the reader is supported with sparsity plots, simulation plots, and plots of posterior predictive mean and uncertainty corresponding to the best-generated model.

Three linear processes are identified, the Hairdryer, corresponding to the PT326 process trainer [18], a Heat exchanger [19] and Glass Tube manufacturing process [20]. The datasets of these processes are provided by Matlab in the corresponding tutorials on linear system identification. The chosen best validated models are compared to the methods used in the corresponding tutorials. Additional model structures used for the identification of the Hairdryer are taken from chapter 17.3 of [29] and run in Matlab. Check 4.3.1.6 Table. 4.2 for the comparisons. Two nonlinear processes, the Cascaded Tanks [45], Coupled Electric Drives [51] are also identified. Information and datasets of these benchmarks are compiled in the web page of the Workshop on Nonlinear System Identification Benchmarks. The cascaded tank system is a fluid level control system, which is consist of two tanks with free outlets fed by a water pump [45]. The fluid levels of these two tanks are adjusted by the input signal that controls the water pump. The coupled electric drives is a system that drives a pulley by controlling a flexible belt. Two electric motors provide the driving force. And the spring is used to fix the pulley. The setup of the Coupled Electric Drives is in Fig. 4.2. A more detailed description of the system and datasets of these

Algorithm 4 Sparse Bayesian Deep Learning Algorithm for System Identification.

Input:
- Collect input-output data $u(t)$ and $y(t)$ for $t = 1, 2, \cdots, T$.
- Arrange input regressors according to the chosen lags l_u, l_y.
- Define the network structure (number of layers L, neurons per layer n_l and activations if it applies).
- Set regularization parameter λ (empirically tuned) and NN pruning thresholds κ_ψ, κ_W ($\approx 10^{-3}$).
- Set the number of repeated experiments M, identification cycles C_{max} and the number of epochs in each cycle $E_{\max}$.
- Initialize hyper-parameters $\Psi(0) = \mathbf{I}$ and $\omega(0) = \mathbb{1}$.

Output: Return the set of connection weights W

for $m = 1$ to M **do**
  for $i = 1$ to $C_{\max}$ **do**
    for $j = 1$ to $E_{\max}$ **do**
      (1) Stochastic Gradient Descent with loss function:

$$W(k+1) = \min_{W} \mathbf{E}(\cdot) + +\lambda \sum_{l=1}^{L} \sum_{b=1}^{n^l} ||\omega_{:b}^{l} \cdot C_{:b}^{l} \odot W_{:b}^{l}||_{\ell_2}$$

    end for
    (2) Update hyper-parameters ψ and ω as Table. 2.1
    (3) Update mask C
  end for
  Simulate on the validation dataset and choose the model with the smallest RMSE.
end for

benchmarks are compiled on the web page of the Nonlinear System Identification Benchmarks. The models with the best validation performance are compared with best models obtained using conventional neural network methods for multiple experiments ($M = 20$) and previous works in literature for every benchmark in 4.3.1.6 Table. 4.3.

Besides the simulation experiment on the above five linear and nonlinear benchmarks, we also explore the effectiveness of our method with a small ratio of datasets on five different benchmarks, i.e., SilverBox Benchmark Dataset, Coupled Electric Drive Dataset, Bouc-Wen Hysteresis Model, Electro-Mechanical Positioning System and Cascaded tank system. The one-step head prediction result shows that the proposed approach can achieve similar prediction accuracy with a sparser model given a small dataset. The detailed experiment result is in Section. 4.3.2.

HAIRDRYER

In typical industrial settings with heating, temperature control is highly desired, given the high transport lags and process delay. The hairdryer is a small scale laboratory apparatus that designates the PT326 process trainer [19]. A mass of air is heated with thermal resis-

Table 4.1: Models are trained to identify linear and nonlinear processes with validation information

Process-Model	Layers-Units	Lags	$\texttt{RMSE}_{\texttt{val}}$ $(\mu \pm \sigma)$	$\texttt{RMSE}_{\texttt{val}}$ (min)	Sparsity	Supporting Material
Hairdryer-MLP	1 - 50	5	0.074 ± 0.0005	0.073	88.1%	4.3.1.1
Hairdryer-LSTM	1 - 10	5	0.093 ± 0.0166	0.081	93.5%	4.3.1.1
Heat Exchanger-MLP	1 - 50	150	0.086 ± 0.0002	0.086	99.3%	4.3.1.2
Heat Exchanger-LSTM	1 - 10	150	0.114 ± 0.0299	0.088	96.4%	4.3.1.2
GT Manufacturing-MLP	1 - 50	5	0.660 ± 0.0013	0.657	97.8%	4.3.1.3
GT Manufacturing-LSTM	1 - 10	5	0.671 ± 0.0019	0.669	99.0%	4.3.1.3
Cascaded Tanks-MLP	3 - 10	20	0.428 ± 0.1032	0.257	84.5%	4.3.1.4
Cascaded Tanks-LSTM	1 - 50	20	0.500 ± 0.1012	0.362	60.3%	4.3.1.4
CED-MLP	2 - 50	10	0.187 ± 0.0285 0.134 ± 0.0192	0.149 0.120	78.4%	4.3.1.5
CED-LSTM	1 - 10	10	0.155 ± 0.0257 0.126 ± 0.0201	0.121 0.097	72.8%	4.3.1.5

tors and flows in a tube. The temperature at the outlet is measured by a thermocouple in volts. The objective is to identify the dynamic relationship between the input voltage to the thermal resistors and the thermocouple voltage at the outlet. The dataset specific to this device is given by MATLAB in a tutorial on linear system identification. The sampling time is 0.08 seconds, and the dataset contains 1000 data points. The dataset is detrended, bringing data to a zero mean. The first 300 data points are used for identification, and the remaining 700 are used for validation.

A fully connected MLP model with one hidden layer and 50 nodes is randomly initialized. The activation function is a linear activation without the bias term. The input and output lags chosen for the regressors are 5. Models are inferred through $K_{\max} = 6$ identification cycles. The best validated model was obtained in the 5th cycle of identification with a sparsity of 88.1%. The model sparsity plot is shown in Fig. 4.3a. Furthermore, an RNN network is randomly initialized with one layer and 10 hidden LSTM units and no bias term. l_u and l_y are set to 5. The 6th and final identification cycle led to the sparsest and best validated model with a sparsity of 93.5%. Fig. 4.3b shows the final model sparsity plot.

Plots of the posterior predictive distribution's mean prediction and standard deviations obtained by sampling 10000 times from the posterior distribution of the connections' weights and by using equations (4.5) and (4.9) are shown in Fig. 4.4a and 4.4b. Plots of the identified models' free run simulations can be found in Fig. 4.14.

HEAT EXCHANGER

A heat exchanger is a thermodynamic device that ensures heat transfer between two fluids separated by a wall. In this experiment, the dynamic relationship between the coolant temperature and the product temperature is identified [19]. The first 3000 data points are used for identification, and the remaining 2000 for validation. This dataset is particularly unique among the others. The process exhibits a delay of around 1/4 of a minute [19].

One hidden-layer MLP with 50 nodes is initialised with a linear activation function and no bias term. The lag chosen is $l_u = l_y = 150$ corresponding to the delay of 0.25 seconds that can be observed in the first instance of the given dataset. The experiment ran for 6

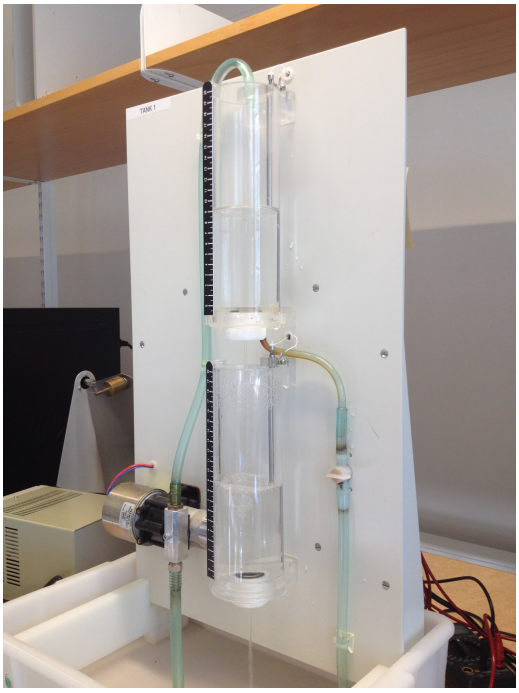


Figure 4.1: The cascaded tanks setup [45]

identification cycles, in which the 4th obtained model was selected as the best validated model. The sparsity of the model is 99.3%.

One layer RNN network with 10 LSTM units is trained with the same lag used previously ($l_u = l_y = 150$). The best validated model was the second out of 6 identification cycles. The accepted model's sparsity is 96.4% for which the sparsity plot is given in Fig. 4.5b.

The predictive mean and standard deviation of the posterior predictive distribution are shown in Fig. 4.6a-4.6b against the real validation signal. These were obtained using 10000 samples of the posterior distribution. Please refer to Fig. 4.15 for a plot of free run simulations.

GLASS TUBE MANUFACTURING PROCESS

In the process of manufacturing glass tubes, the melted glass shapes around a rotating cylinder while homogenizing. It is then drawn on rollers to a certain length. The thickness of the obtained glass tube is measured by a laser beam outside the chamber [50]. The objective is to identify the linear dynamic relationship between the input drawing speed and the output thickness. The datasets are provided by the MATLAB example. These are detrended and decimated by four to get rid of the high-frequency components of the signal [20]. This results in a sampling time of 4 seconds. The data used for identification consist of the first 500 data points, and the remaining is used for validation.

An MLP is randomly initialized with one hidden layer and 50 neurons. The input regressors are chosen such as $l_u = l_y = 5$. The activation function used is linear without a

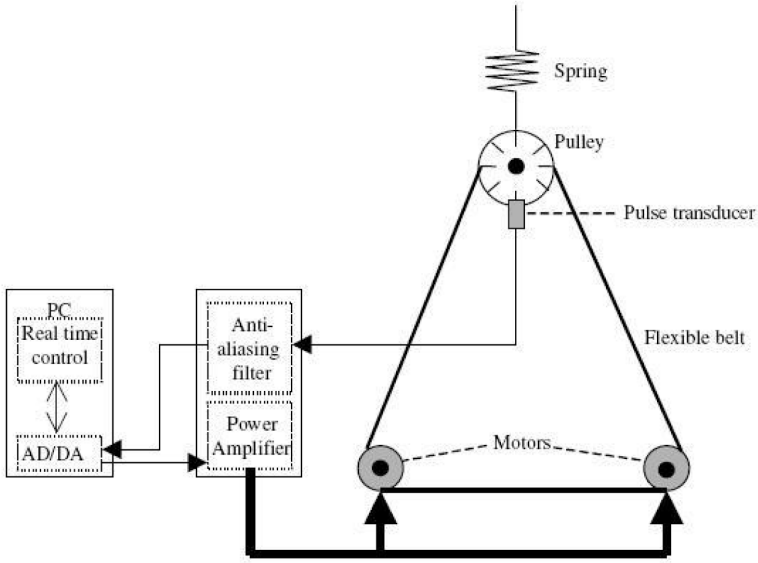


Figure 4.2: The coupled electric drives setup[51]

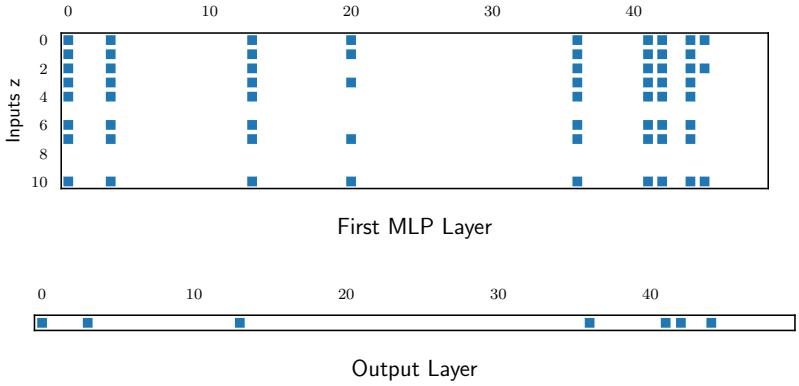
bias term. The final obtained model is 97.8% sparse with a sparsity plot shown in Fig. 4.7a. This model was the third generated model out of 6 identification cycles.

With the same choice of regressors, an RNN network was initialized with one layer of 10 LSTM units. The bias term was not used in this case. In the 6 identification cycles, the 6th generated model was the sparsest and had the best validation performance. The sparsity plot of this network is given by Fig. 4.7b. The model sparsity is 99%, and the only non-pruned parameters in the model correspond to the input to cell state operator W_{ij}.

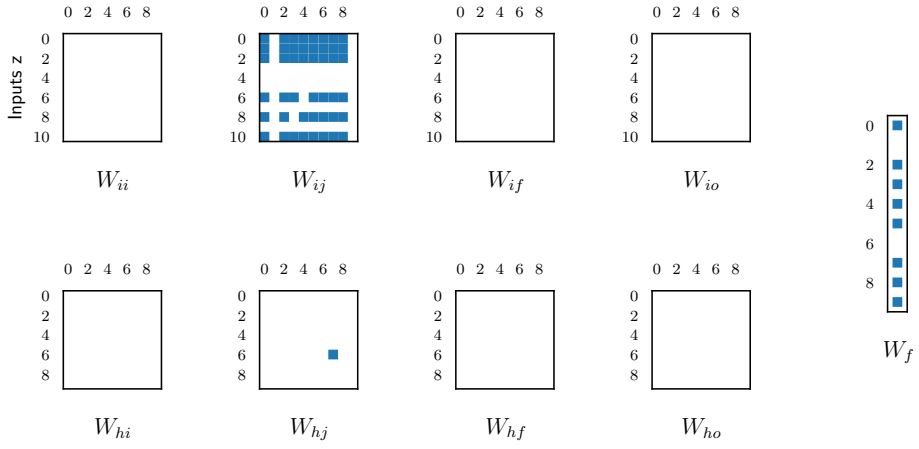
The one-step ahead prediction estimates and uncertainties are obtained by Monte Carlo sampling 10000 times from the posterior and are shown in Fig. 4.8a-4.8b as a representation of the posterior predictive distribution. The free run simulations of the generated models in this chapter are presented in Fig. 4.16.

CASCADED TANKS

A pump drives water up from the reservoir to the upper tank of two vertically cascaded tanks. The upper and lower tanks are separated by a small opening allowing water to fill the lower tank. The lower tank and the reservoir are also separated by a small opening, from which water goes back to the reservoir. In addition to that, water can overflow from the upper tank to the lower tank and reservoir. Water can also overflow the second tank and drop into the reservoir. The small openings and overflows are sources of nonlinearity [45]. The benchmark's objective is to identify the dynamic relationship between the input voltage to the pump and the output measured water level in the lower tank by a capacitive sensor [45]. Two multi-sine input datasets and their corresponding outputs with a sampling rate of 4 seconds are provided. The datasets contain 1024 samples and are with

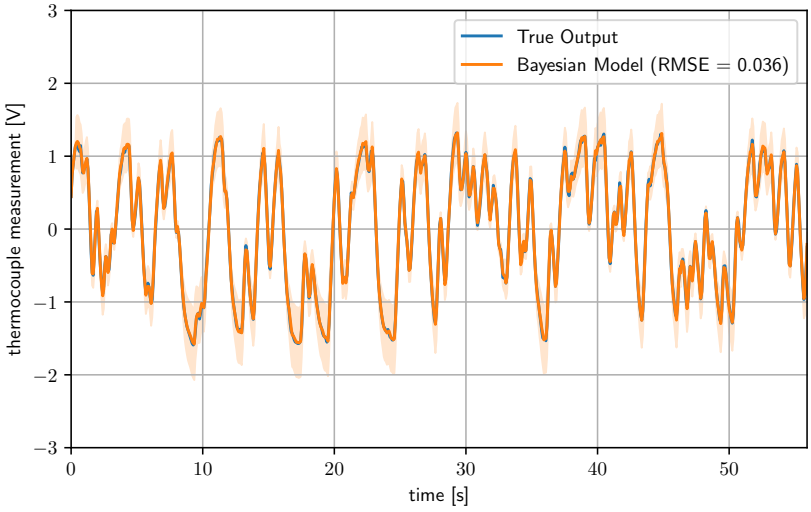


(a) Model sparsity of the identified MLP on Hairdryer dataset. *Blue indicates non-pruned connections and white indicate pruned ones. The same follows with other sparsity plots.*

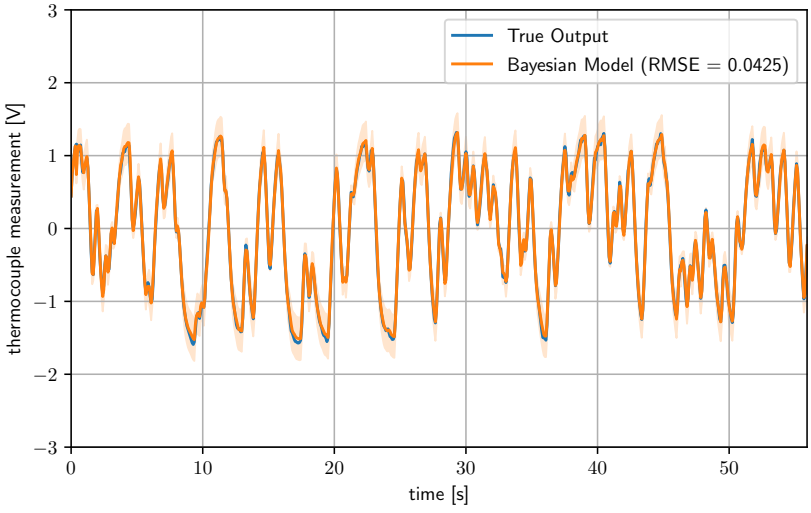


(b) Model sparsity of the identified LSTM on Hairdryer dataset

Figure 4.3: Model sparsity of the identified MLP and LSTM on Hairdryer dataset

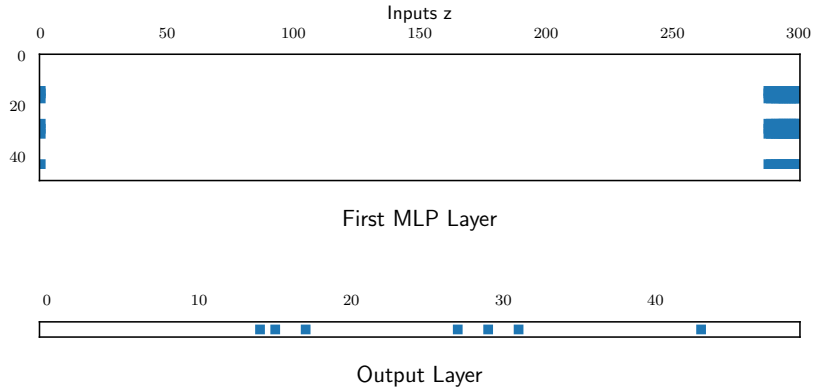


(a) Posterior mean predictions of the identified MLP on Hairdryer dataset ($\pm 2\sigma$)

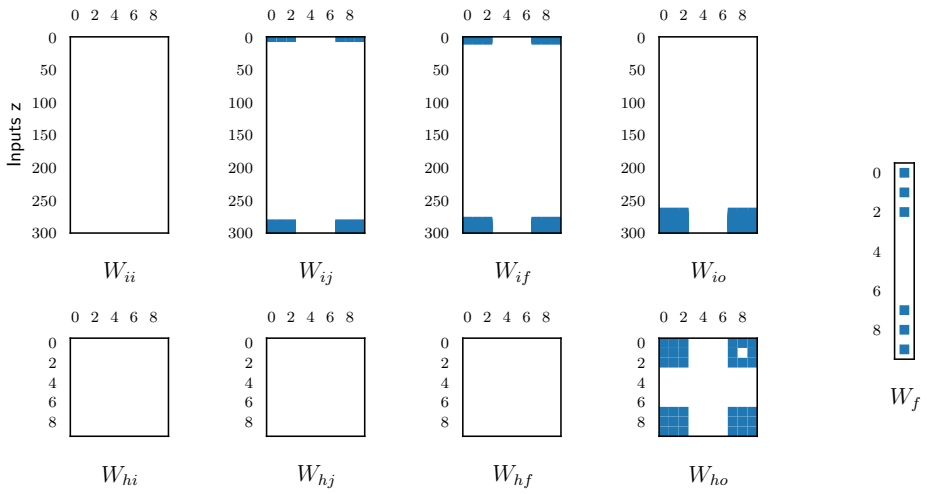


(b) Posterior mean predictions of the identified LSTM on Hairdryer dataset ($\pm 2\sigma$)

Figure 4.4: Posterior mean predictions of the identified MLP and LSTM on Hairdryer dataset

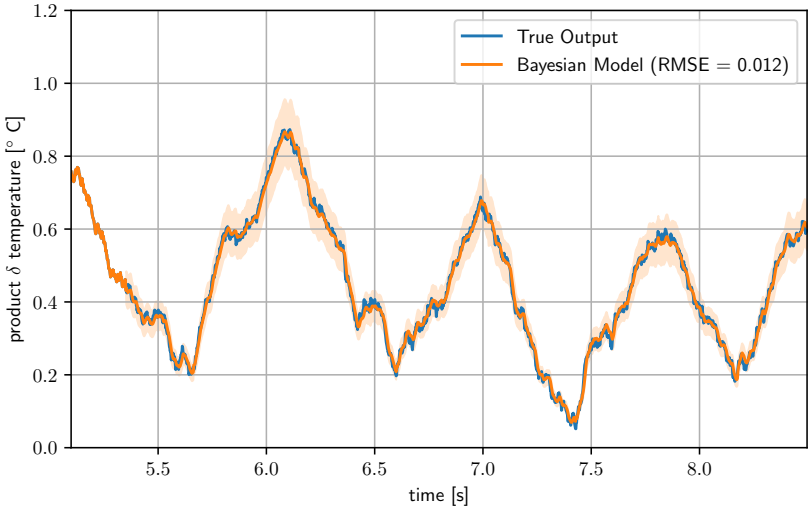


(a) Model sparsity of the identified MLP on Heat Exchanger dataset.

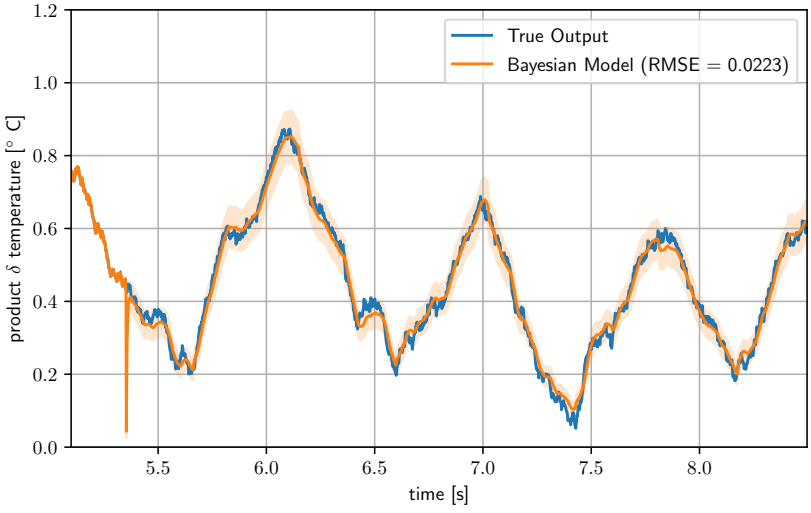


(b) Model sparsity of the identified LSTM on Heat Exchanger dataset.

Figure 4.5: Model sparsity of the identified MLP and LSTM on Heat Exchanger dataset

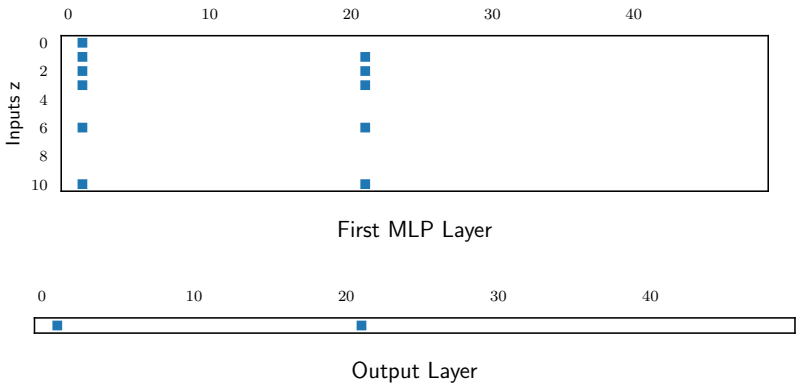


(a) Posterior mean predictions of the identified MLP on Heat Exchanger dataset ($\pm 2\sigma$)

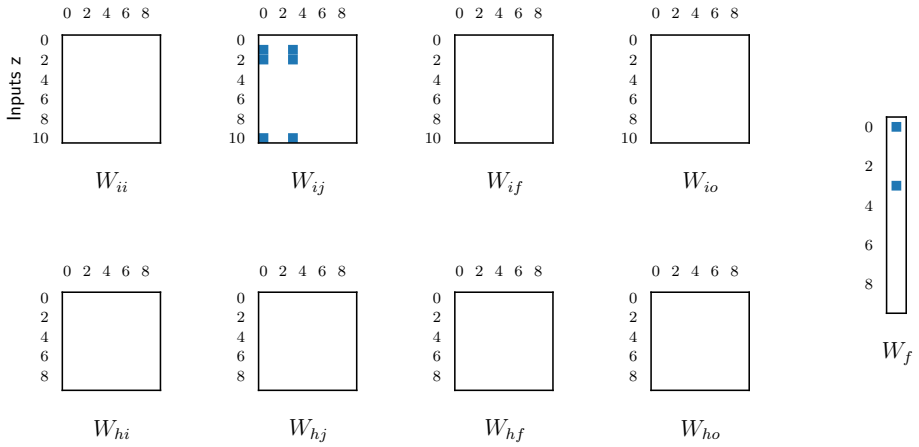


(b) Posterior mean predictions of the identified LSTM on Heat Exchanger dataset ($\pm 2\sigma$)

Figure 4.6: Posterior mean predictions of the identified MLP and LSTM on Heat Exchanger dataset

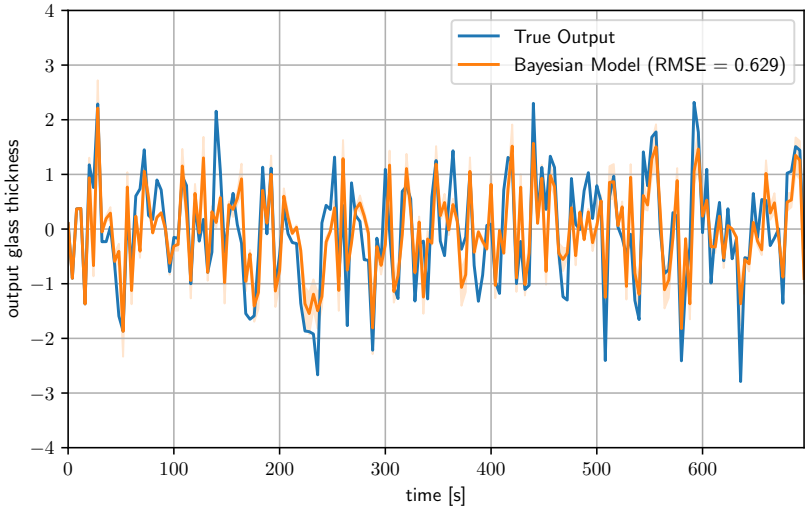


(a) Model sparsity of the identified MLP on Glass Tube Manufacturing dataset.

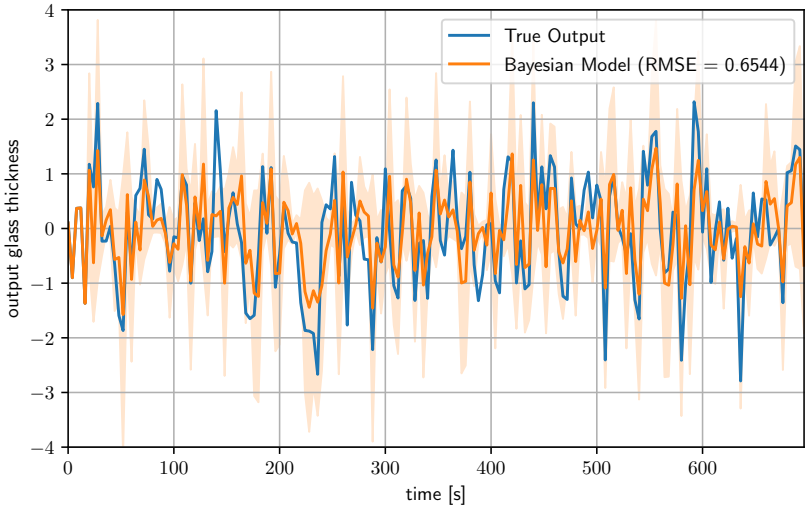


(b) Model sparsity of the identified LSTM on Glass Tube Manufacturing dataset.

Figure 4.7: Model sparsity of the identified MLP and LSTM on Glass Tube Manufacturing dataset



(a) Posterior mean predictions of the identified MLP on Glass Tube Manufacturing dataset ($\pm 2\sigma$)



(b) Posterior mean predictions of the identified LSTM on Glass Tube Manufacturing dataset ($\pm 2\sigma$)

Figure 4.8: Posterior mean predictions of the identified MLP and LSTM on Glass Tube Manufacturing dataset

different initial conditions. One of the datasets is used for estimation and the other for validation. The provided signal shows a static deviation, which is processed by detrending in the preprocessing stage of the identification program.

A 3 hidden layers deep MLP network with 10 neurons per layer is randomly initialized. The activation function used is the relu activation. The input regressors are such as $l_u = l_y = 20$. The identification experiment is run for 10 cycles. The 9th generated model pig erforms the best in validation with a sparsity of 84.5%. The model's sparsity plot is shown in Fig. 4.9a.

Moreover, a one-layer RNN with 10 LSTM units is also used as a model structure for the identification experiment. The 4th identified model with 60.3% sparsity was the best validated model out of 10 identification cycles. The sparsity plot of the corresponding model is shown in Fig. 4.9b.

In addition, the posterior predictive mean and standard deviation are given in Fig. 4.10a and 4.10b. These are obtained by averaging Eq. 4.5 and 4.9 and sampling 50000 times from the inferred posterior distribution of the weights. A plot of the models' free run simulations is by Fig. 4.17.

COUPLED ELECTRIC DRIVES

The coupled electric drives consist of 2 electric motors and a pulley, connected by a flexible belt forming a triangle. The pulley is attached by a spring to a fixed frame. This results in belt tension, slippage, and pulley speed that is harder to model. In addition to that, the output pulley rotational speed is measured in ticks per second, insensitive to rotational directions. The dynamic relationship to be identified is between the input motors voltage and the measured rotational speed of the pulley. For this identification task, 2 uniformly distributed signals of 500 samples were provided spanning 10 seconds. With each of these datasets, the first 300 samples are used for estimation and the remaining for validation.

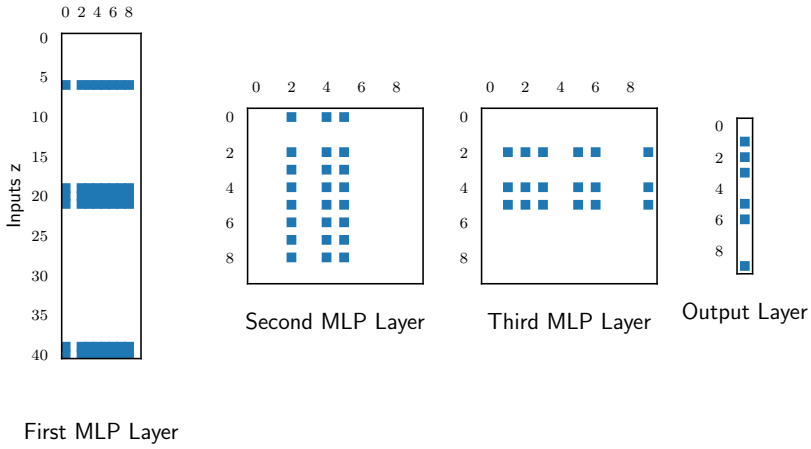
Two hidden layers MLP with 50 neurons each and relu activation functions is randomly initialized and trained with the estimation data for 10 identification cycles. The model's regressors are chosen such that $l_u = l_y = 10$. The model obtained in the 6th identification iteration is the chosen best model. This model is 78.4% for which the sparsity plot is shown in Fig. 4.11a.

The same regressors are used for the identification of the RNN model structure. An RNN with one layer and 10 LSTM units is trained for 10 identification cycles. The 8th identification yields the best simulation validation results. The resulting model sparsity is 72.8% with the sparsity plot in Fig. 4.11b.

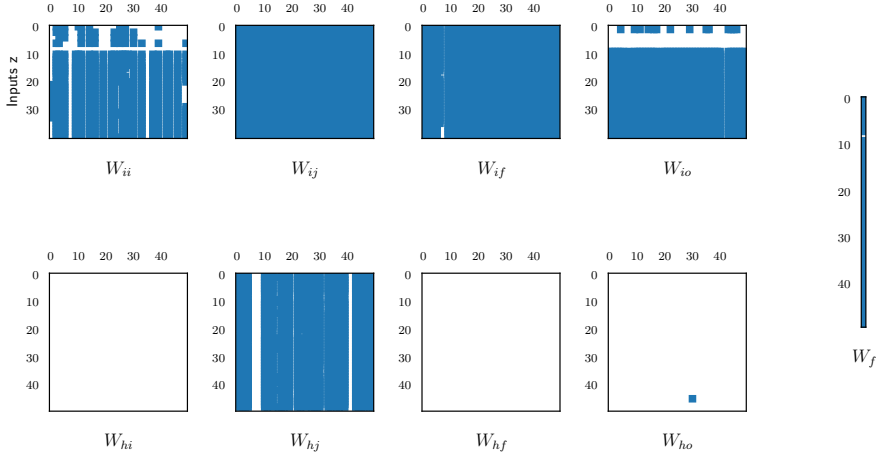
By using Eq. (4.5) and (4.9), the mean and standard deviation of the posterior predictive distributions are plotted in Fig. 4.12a, 4.13a, 4.12b and 4.13b for both validation datasets. These are obtained with Eq. (4.5)- (4.9) and 50000 samples of the posterior distribution. Figures showing the resulting free run simulations are Fig. 4.18-4.19.

FREE RUN SIMULATION RESULTS

This section supports the reader with plots of the simulated experiments using the identified models and a comparison with previous models presented in literature.

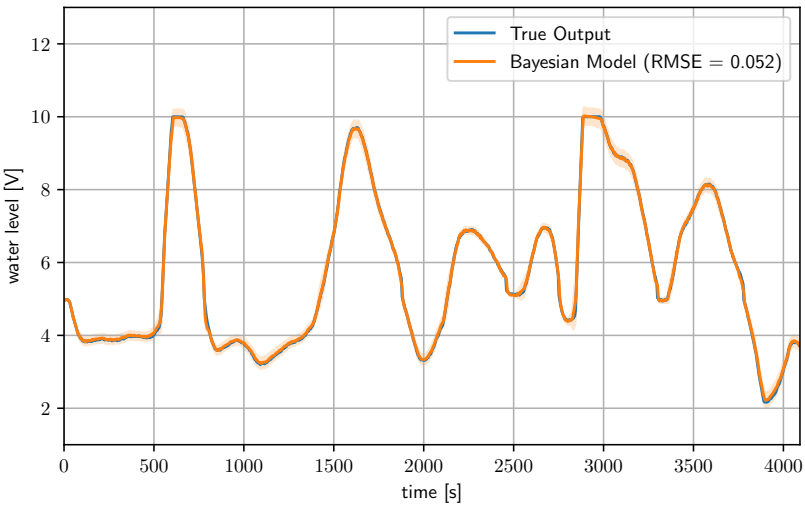


(a) Model sparsity of the identified MLP on Cascaded Tanks dataset.

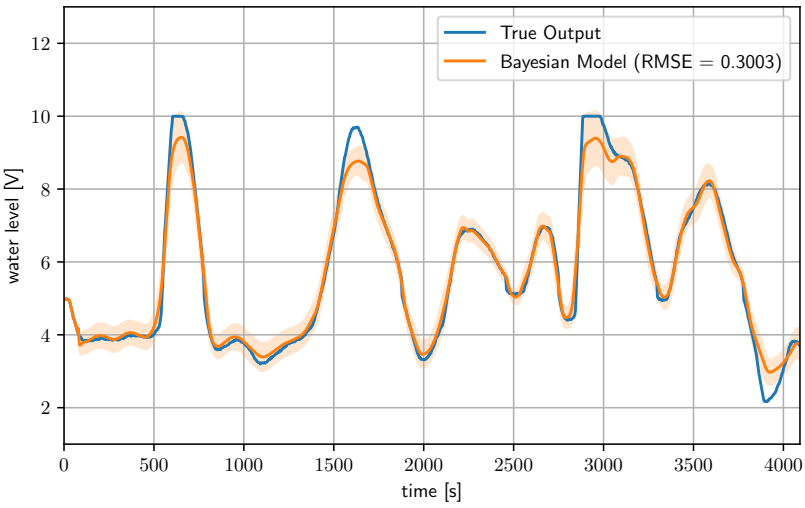


(b) Model sparsity of the identified LSTM on Cascaded Tanks dataset.

Figure 4.9: Model sparsity of the identified MLP and LSTM on Cascaded Tanks dataset

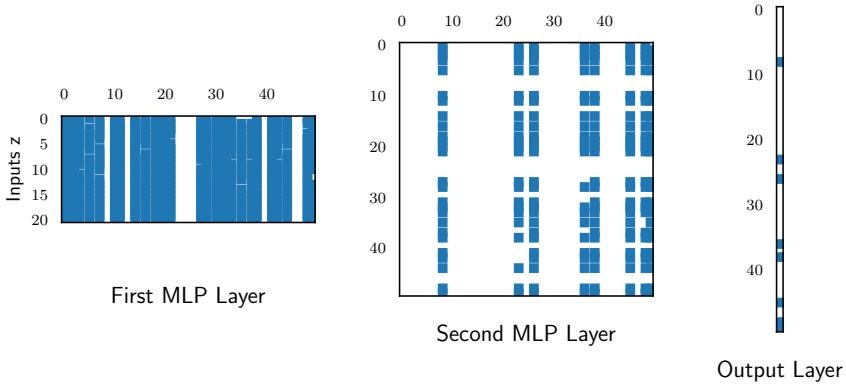


(a) Posterior mean predictions of the identified MLP on Cascaded Tanks dataset ($\pm 2\sigma$)

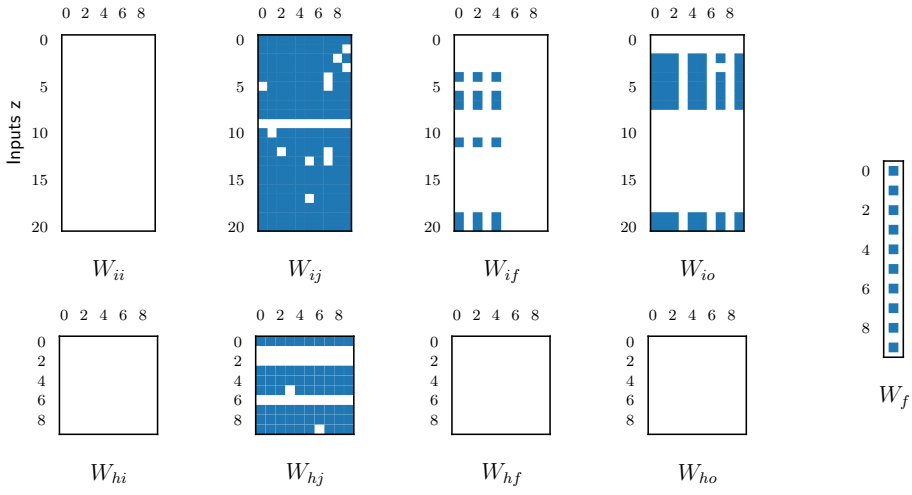


(b) Posterior mean predictions of the identified LSTM on Cascaded Tanks dataset ($\pm 2\sigma$)

Figure 4.10: Posterior mean predictions of the identified MLP and LSTM on Cascaded Tanks dataset

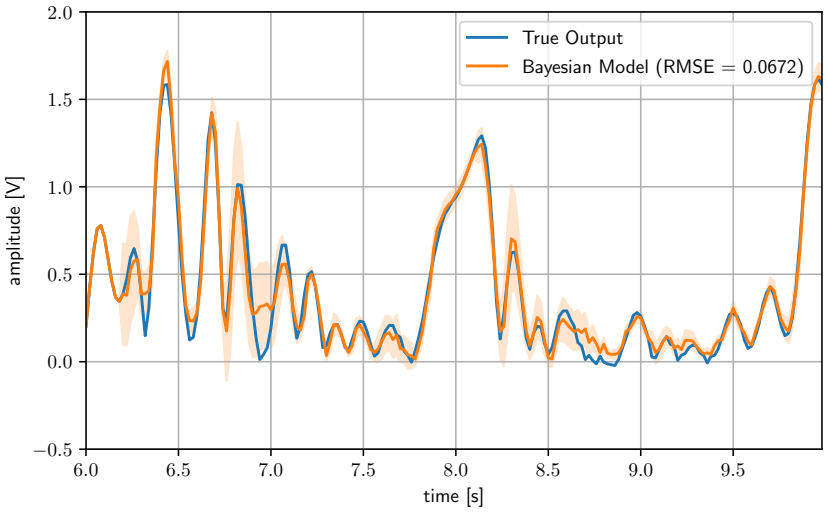


(a) Model sparsity of the identified MLP on Coupled Electric Drives dataset

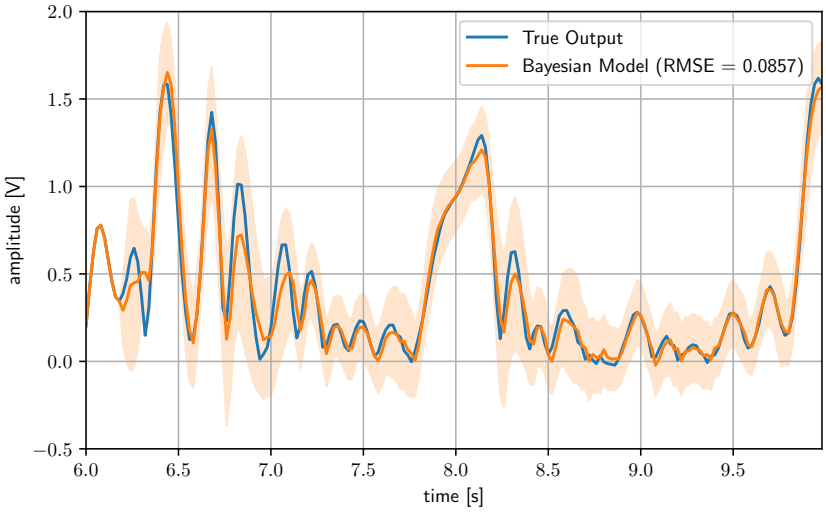


(b) Model sparsity of the identified LSTM on Coupled Electric Drives dataset

Figure 4.11: Model sparsity of the identified MLP and LSTM on Coupled Electric Drives dataset

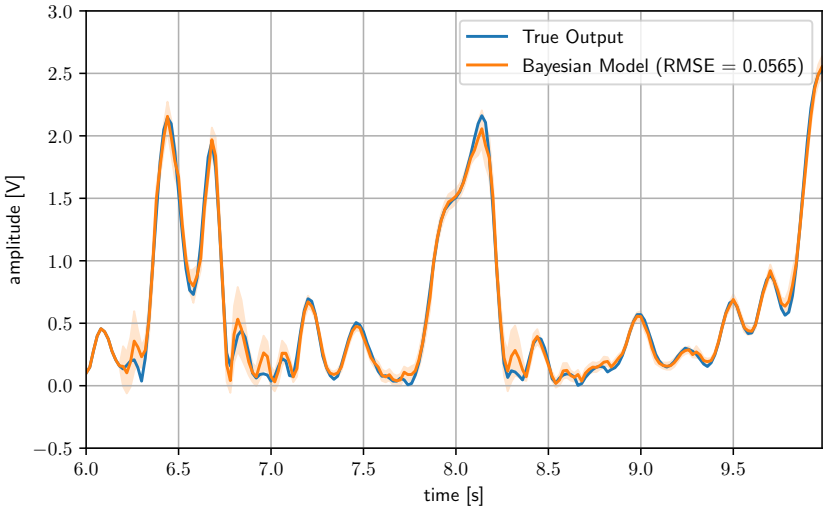


(a) Posterior mean predictions of the identified MLP on the first validation dataset of Coupled Electric Drives dataset ($\pm 2\sigma$)

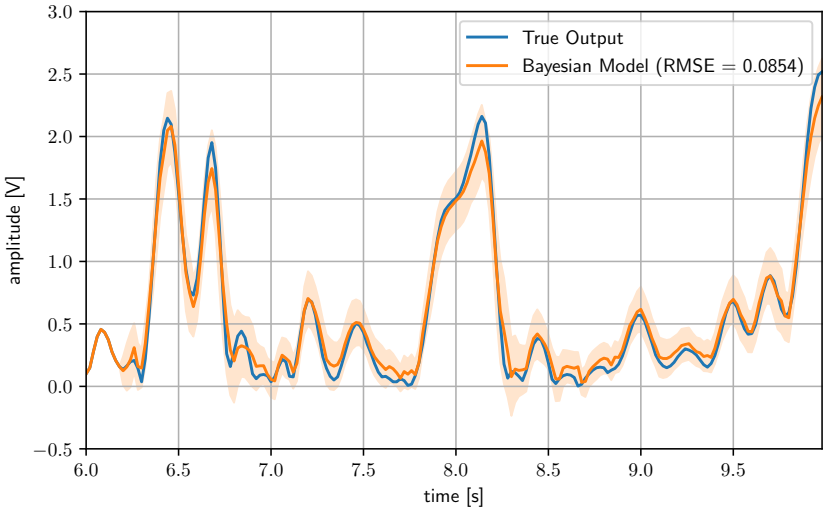


(b) Posterior mean predictions of the identified LSTM on the first validation dataset of Coupled Electric Drives dataset ($\pm 2\sigma$)

Figure 4.12: Posterior mean predictions of the identified MLP and LSTM on the first validation dataset of Coupled Electric Drives dataset ($\pm 2\sigma$)



(a) Posterior mean predictions of the identified MLP on the second validation dataset of Coupled Electric Drives dataset ($\pm 2\sigma$)



(b) Posterior mean predictions of the identified LSTM on the second validation dataset of Coupled Electric Drives dataset ($\pm 2\sigma$)

Figure 4.13: Posterior mean predictions of the identified MLP and LSTM on the second validation dataset of Coupled Electric Drives dataset

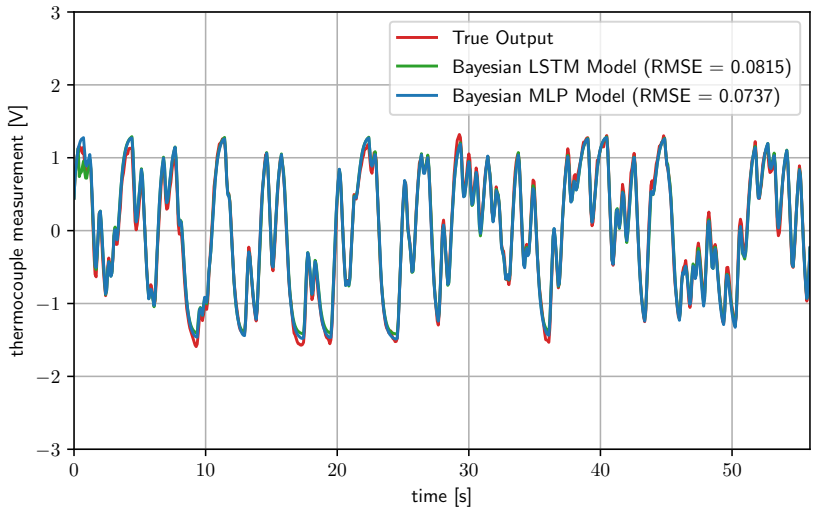


Figure 4.14: Hairdryer Free Run Simulation comparison

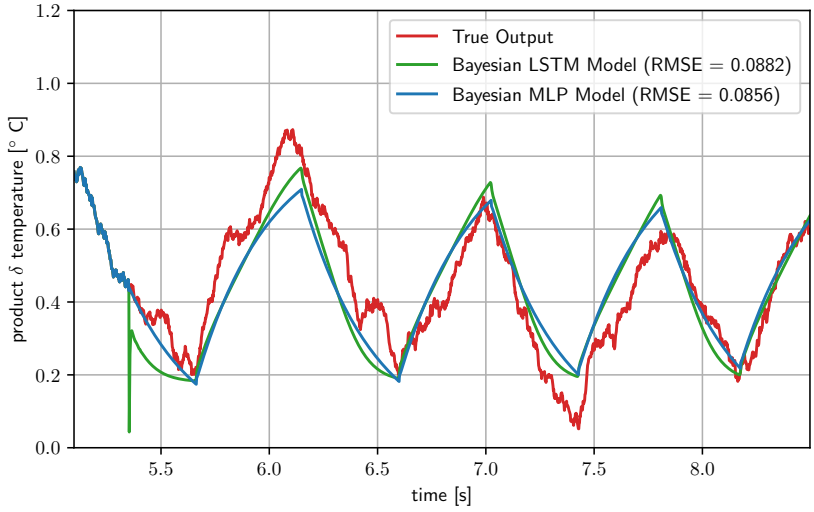


Figure 4.15: Heat Exchanger Free Run Simulation comparison

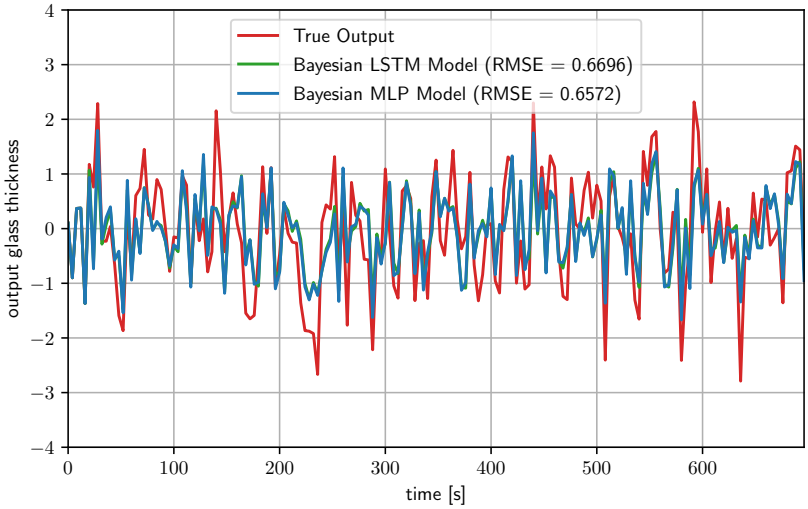


Figure 4.16: Glass Tube Manufacturing Free Run Simulation comparison

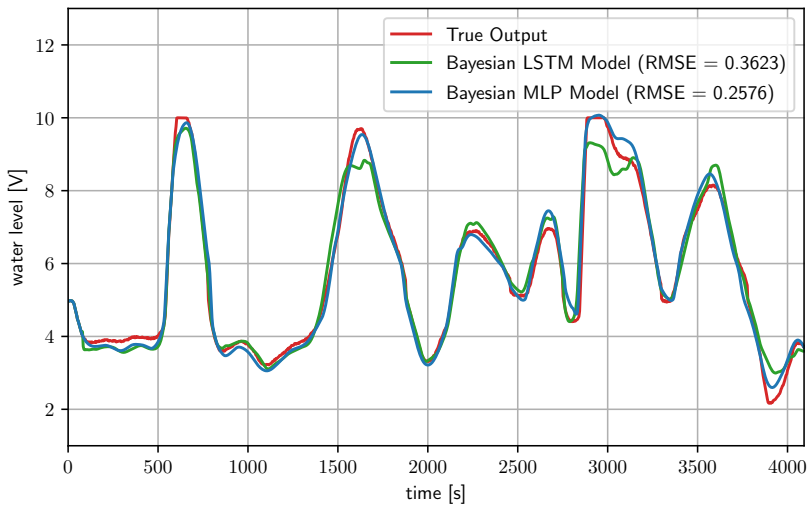


Figure 4.17: Cascaded Tanks Free Run Simulation comparison

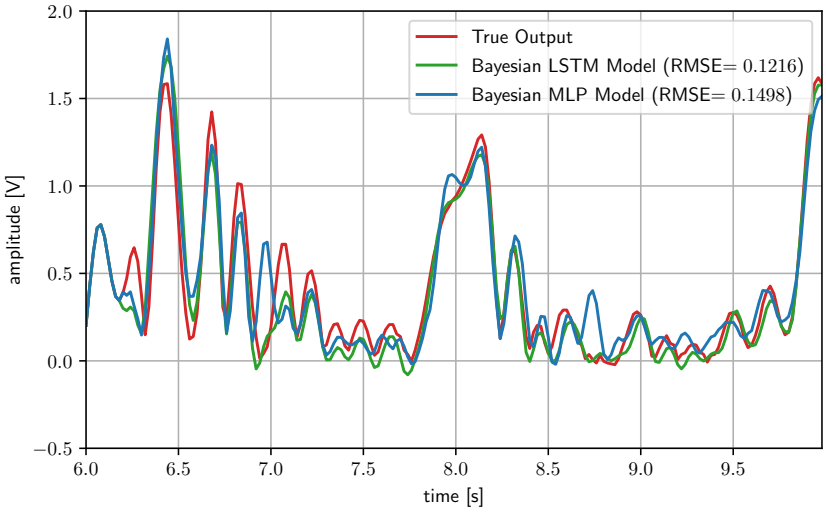


Figure 4.18: Coupled Electric Drives Free Run Simulation comparison for the first validation dataset

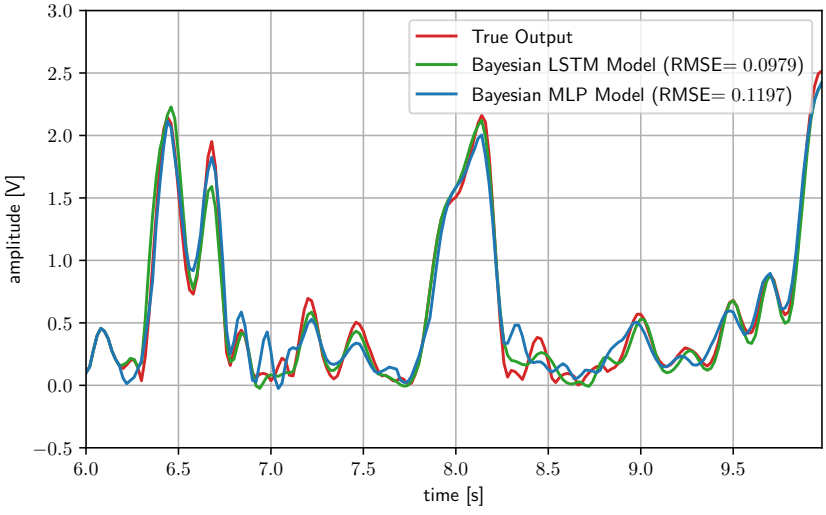


Figure 4.19: Coupled Electric Drives Free Run Simulation comparison for the second validation dataset

Table 4.2: Comparison of RMSE on identified linear systems with other works

Hairdryer	**RMSE [V]**
Transfer Function Estimation [18]	0.108
Subspace Identification [29]	0.105
ARMAX Model [29]	0.104
ARX Model [29]	0.103
GP^a with squared exponential kernel	0.066
LSTM without lags	0.219
LSTM without regularization	0.205
Bayesian LSTM	**0.081**
MLP without regularization	0.076
Bayesian MLP	**0.073**
Heat Exchanger	**RMSE [$^\circ C$]**
Transfer Function Estimation [19]	0.140
Process and Disturbance Model [19]	0.089
Process Model [19]	0.088
GP^a with squared exponential kernel	0.187
LSTM without lags	0.185
LSTM without regularization	0.158
Bayesian LSTM	**0.088**
MLP without regularization	0.092
Bayesian MLP	**0.086**
Glass Tube Manufacturing	**RMSE [·]**
Subspace Identification [20]	0.688
ARX Model [20]	0.676
GP^a with squared exponential kernel	0.656
LSTM without lags	1.099
LSTM without regularization	1.056
Bayesian LSTM	**0.669**
MLP without regularization	0.663
Bayesian MLP	**0.657**

Table 4.3: Comparison of RMSE on identified nonlinear systems with other works

Cascaded Tanks	**RMSE [V]**	
LMN[b] with NFIR [4]	0.669	
Flexible State Space Model [47]	0.450	
Voltera Feedback Model [44]	0.397	
OEM[c] with NOMAD [6]	0.376	
Piecewise ARX Models [33]	0.350	
NLSS[d] [40]	0.343	
Tensor network B-splines [25]	0.302	
GP[a] with squared exponential kernel	0.344	
LSTM without lags	0.954	
LSTM without regularization	0.494	
Bayesian LSTM	**0.362**	
MLP without regularization	0.432	
Bayesian MLP	**0.257**	
Coupled Electric Drives	**RMSE [ticks/s]**	
	Drive 1	Drive 2
Extended Fuzzy Logic [41]	0.150	0.092
Cascaded Splines [42]	0.216	0.110
TAG3P[d] [37]	-	0.128
RBFNN - FSDE[f] [1]	0.130	0.185
GP[a] with squared exponential kernel	0.153	0.132
LSTM without lags	0.394	0.252
LSTM without regularization	0.149	0.131
Bayesian LSTM	**0.121**	**0.097**
MLP without regularization	0.206	0.111
Bayesian MLP	**0.149**	**0.120**

[a] *Gaussian process model.*
[b] *Tree based Local Model Networks with external dynamics represented by NARX or NFIR.*
[c] *Output Error parametric Model estimation based on derivative free method.*
[d] *nonlinear State Space model.*
[e] *Tree Adjoining Grammars*
[f] *Free Search Differential Evolution is used to determine the regressors.*

4.3.2. ONE STEP AHEAD PREDICTION EXPERIMENT ON SMALL DATASET

The SBDL algorithm is also tested on several benchmarks with varying ratios of available data. Four dynamic systems are chosen for the implementation: silverBox benchmark, coupled electric drive, Bouc-Wen hysteresis model, electro-Mechanical positioning system and cascaded tank system. Only the MLP models are adopted in this section with different initialized model structures and hyper-parameter settings. The *one step ahead prediction* performance is used to evaluate the identified models. The SBDL algorithm is also compared with another three training methods,i.e.,DNN training without any regularization; DNN training with conventional shape-wise regularization; Deep Neural Network training with conventional group regularization (row-wise & column-wise).

SILVERBOX

The SilverBox model simulates a second-order mechanical system with a nonlinear stiffness constant. The benchmark input consists of two parts. The first is filtered white Gaussian noise with a cutoff frequency of 200 Hz and varying amplitude (arrow head in Fig. 4.20). The second part is a random odd multi-sine signal with a maximum frequency of 200 Hz, varying amplitude and phases (arrow tail in Fig. 4.20). In this paper, the first part is used for validation and the second part for training.

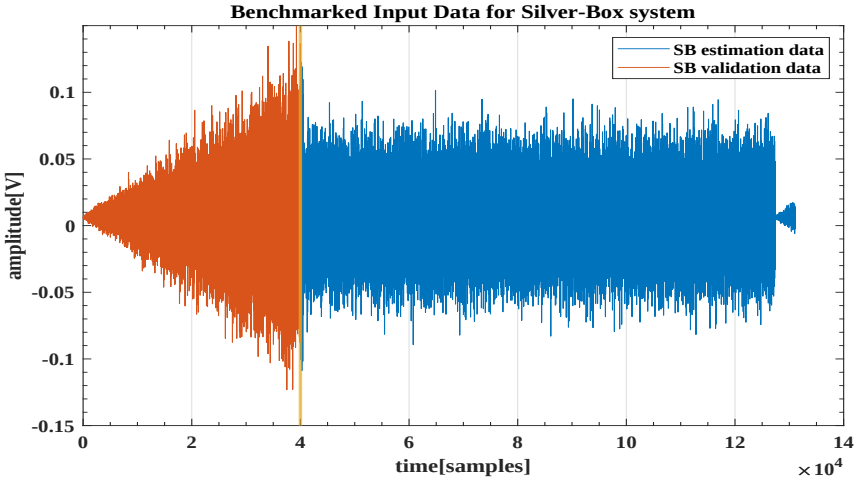
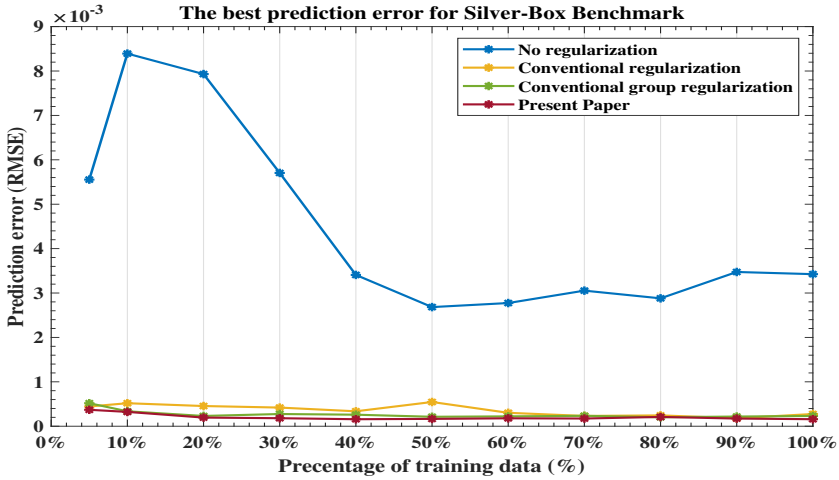


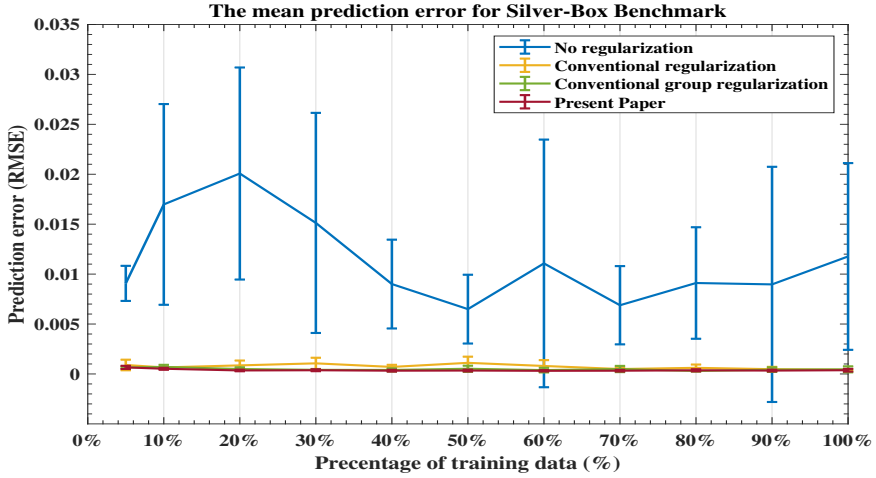
Figure 4.20: Silver-Box provided input data. The head of the arrow represents validation data, and the tail of the arrow the training data.

A MLP is initialized to be with two hidden layers and 500 neurons in each layer. Given that the silverBox model simulates a second-order mechanical system, the input and output lags are chosen to be at least 2. In other words, the input of the MLP includes the system input at time t (u_t) and the system inputs and outputs at times $t-1$ and $t-2$ (i.e., u_{t-1}, u_{t-2}, y_{t-1}, y_{t-2}). The row-wise and column-wise regularization is applied on weight matrices.

The SBDL algorithm provides a more robust result for the *one step ahead prediction* task. As shown in Fig. 4.21b, The SBDL algorithm is more reliable when compared to the

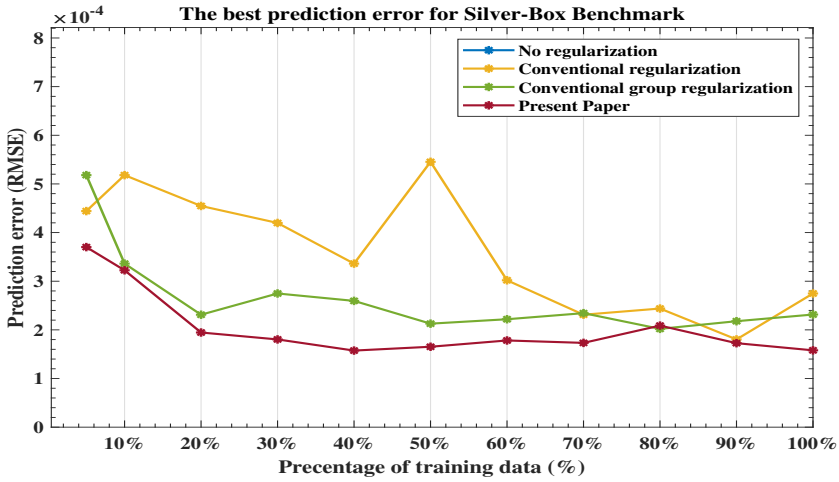


(a) The best prediction errors

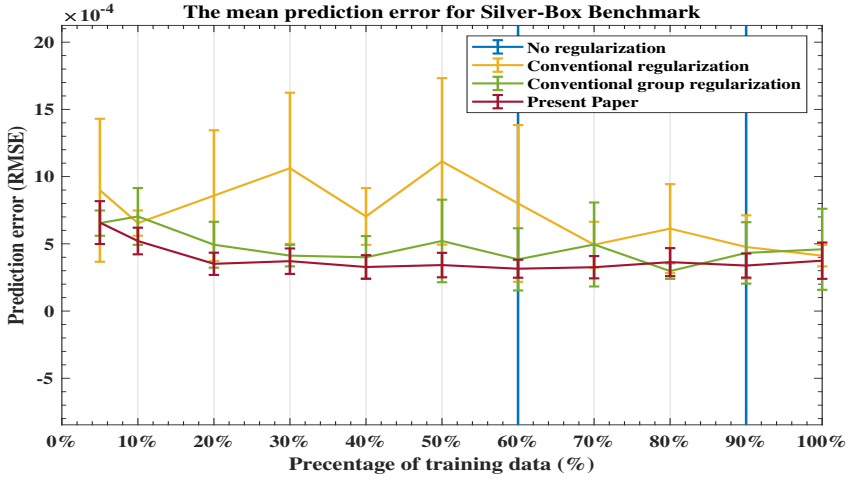


(b) The mean prediction errors

Figure 4.21: The *one step ahead prediction* results with different ratios of original dataset for silver-box Benchmark. **Subfigure (a)** shows the best prediction errors. **Subfigure (b)** shows the means of 20 best prediction errors with vertical error bar representing the standard deviation of validation losses.



(a) The best prediction errors



(b) The mean prediction errors

Figure 4.22: The *One step ahead prediction* results with different ratios of original dataset for silverBox benchmark. The figures above are zoomed versions of figures in Fig. 4.21. This is done to better visualize the results when regularization is used.

no regularization case. It also achieves better results compared to other regularized methods as shown in the zoomed Fig. 4.22.

COUPLED ELECTRIC DRIVES

Coupled Electric Drive (CED) is a mechanical system that consists of a belt connecting two motors and a pulley. The pulley is attached to a suspended spring. Motors can be used to allow control of both the tension and the speed of the belt. The system's dynamics are often expressed as a third-order differential equation relating the input motor loads and the pulley's angular velocity. Two sets of uniform pulses with various random amplitude inputs are provided and are used for training and validation. They are divided into 300 samples for estimation and 200 for validation each. One set is shown in Fig. 4.23.

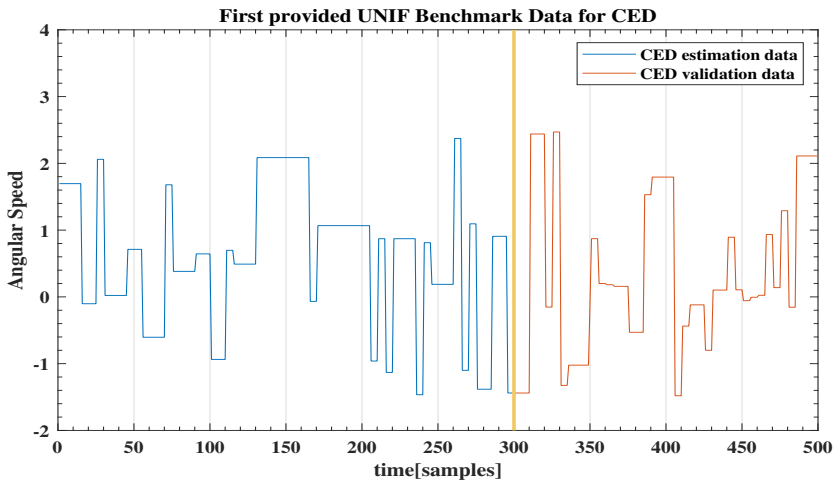
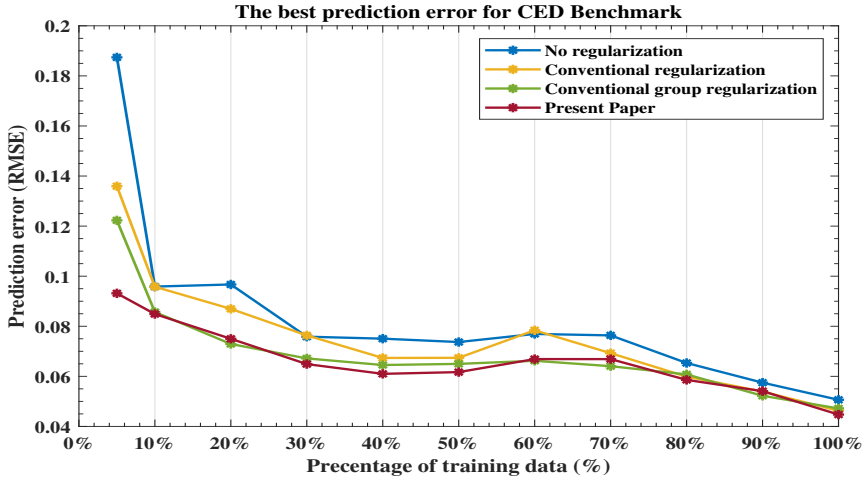


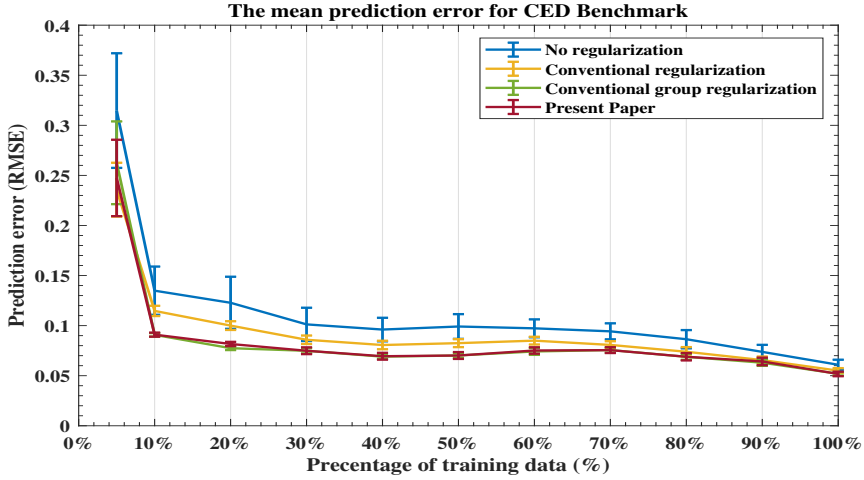
Figure 4.23: A snapshot of the provided input data for Coupled Electric Drive. The sampling frequency is 50 Hz

A MLP is initialized to be with 2 hidden layers and 50 neurons in each layer. The input and output lags are chosen to be 3. The experiments are done 50 times with different λ and various ratios of the available training data from 5% to 100%. As mentioned earlier, the dataset is divided into 300 data points for model estimation and 200 for validation. Both datasets are used to generate one model and the validation is done on both testing sets, with the combined RMSE results shown in the Fig. 4.24a and 4.24b.

The best *one step ahead prediction* performance can be obtained with both conventional group regularization (conventional row-wise and column-wise regularization) and the SBDL algorithm (as shown in Fig. 4.24). In addition, according to Table. 4.5, the best models generated by the SBDL algorithm are sparser than those using conventional group regularization. The SBDL algorithm can provide a relatively sparser model with lower errors given different ratios of data.



(a) The best prediction errors.



(b) The mean prediction errors.

Figure 4.24: The *one step ahead prediction* results with different ratios of available estimation data for CED Benchmark. **Subfigure (a)** shows the best prediction errors. **Subfigure (b)** shows the mean of 50 best prediction errors with vertical error bar for each ratio representing the standard deviation of validation losses.

BOUC-WEN HYSTERESIS MODEL

Hysteresis is a nonlinear phenomenon characterized by a dependency of a system to a previous state, even when the actuating force is not present anymore. Hysteresis can be found in social sciences (for instance, unemployment in economics) and physical sciences (magnetization, random vibration). Several parametric and non-parametric models of this phenomenon exist. However, basing modelling on solely physical laws can be an arduous task [21]. The Bouc-Wen model is a semi-physical model that has been given much attention in the literature, specifically with tuning its parameters for specific applications. The model is provided in Matlab Code, and the user is capable of generating the necessary data for model estimation. In this experiment, and referring to several previous works on this benchmark, a random multisine is generated with an excitation frequency range of 50-150 Hz and an RMS amplitude of 50 N. The latter is shown in Fig. 4.25. For validation, two fixed datasets are provided, one is a random multisine and the other is a sine-sweep [43].

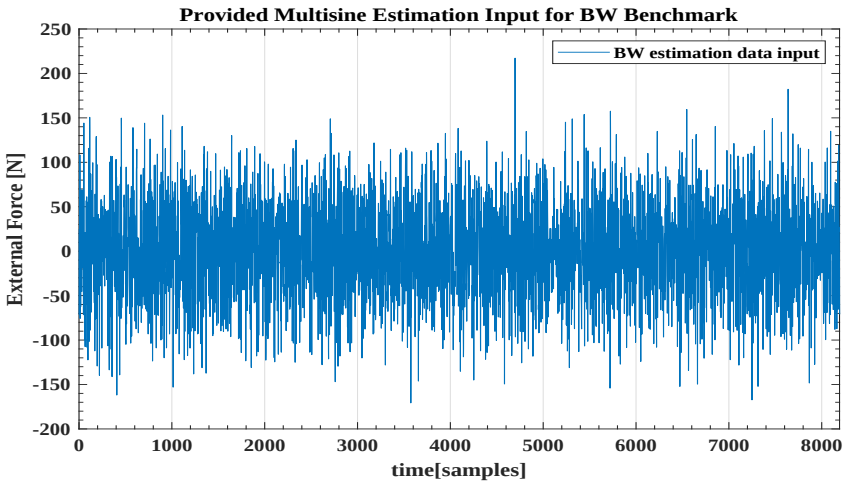


Figure 4.25: Bouc-Wen Model estimation input

A MLP is initialized to be with two hidden layers and 100 neurons in each layer. The input and output lags are set to be 8. The experiments are implemented 50 times with varying regularization parameters λ and ratios of estimation data. The prediction errors in Fig. 4.26a and 4.26b are the joint RMSE over both test datasets.

The errors obtained with SBDL for *one step ahead prediction*, shown in Fig. 4.26, are the lowest among these training methods. In addition, according to Table. 4.6, the models generated by the SBDL algorithm are the sparsest.

ELECTRO-MECHANICAL POSITIONING SYSTEM

The Electro-Mechanical Positioning System (EMPS) is a standard machinery drive system used in various robotics and manufacturing systems. This system consists of a position-controlled DC motor with a translating load. It is mathematically expressed as a second-order function relating the input load to the output actuator position. Both inputs used

Table 4.4: Comparison of model sparsity of Silver-Box trained model.

Ratio	5%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
NN with conventional group regularization	0.51%	0.29%	0.12%	0.10%	0.05%	0.04%	0.04%	0.02%	0.03%	0.03%	0.02%
Present paper with Bayesian framework	0.07%	0.04%	0.41%	0.33%	0.27%	0.21%	0.21%	0.22%	0.15%	0.12%	0.12%

Table 4.5: Comparison of model sparsity of CED trained model.

Ratio	5%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
NN with conventional group regularization	20.24%	7.23%	14.44%	20.53%	24.39%	33.39%	38.84%	39.86%	38.10%	41.15%	32.41%
Present paper with Bayesian framework	13.64%	4.43%	9.16%	21.56%	20.33%	28.42%	31.11%	34.82%	33.61%	33.20%	26.99%

Table 4.6: Comparison of model sparsity of Bouc-Wen trained model

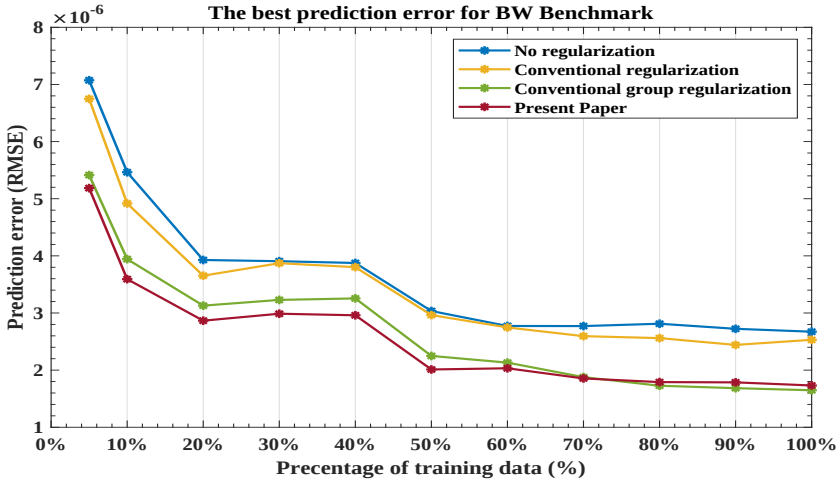
Ratio	5%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
NN with conventional group regularization	50.84%	49.26%	46.26%	46.57%	47.11%	30.11%	29.93%	22.5%	22.35%	17.80%	17.78%
Present paper with Bayesian framework	33.31%	30.12%	27.09%	27.28%	27.08%	14.68%	14.56%	10.02%	9.84%	7.41%	7.55%

Table 4.7: Comparison of model sparsity of EMPS trained model

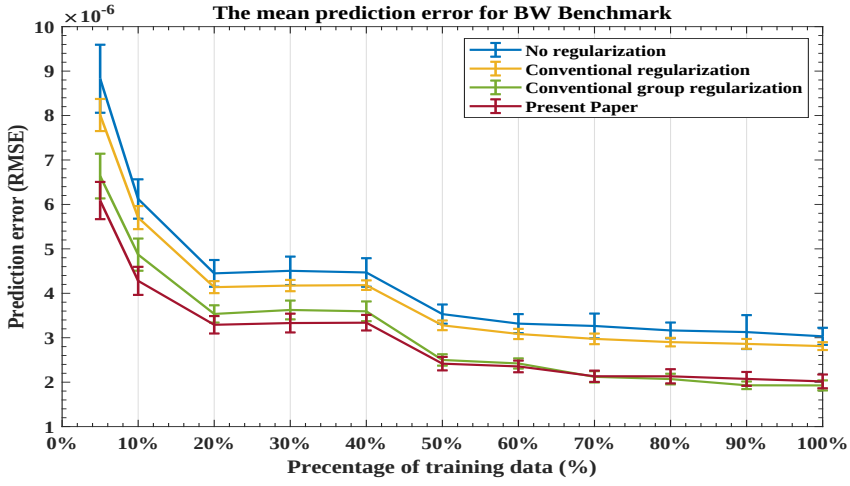
Ratio	5%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
NN with conventional group regularization	36.38%	56.59%	42.74%	40.00%	35.91%	65.26%	61.59%	65.79%	64.68%	42.87%	43.11%
Present paper with Bayesian framework	45.90%	60.00%	58.07%	64.05%	65.20%	57.30%	61.45%	60.83%	71.75%	66.12%	58.21%

Table 4.8: Comparison of model sparsity of cascaded tanks trained model

Ratio	5%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
NN with conventioanl group regularization	46.36%	3.27%	2.23%	2.78%	1.94%	3.04%	3.91%	3.18%	10.98%	51.26%	9.78%
Our method with Bayesian framework	61.52%	2.01%	2.95%	3.42%	2.14%	2.71%	2.79%	2.30%	2.50%	9.79%	2.23%



(a) The best prediction errors.



(b) The mean prediction errors.

Figure 4.26: The *one step ahead prediction* results for BW Benchmark. **Subfigure (a)** shows the best prediction errors. **Subfigure (b)** shows the mean of 50 best prediction errors with vertical error bar for each ratio representing the standard deviation of validation losses.

for training or validation consisting of impulse signals with various amplitudes are shown in Fig. 4.27.

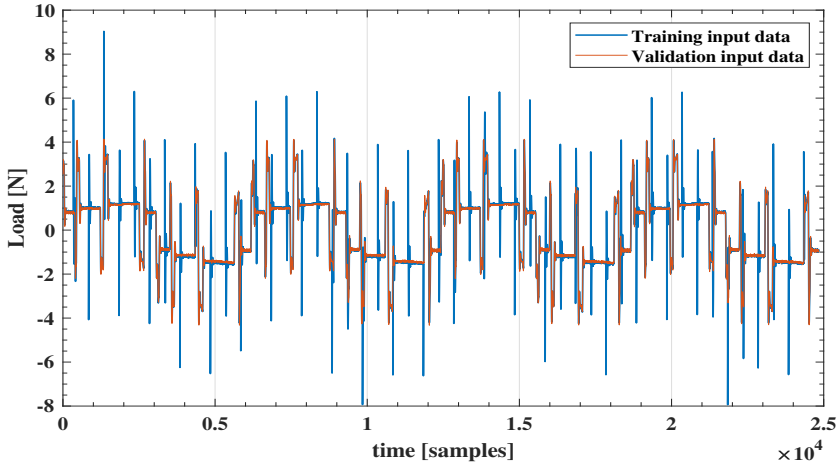


Figure 4.27: The input data of Electro-Mechanical positioning system. The sampling frequency is 1kHz

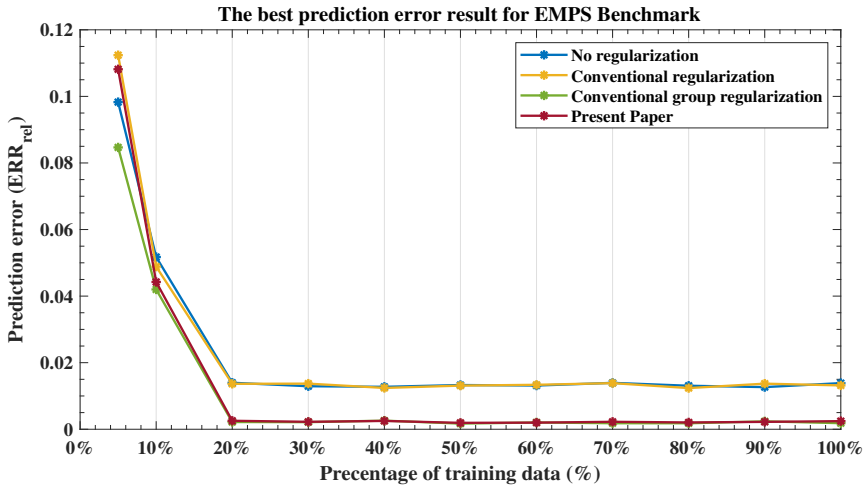
A MLP is initialized to be with two hidden layers and 100 neurons in each layer. The input and output lags are chosen to be 3. The experiments are done 50 times with different regularization parameters λ and various ratios of the training data shown in Fig. 4.27, ranging from 5% to 100%. The errors obtained with SBDL for *one step ahead prediction*, shown in Fig. 4.28, are the lowest among the three methods. In addition, according to Table. 4.7, the models generated by the SBDL algorithm are the sparsest.

CASCADED TANK SYSTEM

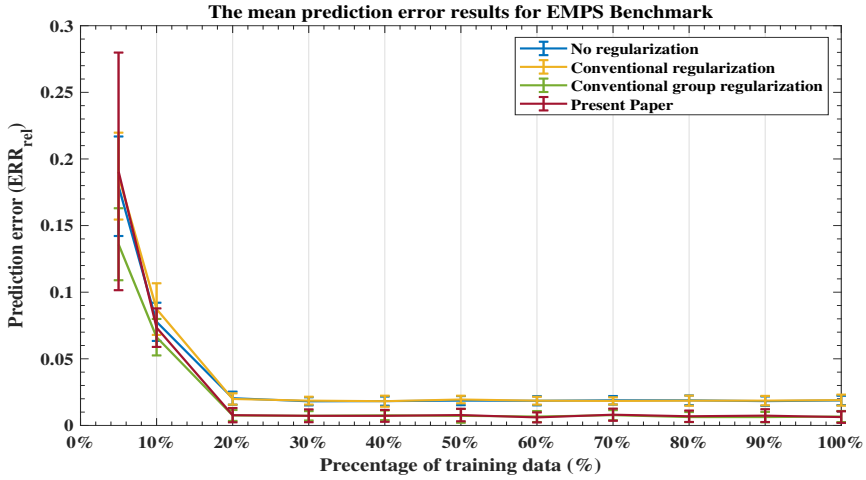
The cascaded tank system is a liquid level control system. Its mathematical model can be constructed as:

$$\begin{aligned} \dot{x_1}(t) &= -k_1\sqrt{x_1(t)} + k_4 u(t) + w_1(t), \\ \dot{x_2}(t) &= k_2\sqrt{x_1(t)} - k_3\sqrt{x_2(t)} + w_2(t), \\ y(t) &= x_2(t) + e(t) \end{aligned} \tag{4.12}$$

where $u(t)$ is the input pump voltage, $y(t)$ is the output which measures the liquid level, $x_1(t)$ and $x_2(t)$ are the states of the system, $w_1(t)$, $w_2(t)$ and $e(t)$ are the noise and k_1, k_2, k_3, and k_4 are the constants which is decided by the system properties. Both the training and test dataset include 1024 samples. A MLP is initialized to be with two hidden layers and 100 neurons in each layer. The data lags for both input and output are set as 5. We apply SBDL algorithm with different λ to the weight matrices. Experiments are implemented with different ratios of the original training dataset. The ratio is selected in the from 5% to 100%. Each experiment with a different setting was repeatedly implemented 50 times in total.



(a) The best prediction error.



(b) The mean prediction error.

Figure 4.28: The *One step ahead prediction* result with different ratios of original dataset for EMPS Benchmark. **Subfigure (a)** shows the best prediction error. **Subfigure (b)** shows the mean prediction error with vertical error bar for each ratio representing the standard deviation of validation losses.

The result is shown in Fig. 4.29, where the smallest and mean prediction errors with a different dataset of four approaches are plotted in different colours. From the result, we observe some interesting phenomena. First of all, all curves show a similar and reasonable trend that the prediction error becomes smaller with more provided training data. Second, it is obvious that our method could obtain a smaller prediction error with a different ratio of the original dataset compared with the experiment without regularisation. Besides, as shown in the blue curve, the prediction error shows a convergence trend. And the optimal prediction error is 0.0472 with only 70% dataset. We argue that there exists a balance between the number of samples and model accuracy. On the one hand, if the provided data is too less (e.g.,less than 50%), we cannot obtain an optimal model no matter whether we introduce the regularization. On the other hand, the result shows that it is possible to obtain the optimal model with a reduced training dataset. On the whole, this result shows that our method can keep good performance even without enough data. Finally, the sparsity of weight matrices also changes over iterations. And compared with the conventional group regularization method, our method could achieve a relatively sparser model, especially when the training data is more than 60%. For example, with 80% training data, the sparsity of the model with our method and the conventional group regularization method is 2.5% and 10.98%. A more direct comparison of the model sparsity for the proposed method and the conventional group regularization method refer to Table. 4.8.

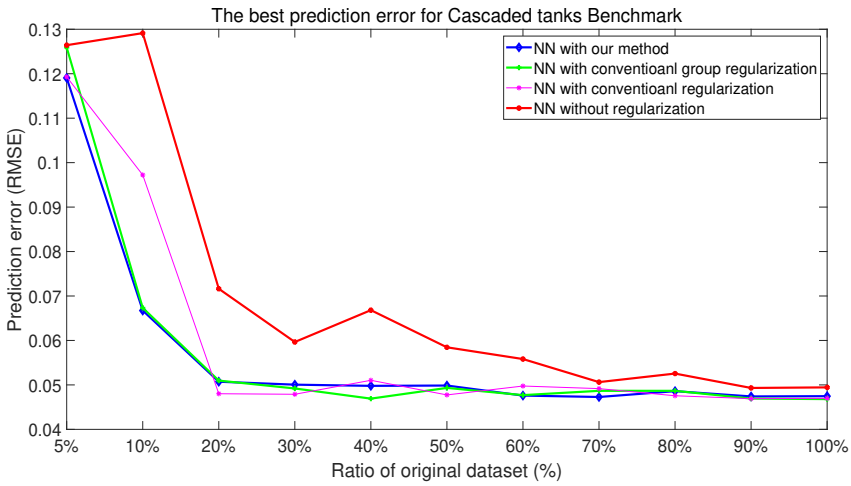
4.4. DISCUSSION

In this section, the results will be discussed and analyzed concerning the claims made on sparsity, uncertainty quantification, and simulation results.

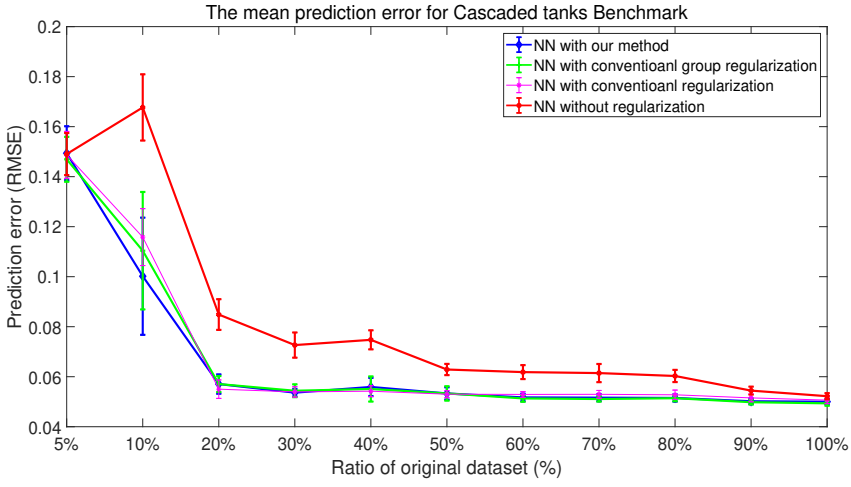
Sparsity: In most cases, the obtained networks are sparse models with a compact structured sparsity. According to Table. 4.1, sparsity was more prominent in the identified linear systems than in nonlinear systems. This demonstrates that the nonlinearity that the data exhibits requires a higher complexity than in the linear case.

Starting with the linear systems, one can note that structured sparsity induced a recognized transport delay in the Heat Exchanger MLP and LSTM models, which characterizes this system. Fig. 4.5b is an example of a sparsity plot of the Heat Exchanger identified LSTM Model. Furthermore, the LSTM models for linear systems have complete operators pruned. This means that the cell state can well be regulated with fewer parameters than imposed on the initialized model structure in the Heat Exchanger case. Similar behaviour is seen across linear benchmarks.

Structured sparsity was also observed in the generated networks for nonlinear systems (Table. 4.1). In addition to that, similarly to RNN models identified for linear systems, a lot of parameters involving the hidden states are pruned. A possible explanation for this behaviour is that the hidden states of LSTM units attempt to retain short-term information from the time series that is also available as lagged elements in the input regressor. The simulation result also shows that the input regressor with lagged elements can improve simulation performance for a LSTM model (check 4.3.1.6 Table. 4.2-4.3). Another observation related to the structured sparsity is the effect of input feature selection. As shown in Fig. 4.8a, the number of input features was reduced from 40 to 2 after applying the sparse Bayesian algorithm. The redundant input features were also identified for other benchmarks and removed from the neural network, reducing model complexity.



(a) The best prediction error.



(b) The mean prediction error.

Figure 4.29: The *one step ahead prediction* results for cascaded tanks Benchmark. **Subfigure (a)** shows the best prediction errors. **Subfigure (b)** shows the mean of 50 best prediction errors with vertical error bar for each ratio representing the standard deviation of validation losses. In both subfigures, the red curve represents the result of our method, the green curve is the result of conventional group regularization method and and blue curve stands for the result of neural network without regularization. 11 data points in each curve correspond to the best prediction error with 11 diverse ratios from 5% to 100%.

Predictive Distributions: The posterior predictive distributions for each model result from the forward propagation of the parameters' posterior uncertainty obtained with the estimation data. Hence, the posterior predictive distribution could spread a more extensive range of predictions if the validation data holds information that the model did not learn from the estimation data [53].

In addition, in some cases, the generated models show an unevenly distributed predictive uncertainty related to nonlinearities or disturbances characteristics of the process and regions where the model can be improved. Fig. 4.10b shows that the identified model for Cascaded Tanks makes less robust predictions when overflow occurs. The Heat Exchanger shows evenly distributed predictions with uncertainty possibly coming from the ambient temperature disturbance. Furthermore, the model type also affects the predictive distribution. Examples include the LSTM models identified for the Glass Tube Manufacturing Process and Cascaded Tanks. In these benchmarks, the identified MLP model provides more robust predictions than the identified LSTM model.

Free Run Simulation Performance: The free run simulation is a good measure of the model's approximation ability to represent a dynamic process by propagating a model's prediction error while forecasting. In this chapter, we select the simulation error as the evaluation metric. It is important to note that, for the studied linear processes, a non-regularized LSTM performs worse when compared to other identification methods. This supports previous concerns made on using LSTM for the identification of linear systems. The Bayesian MLP model outperforms the Bayesian LSTM model in most applications except for the Coupled Electric Drive.

Table. 4.1 shows the mean and standard deviation of the validation simulation errors and the minimum corresponding to the best-chosen model. The minimum is seen to fall close to the range of one standard deviation from the mean. In addition, the variance of validation errors for linear systems is overall less than nonlinear systems, and the variance of validation error for the MLP model is also less than the LSTM model. A possible explanation is that the added complexity in identifying nonlinear processes and the usage of more complex nonlinear structures (LSTM in this case) increases the likelihood of convergence towards saddle points. This is mainly because the Laplace method adopted is a local approximation of the evidence, which is a limitation of the proposed method and justifies running the identification experiment M times.

Nonetheless, in every case (check Table. 4.2-4.3), the Bayesian approach to the identification of each benchmark constitutes an improvement over the conventional MLP and LSTM methods in simulation errors. It pushes these methods to perform competitively against other literature. Besides, we also compare the Gaussian process (GP) [52], which also achieves comparable performance on these benchmarks. However, GP method cannot perform input feature selection efficiently. The detailed result of GP is put in 4.3.1.6 Table. 4.2-4.3.

4.5. CONCLUSION

A Bayesian perspective to system identification has been discussed. The SBDL algorithm with the group prior is evaluated with datasets of three linear and two nonlinear dynamic processes. The Bayesian approach in this chapter used the Laplace approximation to approximate the evidence, a formulation of group sparsity inducing priors to enforce spar-

sity, and Monte-Carlo integration methods to estimate the posterior predictive distribution. The generated models for dynamic systems are sparse models that contribute to input feature selection and perform competitively with other system identification methods in a free-run simulation setting. In addition to that, uncertainties in the inferred predictions and connection weights were quantified. The prediction experiments on different benchmarks with varying ratios of datasets also support that the proposed method is effective and can achieve competitive prediction result given small datasests.

BIBLIOGRAPHY

[1] Helon Vicente Hultmann Ayala et al. “Cascaded free search differential evolution applied to nonlinear system identification based on correlation functions and neural networks”. In: *2014 IEEE Symposium on Computational Intelligence in Control and Automation (CICA)*. Dec. 2014, pp. 1–7. DOI: 10.1109/CICA.2014.7013239.

[2] Roberto Battiti. “Using mutual information for selecting features in supervised neural net learning”. In: *IEEE Transactions on neural networks* 5.4 (1994), pp. 537–550.

[3] Gerben Beintema, Roland Toth, and Maarten Schoukens. “Nonlinear state-space identification using deep encoder networks”. In: *Learning for Dynamics and Control*. PMLR. 2021, pp. 241–250.

[4] Julian Belz et al. “Automatic Modeling with Local Model Networks for Benchmark Processes”. In: *IFAC-PapersOnLine* 50.1 (2017). 20th IFAC World Congress, pp. 470–475. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2017.08.089. URL: http://www.sciencedirect.com/science/article/pii/S2405896317301210.

[5] M. Brunot, A. Janot, and F. Carrillo. “Continuous-Time Nonlinear Systems Identification with Output Error Method Based on Derivative-Free Optimisation”. In: *IFAC-PapersOnLine* 50.1 (2017). 20th IFAC World Congress, pp. 464–469. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2017.08.088. URL: http://www.sciencedirect.com/science/article/pii/S2405896317301209.

[6] Mathieu Brunot, Alexandre Janot, and Francisco Javier Carrillo. “Continuous-time nonlinear systems identification with output error method based on derivative-free optimisation”. In: *IFAC World Congress 2017*. Toulouse, FR, July 2017. URL: http://oatao.univ-toulouse.fr/17993/.

[7] Giovanna Castellano and Anna Maria Fanelli. “Variable selection using neural-network models”. In: *Neurocomputing* 31.1-4 (2000), pp. 1–13.

[8] Tianshi Chen et al. “System identification via sparse multiple kernel-based regularization using sequential convex optimization techniques”. In: *IEEE Transactions on Automatic Control* 59.11 (2014), pp. 2933–2945.

[9] Alessandro Chiuso and Gianluigi Pillonetto. “A Bayesian approach to sparse dynamic network identification”. In: *Automatica* 48.8 (2012), pp. 1553–1565. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2012.05.054. URL: http://www.sciencedirect.com/science/article/pii/S0005109812002270.

[10] Alberto Delgado, Chandra Kambhampati, and Kevin Warwick. “Dynamic recurrent neural network for system identification and control”. In: *IEE Proceedings-Control Theory and Applications* 142.4 (1995), pp. 307–314.

[11] Huyen T Dinh, Shubhendu Bhasin, and Warren E Dixon. “Dynamic neural network-based robust identification and control of a class of nonlinear systems”. In: *49th IEEE Conference on Decision and Control (CDC)*. IEEE. 2010, pp. 5536–5541.

[12] Marco Forgione and Dario Piga. “Continuous-time system identification with neural networks: model structures and fitting criteria”. In: *European Journal of Control* 59 (2021), pp. 69–81.

[13] Yarin Gal. “Uncertainty in Deep Learning”. PhD thesis. University of Cambridge, 2016.

[14] Daniel Gedon et al. “Deep state space models for nonlinear system identification”. In: *IFAC-PapersOnLine* 54.7 (2021), pp. 481–486.

[15] 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*. 1993, pp. 5–13.

[16] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: http://www.sciencedirect.com/science/article/pii/0893608089900208.

[17] Yong Huang et al. “State-of-the-art review on Bayesian inference in structural system identification and damage assessment”. In: *Advances in Structural Engineering* 22.6 (2019), pp. 1329–1351.

[18] The MathWorks Incorporation. *Estimating Simple Models from Real Laboratory Process Data*. https://nl.mathworks.com/help/ident/examples.html. Accessed: 2020-11-25.

[19] The MathWorks Incorporation. *Estimating Transfer Function Models for a Heat Exchanger*. https://nl.mathworks.com/help/ident/examples.html. Accessed: 2020-11-25.

[20] The MathWorks Incorporation. *Glass Tube Manufacturing Process*. https://nl.mathworks.com/help/ident/examples.html. Accessed: 2020-11-25.

[21] Mohammed Ismail, Fayçal Ikhouane, and José Rodellar. “The Hysteresis Bouc-Wen Model, a Survey”. In: *Archives of Computational Methods in Engineering* 16 (June 2009), pp. 161–188. DOI: 10.1007/s11831-009-9031-8.

[22] William R Jacobs et al. “Sparse Bayesian nonlinear system identification using variational inference”. In: *IEEE Transactions on Automatic Control* 63.12 (2018), pp. 4172–4187.

[23] Jeen-Shing Wang and Yen-Ping Chen. “A fully automated recurrent neural network for unknown dynamic system identification and control”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 53.6 (June 2006), pp. 1363–1372.

[24] Pasi Jylänki, Aapo Nummenmaa, and Aki Vehtari. “Expectation propagation for neural networks with sparsity-promoting priors”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1849–1901.

[25] Ridvan Karagoz and Kim Batselier. “Nonlinear system identification with regularized Tensor Network B-splines”. English. In: *Automatica* 122 (2020). ISSN: 0005-1098. DOI: 10.1016/j.automatica.2020.109300.

[26] Moshe Leshno et al. “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”. In: *Neural Networks* 6.6 (1993), pp. 861–867. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(05)80131-5. URL: https://www.sciencedirect.com/science/article/pii/S0893608005801315.

[27] Martin Lindfors and Tianshi Chen. “Regularized LTI system identification in the presence of outliers: A variational EM approach”. In: *Automatica* 121 (2020), p. 109152. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2020.109152. URL: https://www.sciencedirect.com/science/article/pii/S0005109820303502.

[28] Lennart Ljung. “System identification”. In: *Wiley encyclopedia of electrical and electronics engineering* (1999), pp. 1–19.

[29] Lennart Ljung. *System Identification (2nd Ed.): Theory for the User*. USA: Prentice Hall PTR, 1999. ISBN: 0136566952.

[30] Lennart Ljung et al. “Deep learning and system identification”. English. In: *Deep learning and system identification*. Feb. 2020.

[31] David J. C. MacKay. “Bayesian Interpolation”. In: *Neural Computation* 4.3 (1992), pp. 415–447. DOI: 10.1162/neco.1992.4.3.415. URL: https://doi.org/10.1162/neco.1992.4.3.415.

[32] David JC MacKay. “A practical Bayesian framework for backpropagation networks”. In: *Neural computation* 4.3 (1992), pp. 448–472.

[33] Per Mattsson, Dave Zachariah, and Petre Stoica. “Identification of cascade water tanks using a PWARX model”. In: *Mechanical Systems and Signal Processing* 106 (2018), pp. 40–48. ISSN: 0888-3270.

[34] Kumpati S Narendra and Kannan Parthasarathy. “Identification and control of dynamical systems using neural networks”. In: *IEEE Transactions on neural networks* 1.1 (1990), pp. 4–27.

[35] Kumpati S Narendra and Kannan Parthasarathy. “Neural networks and dynamical systems”. In: *International Journal of Approximate Reasoning* 6.2 (1992), pp. 109–131.

[36] Radford M. Neal. “Introduction”. In: *Bayesian Learning for Neural Networks*. New York, NY: Springer New York, 1996, pp. 1–28. ISBN: 978-1-4612-0745-0. DOI: 10.1007/978-1-4612-0745-0_1. URL: https://doi.org/10.1007/978-1-4612-0745-0_1.

[37] Stefan-Cristian Nechita et al. *Toolbox for Discovering Dynamic System Relations via TAG Guided Genetic Programming*. 2020. arXiv: 2012.08834 [eess.SY].

[38] Wei Pan et al. “A sparse Bayesian approach to the identification of nonlinear state-space systems”. In: *IEEE Transactions on Automatic Control* 61.1 (2015), pp. 182–187.

[39] Marios M Polycarpou and Petros A Ioannou. *Identification and control of nonlinear systems using neural network models: Design and stability analysis*. Citeseer, 1991.

[40] Rishi Relan et al. "An unstructured flexible nonlinear model for the cascaded water-tanks benchmark". In: *IFAC-PapersOnLine* 50.1 (2017), pp. 452–457.

[41] F. Sabahi and M. R. Akbarzadeh-T. "Extended Fuzzy Logic: Sets and Systems". In: *IEEE Transactions on Fuzzy Systems* 24.3 (June 2016), pp. 530–543. ISSN: 1941-0034. DOI: 10.1109/TFUZZ.2015.2453994.

[42] Michele. Scarpiniti et al. "Novel Cascade Spline Architectures for the Identification of Nonlinear Systems". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 62.7 (July 2015), pp. 1825–1835. ISSN: 1558-0806. DOI: 10.1109/TCSI.2015.2423791.

[43] M. Schoukens and J.P. Noël. "Three Benchmarks Addressing Open Challenges in Nonlinear System Identification". In: *IFAC-PapersOnLine* 50.1 (2017). 20th IFAC World Congress, pp. 446–451. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2017.08.071. URL: http://www.sciencedirect.com/science/article/pii/S2405896317300915.

[44] Maarten Schoukens and Fritjof Griesing Scheiwe. "Modeling nonlinear systems using a volterra feedback model". In: *Workshop on nonlinear system identification benchmarks*. 2016.

[45] Maarten. Schoukens et al. "Cascaded tanks benchmark combining soft and hard nonlinearities". English. In: *Workshop on Nonlinear System Identification Benchmarks : April 25-27, 2016, Brussels, Belgium*. 2016 Workshop on Nonlinear System Identification Benchmarks ; Conference date: 25-04-2016 Through 27-04-2016. Apr. 2016, pp. 20–23.

[46] Shiliang Sun. "A review of deterministic approximate inference techniques for Bayesian machine learning". In: *Neural Computing and Applications* 23 (2013), pp. 2039–2050.

[47] Andreas Svensson and Thomas B. Schön. "A flexible state–space model for learning nonlinear dynamical systems". In: *Automatica* 80 (2017), pp. 189–199. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2017.02.030. URL: https://www.sciencedirect.com/science/article/pii/S0005109817301048.

[48] Daniel Weber and Clemens Gühmann. "Non-Autoregressive vs Autoregressive Neural Networks for System Identification". In: *arXiv preprint arXiv:2105.02027* (2021).

[49] Wei Wen et al. "Learning structured sparsity in deep neural networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2074–2082.

[50] Vincent Wertz, Georges Bastin, and M. Haest. "Identification of a Glass Tube Drawing Bench". In: *IFAC Proceedings Volumes* 20.5, Part 10 (1987). 10th Triennial IFAC Congress on Automatic Control - 1987 Volume X, Munich, Germany, 27-31 July, pp. 333–338. ISSN: 1474-6670. DOI: https://doi.org/10.1016/S1474-6670(17)55522-6. URL: http://www.sciencedirect.com/science/article/pii/S1474667017555226.

[51] Torbjörn Wigren and Maarten Schoukens. *Coupled electric drives data set and reference models*. English. Technical Report Uppsala Universitet 024. Uppsala University Sweden, Nov. 2017.

[52] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. Vol. 2. 3. MIT press Cambridge, MA, 2006.

[53] Andrew Gordon Wilson. *The Case for Bayesian Deep Learning*. 2020. arXiv: 2001.10995 [cs.LG].

[54] Ye Yuan et al. "Data driven discovery of cyber physical systems". In: *Nature communications* 10.1 (2019), pp. 1–9.

[55] Hongpeng Zhou, Chahine Ibrahim, and Weike Pan. "A Sparse Bayesian Deep Learning Approach for Identification of Cascaded Tanks Benchmark". In: *ArXiv* abs/1911.06847 (2019).

[56] Hongpeng Zhou et al. "Bayesnas: A bayesian approach for neural architecture search". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 7603–7613.

[57] Hongpeng Zhou et al. "Sparse Bayesian Deep Learning for Dynamic System Identification". In: *arXiv preprint arXiv:2107.12910* (2021).

5

THE ART OF PRIOR III: FUSED PRIOR

This chapter implements the proposed sparse Bayesian deep learning algorithm with fused prior on the discovery of governing equations from data. Discovering governing equations from data is critical for diverse scientific disciplines as they can provide insights into the underlying phenomenon of dynamic systems. This work presents a new representation for governing equations by designing the Mathematical Operation Network (MathONet) with a deep neural network-like hierarchical structure. Specifically, the MathONet is stacked by several layers consisting of unary operations (e.g. $\sin, \cos, \log$*) and binary operations (e.g.* $+, -, \times$*), respectively. An initialized MathONet is typically regarded as a super-graph with a redundant structure, a subgraph of which can yield the governing equation. The developed sparse Bayesian learning algorithm with fused prior is applied to extract the sub-graph by employing both non-structurally and structurally constructed priors over the model parameters. By demonstrating on the chaotic Lorenz system, Lotka-Volterra system, and Kolmogorov–Petrovsky–Piskunov system, the proposed method can discover the ordinary differential equations (ODEs) and partial differential equations (PDEs) from the observations given limited mathematical operations.*

5.1. INTRODUCTION

Accurate governing equations (e.g., ordinary differential equations (ODEs) or partial differential equations (PDEs)) can facilitate the study on robust prediction, system control, stability analysis, and increase the interpretability of a physical process [38, 41]. The first principle method is dominating to obtain the governing equation, for example, the one for pendulums using Newton's second law of motion pertains to the relation between the acceleration of an object and the external forces acting on it. Nevertheless, the slow modelling process [27] and the lack of domain knowledge [55] of the first principle method motivate the shift to a data-driven approach.

A seminal breakthrough from [45] applied symbolic regression [10] to determine the underlying structure and parameters of time-invariant nonlinear dynamic systems. Typically, the symbolic regression method [45] is too computationally expensive and is prone to overfitting. [4, 15, 34, 42] used sparse regression techniques to determine the fewest terms in the dynamic governing equations to represent the data accurately. The sparse regression approaches of [4] cannot avoid the nontrivial task of choosing appropriate sets of basis functions and cannot build the model with more unary functions (e.g., such as sin, cos, log). More recently, physical-based machine learning methods arouse much interest, which focuses on combining physical-based modelling and machine learning methods [43, 53]. [37] proposed physics-informed neural networks (PINNs) to discover both the solution and structure of PDEs by encoding prior knowledge in the cost functions. The physics-informed neural networks [35, 37] require prior knowledge and belong to the grey-box approximator, which lacks complete interpretability.

This chapter is motivated by both symbolic regression and sparse regression to discover the governing equations of dynamic systems. Similar to symbolic regression, we start from some basic mathematical operations, i.e., unary (e.g.sin, cos, log) and binary operations (e.g. $+, \times$), instead of a dictionary of predefined equation terms. Instead of exploring the best fitness by trying possible combinations of operations as much as possible, we cast the discovery problem as a sparse optimization problem. Specifically, the optimization is nothing different than training a sparse deep neural network (DNN), i.e., DNN compression problem [12, 25, 52], which can be solved using sparse group Lasso type of algorithms. The key idea is to reformulate the governing equation composed of unary and binary operations into a *augmented hierarchical structure* similar to DNN. In Fig. 5.1, a tutorial is given to show how an expression of $k_4 \sin\big(k_1 y + k_2 y^2\big) + k_5 \cos\big(k_3 x^3\big)$ can be decomposed into a DNN-like model, termed as Mathematical Operation Network (MathONet). The model in Fig. 5.1(c) can be augmented into an over-parametrized network (see Fig. 5.1(d)) by adding extra mathematical operations. In MathONet, the operations are similar to the activation function in DNN; and the weights of the connection between two operations are expected to be zeros if the corresponding operations are redundant. To this end, the true underlying governing equations can be seen as a sub-graph of an over-parameterized MathONet. Essentially, the governing equation discovery problem is equivalent to the compression problem of MathONet with structural and nonstructural sparsities.

Among various DNN compression techniques [12, 13, 24], the structured sparsity learning method in [52] is the most related algorithm for our approach and can be readily applied. This method is essentially a sparse group Lasso type of algorithm [47] where

$\|W\|_{\ell_1}$ and $\|W_g\|_{\ell_2}$ are added on top of the conventional loss function of data as regularizer to promote sparsity of the network weight W. Our practical implementation found that the solutions are always non-sparse compared to the true underlying governing equations, even when the hyper-parameters are extensively tuned (more details can be found in Fig. 5.6 in Section 5.4.1, Fig. 5.9 in Appendix.Section 5.4.2.3 and Fig. 5.12 in Appendix.Section 5.4.3.3). This motivates us to seek a Bayesian learning solution that can potentially result in sparser solutions. The Bayesian learning approach can incorporate the structural and nonstructural sparsity as priors in a principled manner. The algorithm is demonstrated on chaotic Lorenz system, Lotka-Volterra system, and Kolmogorov–Petrovsky–Piskunov system. The proposed method can discover the ordinary differential equations (ODEs) and partial differential equations (PDEs) from the observations given limited mathematical operations, without any prior knowledge on possible expressions of the ODEs and PDEs. Overall, the contribution of this chapter has two folds:

Mathematical operation network design: The governing equations of dynamic systems (ODEs or PDEs) characterized by basic mathematical operations, i.e., unary and binary ones, can be represented as a DNN-like hierarchical structure, termed MathONet. The governing equation discovery problem can be treated as a subgraph search problem of an over-parameterized MathONet graph with redundant mathematical operations.

Bayesian learning discovery algorithm : A Bayesian learning alternative of the sparse group Lasso type of algorithms is applied to discover a more sparse solution, the mathematical operations in governing equations without too much hyperparameter tuning. The algorithm was demonstrated on some well-known dynamics systems in physics and ecology, for which the governing equations are learned from scratch given basic mathematical operations without any prior knowledge on the format of the underlying governing equations.

5.2. MODEL DESIGN

5.2.1. MOTIVATION

We will describe the motivation for MathONet design in more depth following the tutorial in Section 5.1 on the decomposition of $k_4 \sin\big(k_1 y + k_2 y^2\big) + k_5 \cos\big(k_3 x^3\big)$ into a DNN-like hierarchical representation (see Fig. 5.1(c) and (d)). As shown in Fig. 5.1(c), the hierarchical structure is stacked by two typical operations, i.e., unary operations (e.g., sin,cos,log denoted by diamond blocks) and binary operations (e.g. $+, \times$ denoted by square blocks). It should be noted that the input of the system will be "copied" (denoted as the green line in Fig. 5.1(c)) and "pasted" in the following layers (similar to the principle in DenseNet [16]) for the calculations performed by binary operations. The design connecting each layer to the preceding layers can augment the information flow and preserve the feed-forward nature similar to in DenseNet [16].

Based on the motivation, the MathONet is stacked by two typical layers, i.e., binary layer and unary layer. As in Fig. 5.1(b), the binary layer is stacked by Polynomial-Network (PolyNet), and the unary layer is stacked by Operation-Network (OperNet). For a PolyNet, its input is the same as the system input. Its output is a polynomial embodying the linear combination of the inputs. A PolyNet can directly access the gradients of the loss function on the system input, thereby achieving implicit deep supervision. The unary layer is

placed behind the binary layer and performs the linear (e.g., `ident`) or nonlinear transformation (e.g., sin, cos). It should be noted that the structure of MathONet is similar to the structure of a fully connected (FC) neural network, with only two modifications. First, a PolyNet is introduced to replace the weights of a FC neural network. Second, an OperNet is introduced to replace the activation function of a FC neural network. More detailed illustration for PolyNet and OperNet is in Section 5.2.2 and Section 5.2.3, respectively.

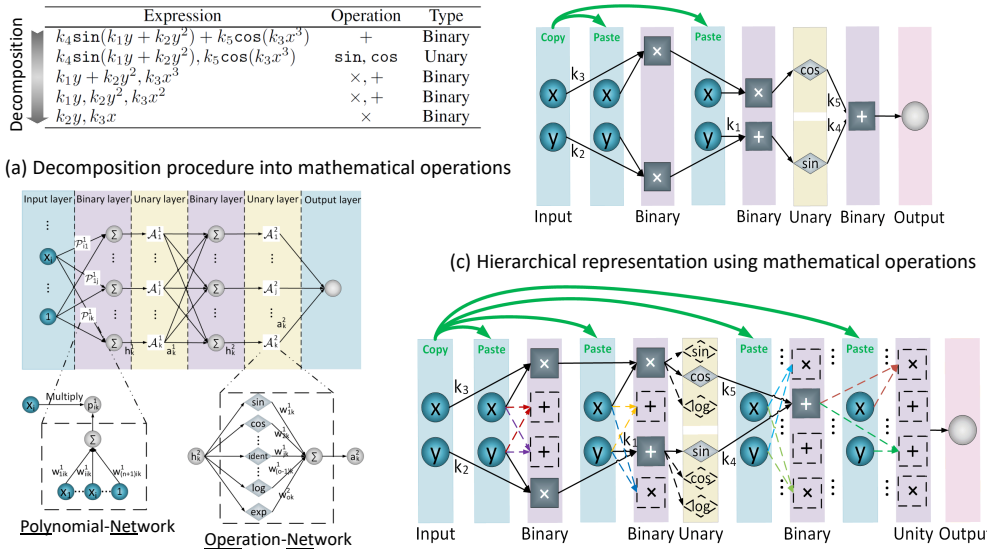


Figure 5.1: (a) Decomposition procedure for $k_4\texttt{sin}(k_1y+k_2y^2)+k_5\texttt{cos}(k_3x^3)$ into mathematical operations in term of unary operations (e.g., sin, cos, log, etc) and binary operations (e.g., $+,-,\times$, etc). (b) A MathONet includes two basic modules, i.e., Polynomial-Network (PolyNet) and Operation-Network (OperNet). The stacked PolyNets form the binary layer, and the stacked OperNets form the unary layer. (c) A DenseNet-like hierarchical representation using mathematical operations, where the square block stands for the binary operation and the diamond block represents the unary operation. The green line denotes that the input of the system will be copied and get involved in mathematical calculations performed by binary operations. (d) An over-parameterized hierarchical structure. The black dotted connection represents the redundant connection that can be regularized on each weight. The dotted connections with different colours denote the redundant connections that can be regularized on a group of weights.

5.2.2. POLYNOMIAL-NETWORK

The Polynomial-Network (PolyNet) is a simple Fully-Connected network without hidden layers, as shown in Fig. 5.1(b). Its input features are consist of two parts. The first part is the original input of the system. The second part is a constant, which enables the MathONet to represent a linear system. We take a PolyNet $\mathscr{P}_{ik}^1$ as an example (see Fig. 5.1(b)).

Typically, the output of $\mathscr{P}_{ik}^{1}$ is a polynomial embodying the linear combination of the input as:

$$p_{ik}^{l} = w_{(n+1)ik}^{l} + \sum_{i=1}^{n} (w_{iik}^{l} x_i) \tag{5.1}$$

where p_{ik}^{l} represents the output of the PolyNet $\mathscr{P}_{ik}^{l}$. l is the layer index. w_{iik}^{l} is the weight parameter whose magnitude determines the strength of the connection within $\mathscr{P}_{ik}^{l}$.

5.2.3. OPERATION-NETWORK

The Operation-Network (OperNet) is designed as a linear combination of unary operations. As in Fig. 5.1(b), the general mathematical expression for the output of an OperNet is

$$a_k^l = \sum_{o=1}^{O} f_o(w_{ok}^l h_k^l) \tag{5.2}$$

where h_k^l is the output of the k_{th} neuron in layer l with $h_k^l = \sum_{i=1}^{n^l}(p_{ik}^l a_i^{l-1} + b_k^l)$. However, the calculation of h_k^1 is different for the first hidden layer with $h_k^1 = \sum_{i=1}^{n^1}(p_{ik}^1 x_i + b_k^1)$. b_k^l is the bias. $f_o()$ stands for unary functions, e.g., sin, cos, log, exp.

By including the PolyNet and OperNet, the MathONet can construct the governing equations of a dynamic system. The designed MathONet has the following advantages: a) a MathONet with a simple structure can represent a considerable expression space. For example, suppose a MathONet includes only 1 hidden layer and 1 hidden neuron with 2 input features and 2 unary functions, the compressed model can represent 2^9 different expressions. b) a MathONet can be trained with typical optimization methods for deep neural networks, e.g., SGD. c) the MathONet can approximate both linear and nonlinear systems. It should be noted that the complexity of a MathONet is mainly limited by three user-defined parameters: a) the number of hidden layers L, which denotes initialized model order and limits the depth of the MathONet; b) n^l, the number of hidden neurons in layer l; c) O, the number of unary functions for OperNet.

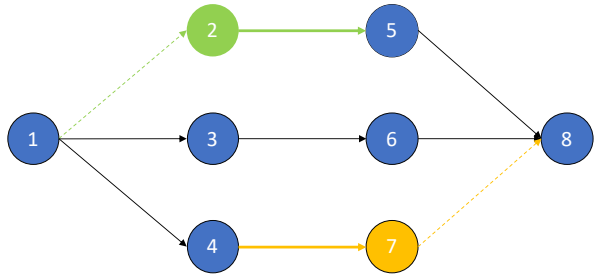
5.2.4. DEPENDENCY BETWEEN CONNECTIONS

As explained before, the overfitting problem can be addressed by pruning the redundant connections, which is a prevalent technique for neural network design and neural architecture search problems [5, 28, 57]. To elaborate the problem of pruning redundant connections better, we suppose there are two states {ON, OFF} to describe a connection, where {ON represents the retained state and {OFF represents the redundant state. The traditional criterium to determine the connection redundancy is only based on the magnitude of the weight of that connection.

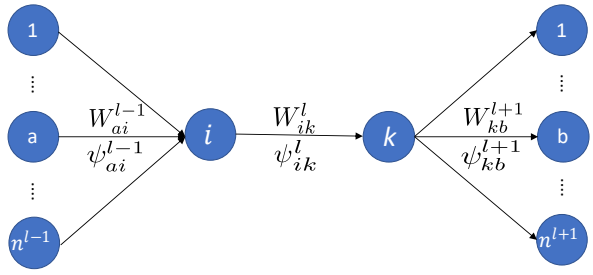
However, this criterium ignores the dependencies between the adjacent connections. In other words, the state of the connection will also be affected by its predecessor and successor. It may result in a disconnected graph that includes some redundant structure if the dependency is not appropriately considered. Figure. 5.2(a) shows a Directed Acyclic Graph (DAG), which can explain several typical types of disconnected graphs. It should be noted that the behaviour of a neural network is similar to the behaviour of this Directed Acyclic Graph (DAG). We use the circle to represent a neuron, whose index is represented by the number within a circle. The edge between neuron i and neuron k is

e_{ik}. The coloured edges (e_{25}, e_{47}) represent two types of disconnected connections. Both edges should be removed since they cannot perform the function of transmitting information within the network. Specifically, e_{25} has no input information caused by its redundant predecessor e_{12}. e_{47} has no output information caused by its redundant successor e_{78}. However, if we mistreat the dependencies between connections, the states of e_{25}, e_{47} is still possible to be evaluated as ON. Therefore, we need to consider how to encode the dependencies into a novel pruning criterium. To address this problem, we first have proposition 1:

Proposition 1 *The state $\{ON, OFF\}$ of a connection e_{ik}^l is determined by whether it can transmit information within the neural network. As shown in Fig. 5.2(b), 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 5.2: An illustration for the dependencies between connections.

Fig. 5.2(b) shows an abstracted multi-in-multi-output motif, which can be regarded as a basic component for any neural network. With the assumption that s_{ik}^l denotes the state of e_{ik}^l, the proposition 1 can be encoded as:

$$\bigcup_a s_{ai}^{l-1} \cap s_{ik}^l \cap \bigcup_b s_{kb}^{l+1} \quad \text{or} \quad \overline{\bigcup_a s_{ai}^{l-1} \cup \overline{s_{ik}^l} \cup \bigcup_b s_{kb}^{l+1}} \tag{5.3}$$

where a, b denote the index of the neuron for the predecessors and successors, respectively, where $a \in (1,\dots,n^{l-1})$, $b \in (1,\dots,n^{l+1})$. n^l stands for the number of neurons in layer l.

In this chapter, we propose two intuitive methods to encode the logic Eq. (5.3). The first method is to calculate the joint probability distribution of e^l_{ik}, as well as its predecessors and successors. We use W^l_{ik} to represent the weight of e^l_{ik}. If the prior distribution of W^l_{ik} is a Gaussian distribution with $p(W^l_{ik}) = \mathcal{N}(W^l_{ik}|0,\psi^l_{ik})$, the variance ψ^l_{ik} representing the uncertainty of W^l_{ik} can be used as a pruning criteria. A smaller ψ^l_{ik} means the W^l_{ik} has a high possibility to be 0 and should be pruned. Assume the distribution of each connection is independent, the joint distribution over $W^l_{ik}, W^{l-1}_{ai}, W^{l+1}_{kb}$ can be expressed as:

$$\begin{aligned} & p(c(W^l_{ik}, W^{l-1}_{ai}, W^{l+1}_{kb})) \\ & \triangleq \mathcal{N}(W^l_{ik}|0,\psi^l_{ik}) \sum_{a=1}^{n^{l-1}} \mathcal{N}(W^{l-1}_{ai}|0,\psi^{l-1}_{ai}) \sum_{b=1}^{n^{l+1}} \mathcal{N}(W^{l+1}_{kb}|0,\psi^{l+1}_{kb}) \\ & = \mathcal{N}\left(W^l_{ik} \sum_{a=1}^{n^{l-1}} \frac{\psi^{l-1}_{ai} W^{l-1}_{ai}}{\sum_{a=1}^{n^{l-1}} \psi^{l-1}_{ai}} \sum_{b=1}^{n^{l+1}} \frac{\psi^{l+1}_{kb} W^{l+1}_{kb}}{\sum_{b=1}^{n^{l+1}} \psi^{l+1}_{kb}} |0, \bar{\psi}^l_{ik}\right) \end{aligned} \tag{5.4}$$

where

$$\bar{\psi}^l_{ik} \triangleq \left(\frac{1}{\sum_{a=1}^{n^{l-1}} \psi^{l-1}_{ai}} + \frac{1}{\sum_{b=1}^{n^{l+1}} \psi^{l+1}_{kb}} + \frac{1}{\psi^l_{ik}}\right)^{-1} \tag{5.5}$$

if κ_ψ is the threshold for uncertainty, the connection with $\bar{\psi}^l_{ik}$ smaller than κ_ψ will be determined as redundant.

The second method is to encode the dependency from the perspective of the magnitude of weight. It is well known that a larger magnitude typically means the corresponding edge is more critical. If we define κ_w as the weight threshold, the proposition 1 can be translated as follows. The state of e^l_{ik} is OFF if at least one of the following three conditions is satisfied: a) the magnitude of W^l_{ik} is smaller than κ_w; b) the weight magnitudes of all predecessors W^{l-1}_{ai} are smaller than κ_w; c) the weight magnitudes of all successors W^{l+1}_{kb} are smaller than κ_w. The redundancy of e^l_{ik} can be evaluated by the sign function $sgn(W^l_{ik})$:

$$sgn(W^l_{ik}) = sgn(|W^l_{ik}|) \sum_{a=1}^{n^{l-1}} sgn(|W^{l-1}_{ai}|) \sum_{b=1}^{n^{l+1}} sgn(|W^{l+1}_{kb}|) \tag{5.6}$$

where $sgn(|x|) = \begin{cases} 0, & |x| \le \kappa_w \\ 1, & |x| > \kappa_w \end{cases}$.

With the above two methods to encode dependency, the state s^l_{ik} can be decided by:

$$s^l_{ik} = \begin{cases} \text{OFF}, & \bar{\psi^l_{ik}} < \kappa_\psi \ or \ sgn(W^l_{ik}) = 0 \\ \text{ON}, & others \end{cases} \tag{5.7}$$

5

5.3. IDENTIFICATION METHOD

5.3.1. SPARSE GROUP LASSO

Sparse group Lasso [47] is a regularization optimization technique that is a combination of Lasso [49] and group Lasso [56]. Suppose a weight matrix can be divided into different groups and W_g represents a group of weights, its optimization target can be formulated as:

$$\min_{W} E(\cdot) + \sum_{W} \lambda \|W\|_{\ell_1} + \sum_{W} \sum_{g} \lambda_g \|W_g\|_{\ell_2} \tag{5.8}$$

where $E(\cdot)$ is the loss of data. $\|W\|_{\ell_1}$ represents the Lasso regularization on each weight. And $\|W_g\|_{\ell_2}$ is the group Lasso regularization on a group of weights. In this chapter, the weights within each PolyNet and OperNet can be collected as one group. λ and λ_g are the tuning parameters.

As in [20, 52], sparse group Lasso can identify the redundancy of each connection and achieve structured sparsity by zeroing out all weights of a group. It should be noted in our algorithm, the sparse group Lasso is exactly the first cycle to start. As shown in the Lorenz experiment, the identified model by sparse group Lasso is still redundant/non-sparse with 651 terms (see Fig. 5.6b) and cannot reproduce the attractor dynamics precisely (see Fig. 5.3(b)). Further research is needed to study why the sparse group Lasso is inefficient. These inefficiencies motivate us to develop a Bayesian learning version of the sparse group Lasso, which can potentially yield sparser solutions.

5

5.3.2. ALGORITHM

The Bayesian approach with fused priors can be referred to Section. 2.1.4 of Chapter. 2. We generate the binary matrix C as the *mask* of W, which denotes the connection redundancy. C has the same dimension as W and will be optimized during the training process. The update of C is decided by:

$$C = \begin{cases} 0, & s = OFF \\ 1, & s = ON \end{cases} \tag{5.9}$$

where 1 denotes the redundancy, and 0 means the weight should be retained. As the dependency between connections is considered in this chapter, the calculation of s is in Eq. (5.7). It should be noted that the *mask* C will be updated at the last epoch of each cycle. The generic optimization target for the MathONet can be formulated as:

$$\min_{W,C} E(\cdot) + \sum_{W,c} \lambda \|\omega \odot C \odot W\|_{\ell_1} + \sum_{W_g, C_g} \lambda_g \|\omega_g \cdot C \odot W_g\|_{\ell_2}^2. \tag{5.10}$$

The dimension of C and ω are the same as W. ω_g a scalar which is shared by all connections within the group W_g. where $E(\cdot)$ is the energy function, $E(\cdot) = \frac{1}{\mathscr{D}} \sum_{k=1}^{K} (\mathbf{y}_k - \hat{\mathbf{y}}_k)^2$. $\odot$ denotes the Hadamard product. Inspired by [9, 30], we also evaluate the predictive uncertainty using a practical Monte-Carlo sampling method. By sampling over the inferred posterior distribution for T repetitions, an unbiased estimate of prediction can be approximated by the average of predicted output, i.e., $\bar{\mathbf{y}}_k = \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{y}}_k^t$. where $\hat{\mathbf{y}}_k^t$ is the predicted output of the t-th samples and $\Sigma_{\hat{\mathbf{y}}_k} = \frac{1}{T} \sum_{t=1}^{T} (\hat{\mathbf{y}}_k^t - \bar{\mathbf{y}}_k)^2$, A pseudo-code for the discovery algorithm is given by Algorithm 5.

It should be noted that Algorithm 5 is a modification of Algorithm 3, which includes the dynamic pruning strategy as described in Algorithm. 4.

Algorithm 5 Bayesian learning discovery algorithm

Initialize: hyper-parameters $\omega, \psi = I$; regularization tuning parameter $\lambda, \lambda_g \in \mathbb{R}^+$; threshold for pruning $\kappa_\alpha, \kappa_{\alpha_g} \in \mathbb{R}^+$. $C_{\max} \in Z^+$ denotes the maximum cycles; $E_{\max} \in Z^+$ denotes the number of epochs in each cycle.
for $i = 1$ to $C_{\max}$
  for $j = 1$ to $E_{\max}$
    1. Update the weight W by minimizing the loss function as Eq. (5.10).
  end for

2. Update hyper-parameters $\boldsymbol{\psi}$, ω as Table. 2.1.

3. Update mask C and C_g as Eq. (5.9).

end for

5.4. EXPERIMENT

In this section, the algorithm is demonstrated on the chaotic Lorenz system [31], Lotka-Volterra system [2] and Kolmogorov Petrovsky Piskunov (Fisher-KPP) system [8]. All experiments are performed in PyTorch framework by using a single GPU (NVIDIA TITAN V).

5.4.1. CHAOTIC LORENZ SYSTEM

SYSTEM DESCRIPTION

As a typical canonical model for chaotic dynamics, the Lorenz system is nonlinear, non-periodic and is notable for its chaotic solution being sensitive to system parameters and initial conditions. Although the dynamics of the Lorenz attractor is difficult to interpret, the attractor action can be described by a simplified mathematical model, which is a three-dimensional and deterministic ODE:

$$\dot{x} = \sigma(y - x), \qquad \dot{y} = x(\rho - z) - y, \qquad \dot{z} = xy - \omega z \tag{5.11}$$

with the standard parameter values $\sigma = 10, \omega = 8/3$ and $\rho = 28$, the system exhibits chaotic behavior as shown in Fig. 5.3(c). The data is collected through simulation experiments. The state vector and their derivatives are stacked as input and output datasets, respectively.

EXPERIMENT SETUP

The MathONet is initialized with 1 hidden layer and 3 hidden neurons. The OperNet includes 5 basic unary functions, i.e. `identity`, sin, cos, log, `exp`. The initial values of the regularization parameters λ, λ_g are assigned from the set $\{1e^{-2}, 1e^{-4}, 1e^{-6}, 1e^{-8}, 1e^{-10}\}$ and are decayed to one-tenth every 200 epochs. For each hyper-parameter, the identification procedure is repeated 10 times with differing weight initialization.

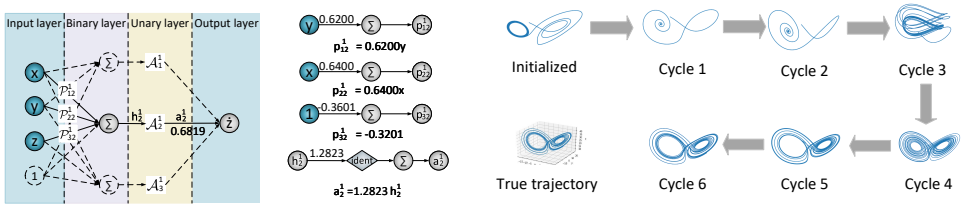
RESULT ON NOISE-FREE DATASET

With the simulated dataset without noise, the identified governing equations without fine-tuning are:

$$\dot{x} = -10.000x + 10.000y \tag{5.12a}$$
$$\dot{y} = -1.000xz + 28.000x - 1.000y \tag{5.12b}$$
$$\dot{z} = 1.000xy - 2.667z \tag{5.12c}$$

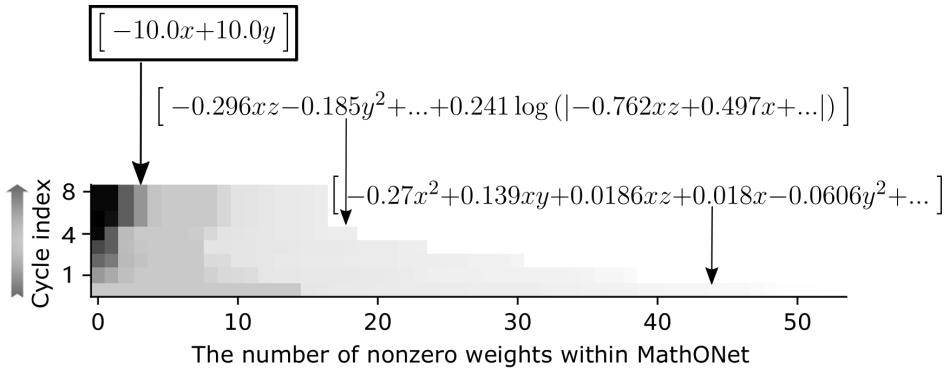
It can be observed that both the equation structures and parameters are captured accurately. The identified result for model z is shown in Fig. 5.3. Fig. 5.3(b) is the attractor trajectory of the model generated in each cycle during the training process. The first cycle is the conventional regularization that cannot identify the dynamics precisely. By applying the sparse Bayesian deep learning algorithm 5, the identified model of cycle 6 can reproduce the attractor dynamics accurately.

Fig. 5.4, Fig. 5.5 and Fig. 5.6 show the searching process for the governing equations of model x, y and z, respectively. In Fig. 5.4a, Fig. 5.5a and Fig. 5.6a, only the non-zero weight elements of the MathONet generated in each cycle are collected to form a weight vector. The identified governing equation has undergone a transformation from complex to simple. If combined with Fig. 5.4b, Fig. 5.5b and Fig. 5.6b which show the change of predictive ability and model sparsity, it can be observed that the predictive ability tends to improve as the model complexity decreases. It only takes 7 cycles to identify the correct structures and coefficients for model x, 9 cycles for model y and 6 cycles for model z . This is reasonable because the model y is more complex with a second-order term than model x and z as described in (5.12). This also verifies that the proposed method can discover governing equations in a reasonable, effective and accurate manner.

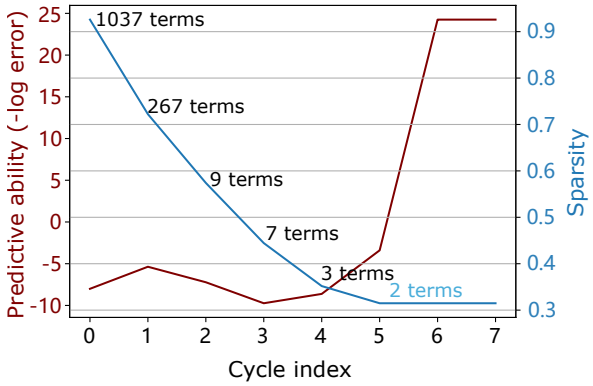


(a) Identified MathONet (b) Identified PolyNet and OperNet (c) Simulated attractor trajectory of each cycle

Figure 5.3: Identified result of Lorenz system. (a) Identified MathONet model for model z. The MathONet is initialized with 1 hidden layer and 3 hidden neurons. The OperNet includes 5 basic unary functions in the beginning, i.e., `identity`, sin, cos, log, `exp`. Algorithm 5 can identify the essential connections after 6 cycles. The dotted parts are the identified redundant connections and neurons. (b) The identified PolyNets and OperNets represent simple mathematical expressions that are placed under each basic module. (b) The attractor trajectories generated by the intermediate model of each cycle. The identified model can reproduce the attractor dynamics as the true model after 6 cycles.

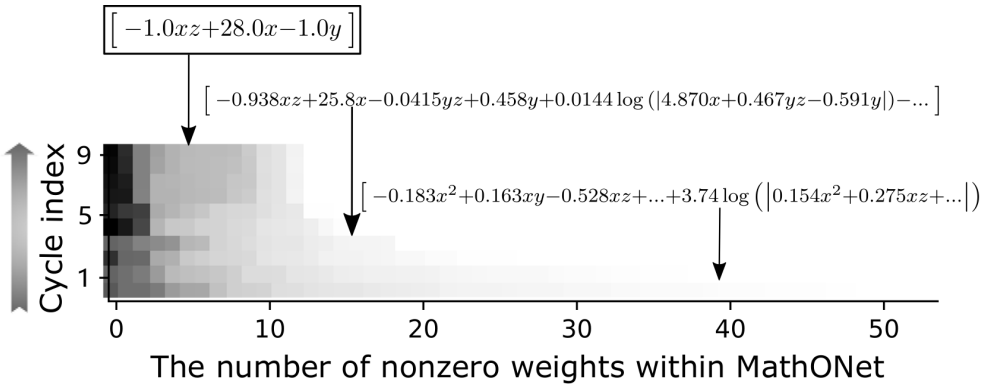


(a) The number of nonzero weights of the MathONet in each cycle.

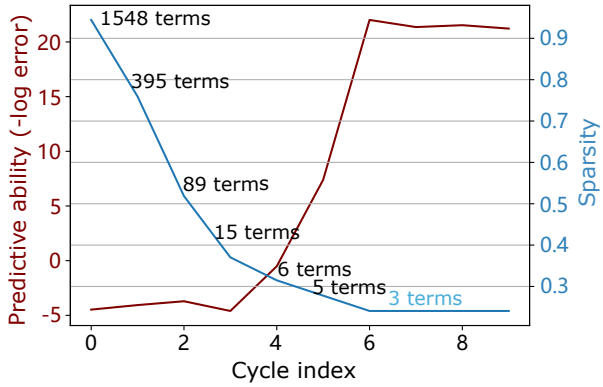


(b) The sparsity and predictive ability of the MathONet in each cycle.

Figure 5.4: The sparsity, predictive ability and weights of the MathONet generated in each cycle, which aims to discover governing equation of model x of Lorenz system. a) The nonzero weights of the MathONet generated in each cycle. The horizontal axis represents the combination of non-zero weights in the MathONet generated in each cycle. The vertical axis denotes the index of training cycles. The expression at each turning line (the cliff) represents the governing equation identified in the corresponding cycle. b) The sparsity and prediction ability of the MathONet identified in each cycle. The model becomes more and more sparse and has more and more predictive ability. The annotation next to the sparsity line represents the number of identified mathematical terms of the corresponding cycle. The first cycle represents the result identified by the sparse group Lasso method, which is still redundant (267 terms), and the prediction ability is low. The correct structure and coefficients can be identified around 7 cycles. *The same follows with other sparsity, predictive ability and weights plots.*

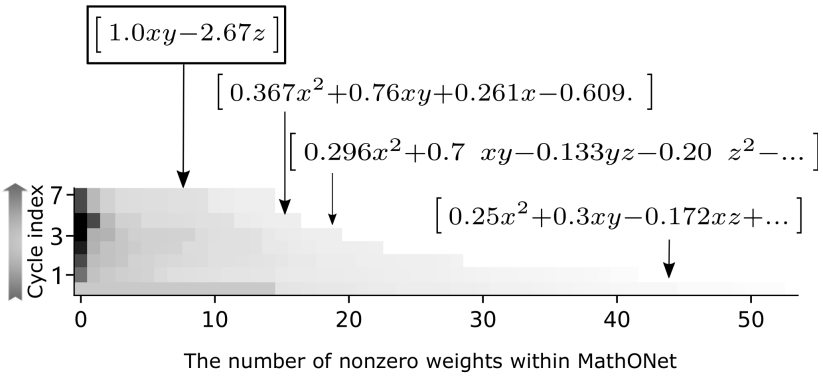


(a) The number of nonzero weights of the MathONet in each cycle.

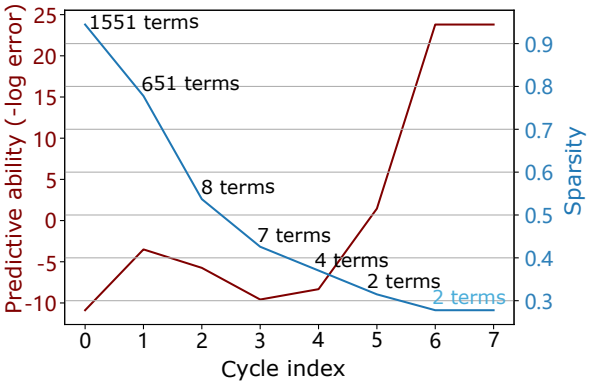


(b) The sparsity and predictive ability of the MathONet in each cycle.

Figure 5.5: The sparsity, predictive ability and weights of the MathONet generated in each cycle, which aims to discover governing equation of model y of Lorenz system.



(a) The number of nonzero weights of the MathONet generated in each cycle.



(b) The sparsity and predictive ability of the MathONet generated in each cycle.

Figure 5.6: The sparsity, predictive ability and weights of the MathONet generated in each cycle, which aims to discover governing equation of model z of Lorenz system.

Table 5.1: The identified governing equations for the chaotic Lorenz system with noisy measurements

Noise (σ)	Identified governing equations
0	$\dot{x} = -9.99999935280567x + 9.99999935280567y$ $\dot{y} = -0.999997263120259xz + 27.9998710115329x - 0.999953545630988y$ $\dot{z} = 0.99999998781167xy - 2.66666669723636z$
0.1	$\dot{x} = -9.99999496925967x + 9.99999952895684y$ $\dot{y} = -1.00052719172829xz + 28.0258380824985x - 1.00998669015022y$ $\dot{z} = 0.999999113716832xy - 2.66666544271354z$
1	$\dot{x} = -10.0033841069518x + 10.0034298967151y$ $\dot{y} = -1.00010106902181xz + 28.0040216562134x - 1.00069798536041y$ $\dot{z} = 0.999929109879073xy - 2.66622599261675z$
10	$\dot{x} = -9.99893134664084x + 10.0048109236265y$ $\dot{y} = -1.0092184658872xz + 28.4682092943604x - 1.1929184472878y$ $\dot{z} = 1.00051935274034xy - 2.66709953162919z$

5

The predicted distribution of the identified model is also investigated. We sampled a total of 1000 times based on the identified model and parameters. The first row in Fig. 5.7 shows each model's predicted mean and variance on the dataset without noise. Although the variance is very small almost for all data points, it still can be observed that the predicted distribution spreads a bigger range around the turning points, which means more training data is required around these points.

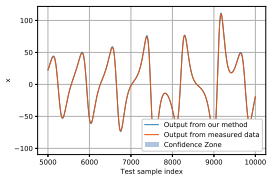
RESULT ON NOISY DATASET

To explore the robustness of the proposed method with noisy derivatives measurement, a Gaussian noise $\xi = \mathcal{N}(0, \sigma^2)$ with $\sigma \in \{0.01, 1, 10\}$ is added to the exact derivatives, respectively. The experiments were implemented with the same experiment setting as described in Section 5.4.1.2. As shown in Table 5.1, both structure and parameters are correctly identified even under the large noise value ($\sigma = 10$). Besides, although the simulation accuracy of attractor dynamics decreases with the increase of noise, the coefficients σ, ω, γ can still be determined accurately within 0.5% around the true value. The detailed identified structures and parameters are in Table 5.1. Fig. 5.7 is the prediction uncertainty for each dataset, which shows that the predicted uncertainty of the identified model improves along with increasing noise.

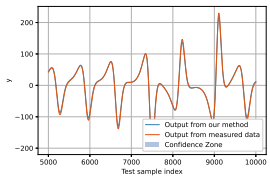
5.4.2. LOTKA-VOLTERRA SYSTEM

SYSTEM DESCRIPTION

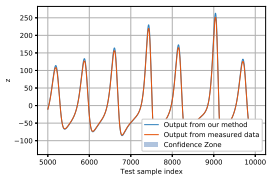
The Lotka-Volterra system is known as a general framework of an ecological system that can describe the dynamic relationship between the natural population of a predator and prey through time [2]. With two main assumptions: a) the growth rate of the population is proportional to its size; b) the predator population can only get food from the prey population, and the food for the prey population is supplied sufficiently at all times. The Lotka-Volterra equation can be described by a pair of deterministic first-order nonlinear



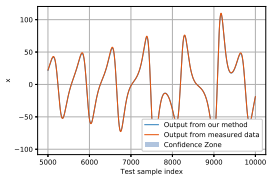
(a) Model x, $\sigma = 0$



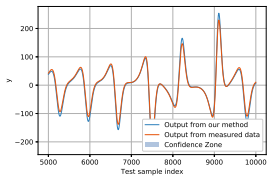
(b) Model y, $\sigma = 0$



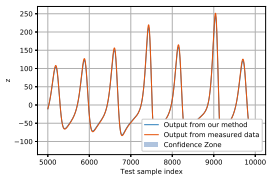
(c) Model z, $\sigma = 0$



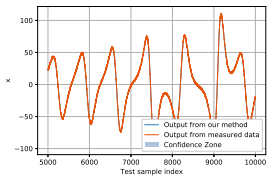
(d) Model x, $\sigma = 0.01$



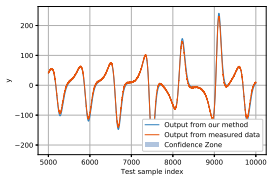
(e) Model y, $\sigma = 0.01$



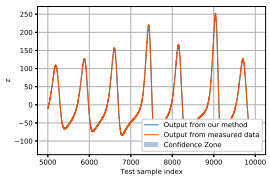
(f) Model z, $\sigma = 0.01$



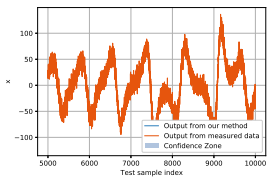
(g) Model x, $\sigma = 1$



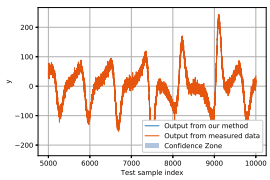
(h) Model y, $\sigma = 1$



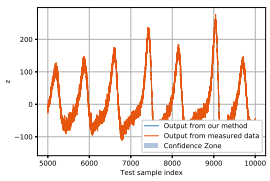
(i) Model z, $\sigma = 1$



(j) Model x, $\sigma = 10$



(k) Model y, $\sigma = 10$



(l) Model z, $\sigma = 10$

Figure 5.7: Predicted distribution for the identified Lorenz system generated from data with different noise $\xi = \mathcal{N}(0, \sigma^2)$, where $\sigma \in \{0.01, 1, 10\}$.The blue curve stands for the best-predicted output of our method. The shaded area represents the model uncertainty (showing 2 standard deviations)

ODEs:

$$\dot{x} = \alpha x - \omega xy, \qquad \dot{y} = \delta xy - \gamma y \tag{5.13}$$

where $\dot{x}$ and $\dot{y}$ represents the instantaneous growth rates of the prey (x) and predator (y), respectively. It can be found that the growth rate of each species is determined by two factors, i.e., the population of itself and the interaction with the other species.

EXPERIMENT SETUP

The coefficients $\alpha, \omega, \delta, \gamma$ in Eq. 5.13 are the positive real parameters identified in this experiment and set as $1.3, 0.9, 0.8, 1.8$, respectively. Fig. 5.8a shows the generated prey and predator population along time. Specifically, Fig. 5.8b illustrates the dynamic changes of the prey and predator population in a circle of growth and decline. The input and output data with the dimension 300×2 is generated, where 300 represents the number of data samples and 2 denotes the number of features. The ratio of training data and test data is set to 90% : 10%. The regularization parameters λ and λ_g are selected from the alternatives among $\{1e^{-2}, 1e^{-4}, 1e^{-6}, 1e^{-8}, 1e^{-10}\}$ and are decayed to one-tenth every 800 epochs. For each λ (λ_g), 10 repeated experiments are implemented with differing weight initialization.

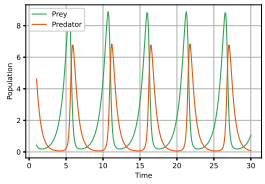
RESULT

The identified model with the best prediction accuracy and in line with Occam's razor principle is selected as the best model. The identified system equations are:

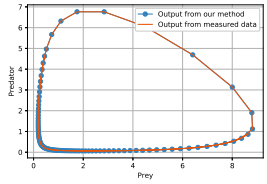
$$\dot{x} = 1.300x - 0.900xy \tag{5.14a}$$

$$\dot{y} = 0.800xy - 1.800y \tag{5.14b}$$

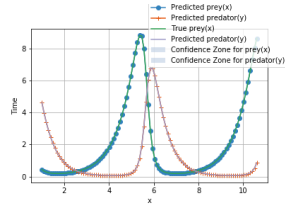
It can be observed that the identified equations are the same as the theoretical system



(a) Predator and prey population



(b) Phase-space plot of solution.

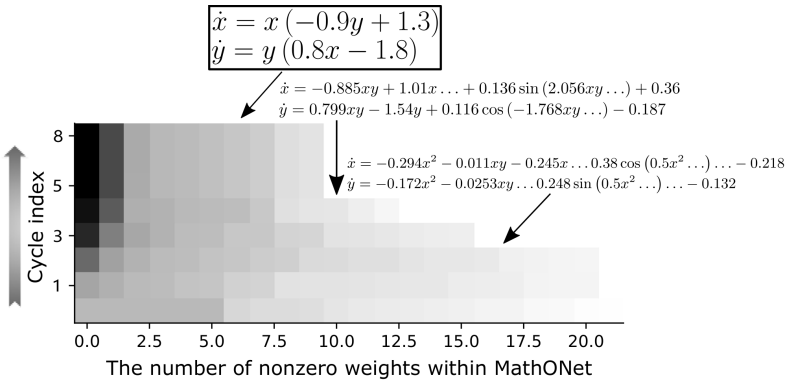


(c) Predicted output.

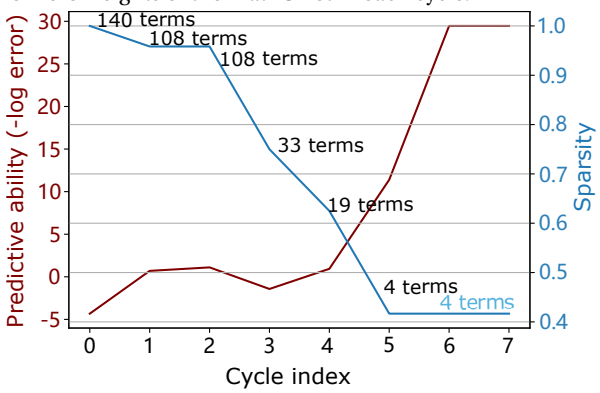
Figure 5.8: The solution of the prey and predator population with the initial conditions $x(0) = 0.442, y(0) = 4.628$. a) The data of the prey and predator population generated by (5.13); b) Phase-space plot for the predator and prey population of the identified model (5.14); c) The uncertainty of the predicted output.

Eq. (5.13). Fig. 5.9a shows the searching process of the algorithm. Fig. 5.9b shows the change of predictive ability and model sparsity, and the annotations along the sparsity line represent the number of retained mathematical terms of each cycle. It can be observed that the algorithm can identify a model with only 4 terms that originate from the

initialized structure with 140 terms. The predictive ability improves as the model complexity decreases.



(a) The number of nonzero weights of the MathONet in each cycle.



(b) The sparsity and predictive ability of the MathONet in each cycle.

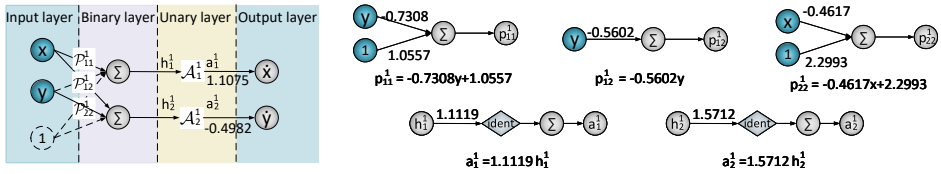
Figure 5.9: The sparsity, predictive ability and weights of the MathONet generated in each cycle of the Lotka-Volterra system.

Fig. 5.10 shows the identified MathONet. It can be found that the introduced constant input is removed from the input layer, which is in accordance with Eq. (5.13). The PolyNet between input feature y and the first hidden neuron is also identified to be redundant. Although the other basic modules are retained, the identified structures are sparse and equivalent to a simple mathematical expression, as shown in Fig. 5.10(b). We also investigate the simulated output and predicted distribution of the identified model, which is shown in Fig. 5.8.

5.4.3. FISHER-KPP (KOLMOGOROV–PETROVSKY–PISKUNOV) EQUATION

SYSTEM DESCRIPTION

The Fisher equation, also known as Kolmogorov–Petrovsky–Piskunov (KPP) equation, is a typical semilinear reaction-diffusion equation that describes the population growth and



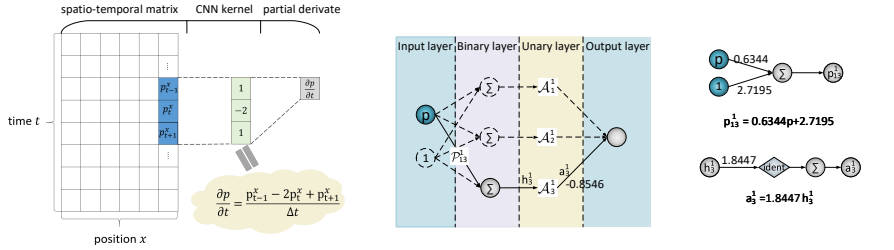
(a) Identified MathONet (b) Identified PolyNet and OperNet

Figure 5.10: Identified MathONet model for Lotka-Volterra system. The MathONet is initialized with 1 hidden layer and 2 hidden neurons. The OperNet includes 3 unary functions, i.e. `identity`,sin,cos. (b)The identified PolyNets and OperNets, where the dotted line are the identified redundant parts.

propagation of a species [8]. The spatio-temporal dynamics of Fisher-KPP can be represented by a PDE:

$$\frac{\partial p}{\partial t} = d\frac{\partial^2 p}{\partial x^2} + r\,p(1-p) \tag{5.15}$$

where p denotes the population density. $x \in [0,1]$ represents the coordinate measurement position of a species. $t \in [0,T]$ stands for the time of generation. $d = 6.25$ is a constant denoting the diffusion coefficient. $r = 1$ represents the intensity in favor of local population growth, assumed to be independent of p. The dataset is generated by the Julia code provided in [35]. The dimension of the dataset is 26×11, in which 11 refers to the number of samples for time; and 26 refers to the number of samples for the position.



(a) the partial derivate represented by a CNN stencil (b) Identified MathONet (c) Identified PolyNet and OperNet

Figure 5.11: An illustration to model the PDE by combining a MathONet and a CNN with a special kernel. (a) An explanation of the partial derivative $\frac{\partial p}{\partial t}$ represented by a CNN stencil with kernel $[1,-2,1]$. (b) Identified MathONet model for Fisher-KPP system. The MathONet is initialized with 1 hidden layer and 3 hidden neurons. The OperNet includes 3 basic unary functions, i.e. `identity`,sin,cos. Only two basic modules, i.e., $\mathscr{P}^1_{13}$ and $\mathscr{A}^1_3$ are retained in the graph. (c) The retained PolyNet and OperNet.

EXPERIMENT SETUP

As in Eq. (5.15), a time history of the population density p and the derivative $\frac{\partial p}{\partial t}$ are used as input and output. The regularization parameters λ and λ_g are selected from the al-

ternatives among $\{1e^{-8}, 1e^{-10}, 1e^{-12}, 1e^{-14}, 1e^{-16}\}$ and are decayed to one-tenth every 800 epochs. For each hyper-parameter, the identification procedure is repeated 10 times with differing weight initializations. The best model is selected according to the prediction accuracy, which is evaluated by the predicted mean square error.

As shown in Eq. (5.15), the one-dimensional Fisher equation consists of two parts, i.e., a polynomial representing the local growth item and a derivative operator. In this experiment, we use a MathONet to approximate the local growth item. Inspired by [35, 44], a discretized PDE can be interpreted as a convolutional layer that can exploit the relation between adjacent elements of a 2-D matrix [36]. In Fig. 5.11(a), we explain this affine transformation on a spatial-temporal matrix. It should be noted that an ideal CNN stencil should be $[1, -2, 1]$, which satisfies the physical constraint that the sum of CNN stencils is zero. To ensure the physical interpretability of the learned CNN stencil, previous work [35] imposed the physical constraint on the CNN kernel and obtained the desired result. In this work, we aim to discover the governing equations only from data. Therefore, we only include a simple convolutional neural network with the kernel size 3×1 is and try to learn the stencil.

RESULT

The optimal learned model is obtained with the identified equation:

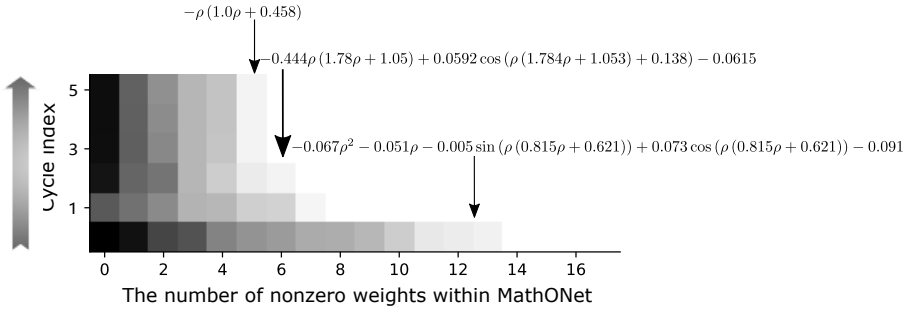
$$\frac{\partial p}{\partial t} = 3.2424 * \mathrm{CNN}(p) - 1.5765 p(0.6344 * p + 2.7195)$$

The identified CNN stencil is $[1.9276, -2.2245, 1.9276]$. If this stencil is rescaled to [1.000, −2.000,1.000], the equivalent equation will be $\frac{\partial p}{\partial t} = 6.250 * \mathrm{CNN}(p) + 1.000 p(1.000 - 1.000 p)$ which is almost the same as the true governing equations. Therefore, both the structure and coefficients of Eq. (5.15) can be accurately learned. The identified structure of MathONet is in Fig. 5.11(b).

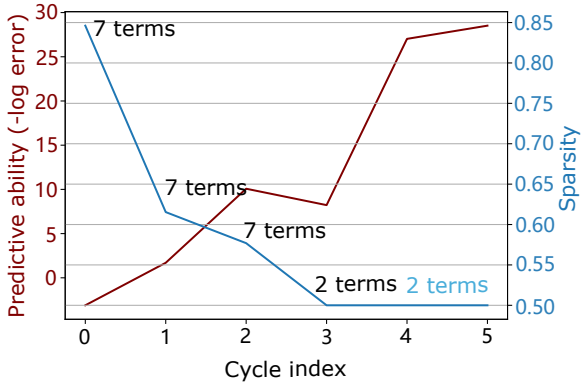
Fig. 5.12a shows the searching process of the algorithm for the governing equations. Fig. 5.12b shows the change of predictive ability and model sparsity, and the annotations along the sparsity line represent the number of retained mathematical terms of each cycle within the MathONet. It should be noted that MathONet contains the same number of mathematical terms (7 terms) in the first three cycles. However, the sparsity is gradually decreasing, which can also be observed in Fig. 5.12a. After some redundant connections are removed from the model, the retained connections can still represent the same or similar mathematical items represented by the redundant edges, so the number of mathematical terms remains unchanged.

5.5. DISCUSSION

Regularization parameter λ and λ_g in Algorithm 5: The regularization parameters λ and λ_g in Eq. (5.10) needs to be properly tuned for network training. With the Bayesian approach that calculates ψ and ψ_g as the determining factors for redundancy, the effort for tuning λ can be saved a lot. Typically, λ is also the necessity for the sparse group Lasso method as shown in Eq. (5.8). More strict conditions are required to discover governing equations through sparse group Lasso, including proper network initialization and an extensive computational resource used for tuning λ. It should be noted that the sparse group



(a) The number of nonzero weights of the MathONet in each cycle.



(b) The sparsity and predictive ability of the MathONet in each cycle.

Figure 5.12: The sparsity, predictive ability and weights of the MathONet generated in each cycle of the Fisher-KPP.

Lasso is the first cycle in our algorithm, which is applied for all experiments. However, it is challenging to discover governing equations precisely and efficiently (see Fig. 5.3(b) and Fig. 5.6b for Lorenz experiment).

Comparison with deep ensemble: Deep ensemble is a learning paradigm to improve generalization ability by training a set of deep neural network models with the same structures and random initializations [58]. An ensemble includes high performing models weighted by their posterior probabilities for better accuracy and variance reduction [54]. In this work, we also train a set of MathONet models starting from random initialization using the Bayesian approach. However, instead of averaging on an ensemble, a single model is selected by evaluating its performance. Although a single point mass may cause worse predictions with flawed assumptions (e.g. improper prior distribution [39]), it also alleviates the issue of a tremendous computational expense that deep ensembles may incur.

Comparison with variational inference: Variational inference (VI) is another typical approximation method for Bayesian inference by minimizing the Kullback-Leibler divergence between an assumed approximated posterior and the true posterior distribution. VI method can provide bounds on probabilities of interest and yield deterministic approximation procedures without tuning regularization parameters [18]. Its applications on model compression have been explicitly interpreted in [14, 32] However, the manual selection of proper pruning thresholds is also required, hindering its compression efficiency for complex models. In contrast, the Laplace approximation method can be implemented more efficiently and extended to complex models.

5.6. RELATED WORK

System identification Data-driven discovery of the governing equations has become an active research area for many years [6, 40]. With the provided heuristics and expert guidance, several pioneering works started from rediscovering known governing equations in specific disciplines (e.g., Proust's law in chemistry [22], ideal gas law [23]) with simulated data [26]. Further work was also implemented for equation discovery of ecological applications with a real collected dataset [7]. Another typical classical method is system identification, which aims to obtain an approximated mathematical model by identifying the model parameters [29]. However, since the typical prerequisite for system identification is that the model structure is known, these methods are impractical for the domains without known mathematical laws(e.g., neuroscience, cell biology, finance, epidemiology) [7, 53]. To improve generalization, the method of discovering both structure and coefficients of the governing equations becomes a key research direction.

Symbolic regression Symbolic regression can be used to discover mathematical operations in governing equations without prior domain knowledge [10, 45]. In [51], symbolic regression is applied for unsupervised learning of motion equations of the object from a distorted unlabelled video. Other successful applications include automated refinement and reverse engineering for metabolic networks [3, 46], synchronization control of oscillator networks [11], motion prediction of harmonic oscillator [33] and constructing smooth value functions for reinforcement learning [21], etc. The symbolic regression method can

learn the mathematical expression by searching over a space of basic arithmetic operations with brute force [17, 50] or self-evolved by genetic algorithms [1, 10, 45]. However, these approaches are too computationally expensive to scale to high-dimensional systems and large datasets and sensitivity to initialization [19].

Sparse regression The sparse regression is a promising technique that can determine (almost) the fewest equation terms to describe a system. In [4], sparse identification of nonlinear dynamics (SINDy) algorithm is proposed to identify the governing equations as a linear combination of basis functions selected from a pre-built function dictionary. The SINDy method has achieved success on many benchmarks, e.g., fluid dynamics [4], chemical kinetics [15]. It was also expanded to address the model recovery of dynamic systems following abrupt changes [34] and the discovery of partial differential equations [42]. These works show that the sparse regression methods provide an effective manner to identify the governing equations. However, these approaches also suffer from the nontrivial task of choosing appropriate basis functions, limiting their capacity for more general applications.

Both the symbolic and sparse regression techniques explore the governing equations within an ample space of possibly nonlinear mathematical terms. The comparisons and discussions between them can be found in a recent work [48] that identifies a distillation column's dynamical equations.

5.7. CONCLUSION

We present a method to learn governing equations of dynamic systems composed of basic mathematical operations, i.e., unary and binary operations. The governing equations are formulated as a DenseNet-like hierarchical structure, termed as MathONet. The governing equation discovery problem can be formulated as a deep neural network compression problem. The sparse Bayesian deep learning algorithm with fused prior is proposed to find a sparser solution than sparse group Lasso-type algorithms. The experiment result shows that the proposed method effectively discovers general differential equations, including linear and nonlinear differential equations, ordinary differential equations (ODEs), or partial differential equations (PDEs).

BIBLIOGRAPHY

[1] Thomas Bäck, David B Fogel, and Zbigniew Michalewicz. *Evolutionary computation 1: Basic algorithms and operators*. CRC press, 2018.

[2] Alan A Berryman. "The orgins and evolution of predator-prey theory". In: *Ecology* 73.5 (1992), pp. 1530–1535.

[3] Josh Bongard and Hod Lipson. "Automated reverse engineering of nonlinear dynamical systems". In: *Proceedings of the National Academy of Sciences* 104.24 (2007), pp. 9943–9948.

[4] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proceedings of the National Academy of Sciences* 113.15 (2016), pp. 3932–3937. ISSN: 0027-8424. DOI: 10.1073/pnas.1517384113. eprint: https://www.pnas.org/content/113/15/3932.full.pdf.

[5] Han Cai, Ligeng Zhu, and Song Han. "ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware". In: *International Conference on Learning Representations*. 2019. URL: https://arxiv.org/pdf/1812.00332.pdf.

[6] Giuseppe Carleo et al. "Machine learning and the physical sciences". In: *Reviews of Modern Physics* 91.4 (2019), p. 045002.

[7] Sašo Džeroski et al. "Equation discovery with ecological applications". In: *Machine learning methods for ecological applications* (1999), pp. 185–207.

[8] Ronald Aylmer Fisher. "The wave of advance of advantageous genes". In: *Annals of eugenics* 7.4 (1937), pp. 355–369.

[9] Yarin Gal. "Uncertainty in deep learning". In: *University of Cambridge* 1.3 (2016).

[10] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN: 0201157675.

[11] Julien Gout et al. "Synchronization control of oscillator networks using symbolic regression". In: *Nonlinear Dynamics* 91.2 (2018), pp. 1001–1021.

[12] Song Han, Huizi Mao, and William J. Dally. "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding". In: *International Conference on Learning Representations*. 2016.

[13] B. Hassibi, D. G. Stork, and G. J. Wolff. "Optimal Brain Surgeon and general network pruning". In: *IEEE International Conference on Neural Networks*. Mar. 1993, 293–299 vol.1. DOI: 10.1109/ICNN.1993.298572.

[14] 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*. 1993, pp. 5–13.

[15] Moritz Hoffmann, Christoph Fröhner, and Frank Noé. "Reactive SINDy: Discovering governing reactions from concentration data". In: *The Journal of chemical physics* 150.2 (2019), p. 025101.

[16] Gao Huang et al. "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[17] Ying Jin et al. "Bayesian symbolic regression". In: *arXiv preprint arXiv:1910.08892* (2019).

[18] Michael I Jordan et al. "An introduction to variational methods for graphical models". In: *Machine learning* 37.2 (1999), pp. 183–233.

[19] Samuel Kim et al. "Integration of neural network-based symbolic regression in deep learning for scientific discovery". In: *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[20] Seyoung Kim and Eric P Xing. "Tree-guided group lasso for multi-task regression with structured sparsity". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. 2010, pp. 543–550.

[21] Jiří Kubalík et al. *Symbolic Regression Methods for Reinforcement Learning*. 2019. arXiv: 1903.09688 [cs.LG].

[22] Pat Langley. "Data-driven discovery of physical laws". In: *Cognitive Science* 5.1 (1981), pp. 31–54.

[23] Pat Langley, Gary L Bradshaw, and Herbert A Simon. "Rediscovering chemistry with the BACON system". In: *Machine learning*. Springer, 1983, pp. 307–329.

[24] Yann LeCun, John S Denker, and Sara A Solla. "Optimal brain damage". In: *Advances in neural information processing systems*. 1990, pp. 598–605.

[25] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. "Snip: Single-shot network pruning based on connection sensitivity". In: *arXiv preprint arXiv:1810.02340* (2018).

[26] Douglas B Lenat. "The role of heuristics in learning by discovery: Three case studies". In: *Machine learning*. Springer, 1983, pp. 243–306.

[27] Chia-Chiao Lin and Lee A Segel. *Mathematics applied to deterministic problems in the natural sciences*. SIAM, 1988.

[28] Hanxiao Liu, Karen Simonyan, and Yiming Yang. "DARTS: Differentiable Architecture Search". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=S1eYHoC5FX.

[29] Lennart Ljung. "System identification". In: *Wiley encyclopedia of electrical and electronics engineering* (1999), pp. 1–19.

[30] Antonio Loquercio, Mattia Segu, and Davide Scaramuzza. "A general framework for uncertainty estimation in deep learning". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3153–3160.

[31] Edward N Lorenz. "Deterministic nonperiodic flow". In: *Journal of the atmospheric sciences* 20.2 (1963), pp. 130–141.

[32] Christos Louizos, Karen Ullrich, and Max Welling. “Bayesian compression for deep learning”. In: *arXiv preprint arXiv:1705.08665* (2017).

[33] Markus Quade et al. “Prediction of dynamical systems by symbolic regression”. In: *Phys. Rev. E* 94 (1 July 2016), p. 012214. DOI: 10.1103/PhysRevE.94.012214.

[34] Markus Quade et al. “Sparse identification of nonlinear dynamics for rapid model recovery”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 28.6 (2018), p. 063116.

[35] Christopher Rackauckas et al. *Universal Differential Equations for Scientific Machine Learning*. 2020. arXiv: 2001.04385 [cs.LG].

[36] Rajat Raina, Anand Madhavan, and Andrew Y Ng. “Large-scale deep unsupervised learning using graphics processors”. In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 873–880.

[37] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707.

[38] Anders Rasmuson et al. *Mathematical modeling in chemical engineering*. Cambridge University Press, 2014.

[39] Carl Edward Rasmussen. “Gaussian processes in machine learning”. In: *Summer school on machine learning*. Springer. 2003, pp. 63–71.

[40] Ribana Roscher et al. “Explainable machine learning for scientific insights and discoveries”. In: *IEEE Access* 8 (2020), pp. 42200–42216.

[41] Samuel H Rudy, J Nathan Kutz, and Steven L Brunton. “Deep learning of dynamics and signal-noise decomposition with time-stepping constraints”. In: *Journal of Computational Physics* 396 (2019), pp. 483–506.

[42] Samuel H Rudy et al. “Data-driven discovery of partial differential equations”. In: *Science Advances* 3.4 (2017), e1602614.

[43] Laura von Rueden et al. “Informed Machine Learning–A Taxonomy and Survey of Integrating Knowledge into Learning Systems”. In: *arXiv preprint arXiv:1903.12394* (2019).

[44] Lars Ruthotto and Eldad Haber. “Deep neural networks motivated by partial differential equations”. In: *Journal of Mathematical Imaging and Vision* 62 (2020), pp. 352–364.

[45] Michael Schmidt and Hod Lipson. “Distilling Free-Form Natural Laws from Experimental Data”. In: *Science* 324.5923 (2009), pp. 81–85. ISSN: 0036-8075. DOI: 10.1126/science.1165893. eprint: https://science.sciencemag.org/content/324/5923/81.full.pdf.

[46] Michael D Schmidt et al. “Automated refinement and inference of analytical models for metabolic networks”. In: *Physical Biology* 8.5 (Aug. 2011), p. 055011. DOI: 10.1088/1478-3975/8/5/055011.

[47] Noah Simon et al. “A sparse-group lasso”. In: *Journal of computational and graphical statistics* 22.2 (2013), pp. 231–245.

[48] Renganathan Subramanian, Raghav Rajesh Moar, and Shweta Singh. “White-box Machine learning approaches to identify governing equations for overall dynamics of manufacturing systems: A case study on distillation column”. In: *Machine Learning with Applications* 3 (2021), p. 100014. ISSN: 2666-8270. DOI: https://doi.org/10.1016/j.mlwa.2020.100014.

[49] Robert Tibshirani. “Regression shrinkage and selection via the lasso”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288.

[50] Silviu-Marian Udrescu and Max Tegmark. “AI Feynman: A physics-inspired method for symbolic regression”. In: *Science Advances* 6.16 (2020), eaay2631.

[51] Silviu-Marian Udrescu and Max Tegmark. “Symbolic pregression: Discovering physical laws from raw distorted video”. In: *arXiv preprint arXiv:2005.11212* (2020).

[52] Wei Wen et al. “Learning structured sparsity in deep neural networks”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 2074–2082.

[53] Jared Willard et al. “Integrating physics-based modeling with machine learning: A survey”. In: *arXiv preprint arXiv:2003.04919* (2020).

[54] Andrew Gordon Wilson. “The case for Bayesian deep learning”. In: *arXiv preprint arXiv:2001.10995* (2020).

[55] David J Wollkind and Chernyk. *Comprehensive Applied Mathematical Modeling in the Natural and Engineering Sciences*. Springer, 2017.

[56] Ming Yuan and Yi Lin. “Model selection and estimation in regression with grouped variables”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.1 (2006), pp. 49–67.

[57] Hongpeng Zhou et al. “Bayesnas: A bayesian approach for neural architecture search”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 7603–7613.

[58] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. “Ensembling neural networks: Many could be better than all”. In: *Artificial Intelligence* 137.1 (2002), pp. 239–263. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(02)00190-X. URL: https://www.sciencedirect.com/science/article/pii/S000437020200190X.

6

APPLICATION TO HIGH DIMENSIONAL DATA

This chapter implements the proposed sparse Bayesian deep learning algorithm on two deep learning topics with high-dimensional datasets, i.e., Neural Architecture Search (NAS) and Neural Network Compression. One-Shot NAS is a promising method to significantly reduce the search time without any separate training. It can be treated as a Network Compression problem on the architecture parameters from an over-parameterized network. However, there are two issues associated with most one-shot NAS methods. First, the dependencies between a node and its predecessor and successors are often disregarded, which result in improper treatment over zero *operations. Second, architecture parameters pruning based on their magnitude is questionable. In this chapter, we employ the proposed sparse Bayesian deep learning learning approach to alleviate these two issues by modelling architecture parameters using* hierarchical automatic relevance determination *(HARD) priors. Unlike other NAS methods, we train the over-parameterized network for only* one *epoch then update the architecture. Impressively, this enabled us to find the architecture on CIFAR-10 within only* 0.2 *GPU days using a single GPU. Competitive performance can be also achieved by transferring to ImageNet. As a byproduct, this chapter also presents the application of our approach to neural network compression tasks. Several typical fully connected neural networks and convolutional neural networks are compressed on three datasets, including MNIST, CIFAR10, and Atrial Fibrillation dataset. The experiment result shows that the sparse networks can be obtained without accuracy deterioration.*

Parts of this chapter have been published in the International Conference on Machine Learning (2019) [42]

6.1. INTRODUCTION

Neural Architecture Search (NAS), the process of automating architecture engineering, is thus a logical next step in automating machine learning since [43]. There are basically three existing frameworks for neural architecture search. Reinforcement learning based NAS [1, 7, 41, 43, 44] methods take the generation of a neural architecture as an agent's action with the action space identical to the search space. More recent neuro-evolutionary approaches [9, 21, 26, 30, 31, 36] use gradient-based methods for optimizing weights and solely use evolutionary algorithms for optimizing the neural architecture itself. However, these two frameworks take enormous computational power when compared to a search using a single GPU. One-Shot based NAS is a promising approach to significantly reduce the search time without any separate training, which treats all architectures as different sub-graphs of a super-graph (the one-shot model) and shares weights between architectures that have the edges of this super-graph in common [2, 5, 6, 20, 29, 32, 37, 38, 40]. A comprehensive survey on Neural Architecture Search can be found in [10].

Our approach is a one-shot based NAS solution which treats NAS as a Network Compression/pruning problem on the architecture parameters from an over-parameterized network. However, despite it's remarkable less searching time compared to reinforcement learning and neuro-evolutionary approaches, we can identify a number of significant and practical disadvantages of the current one-shot based NAS. First, the dependencies between a node and its predecessors and successors are disregarded in the process of identifying the redundant connections. This is mainly motivated by the improper treatment of *zero* operations. On one hand, the logit of *zero* may dominate some of the edges while the child network still has other non-zero edges to keep it connected [6, 20, 37, 40], for example, node 2 in Figure6.1a. Similarly, as shown in Figure 1 of [37], the probability of invalid/disconnected graph sampled will be $\frac{511}{1024}$ when there are three non-zero plus one *zero* operation. Though post-processing to safely remove isolated nodes is possible, *e.g.*, for chain-like structure, it demands extensive extra computations to reconstruct the graph for complex search space with additional layer types and multiple branches and skip connections. This may prevent the use of modern network structure as the backbone such as DenseNet [16], newly designed motifs [21] and complex computer vision tasks such as semantic segmentation [18]. On the other hand, *zero* operations should have higher priority to rule out other possible operations, since *zero* operations equal to all *non-zero* operations not being selected. Second, most one-shot NAS methods [6, 11, 20, 37, 40] rely on the magnitude of architecture parameters to prune redundant parts and this is not necessarily true. From the perspective of Network Compression [17], magnitude-based metric depends on the scale of weights, thus requiring pre-training and is very sensitive to the architectural choices. Also the magnitude does not necessarily imply the optimal edge. Unfortunately, these drawbacks exist not only in Network Compression but also in one-shot NAS.

In this work, we applied the proposed sparse Bayesian deep learning approach to alleviate these two issues simultaneously. We model the architecture parameters by a *hierarchical automatic relevance determination* (HARD) prior. The dependency can be translated by multiplication and addition of some independent Gaussian distributions. The classic Bayesian learning framework [24, 27, 34] prevents overfitting and promotes sparsity by specifying sparse priors. The uncertainty of the parameter distribution can be used

as a new metric to prune the redundant parts if its associated entropy $\frac{1}{2}\ln\left(2\pi e\gamma_{jk}^{o'}\right)$ is non-positive. The majority of parameters are automatically zeroed out during the learning process.

Our Contributions

- **Bayesian approach:** BayesNAS is the first Bayesian approach for one-shot NAS. Therefore, our approach shares the advantages of Bayesian learning, which prevents overfitting and does not require tuning a lot of hyperparameters. Hierarchical sparse priors are used to model the architecture parameters. Priors can not only promote sparsity, but model the dependency between a node and its predecessors and successors ensuring a connected derived graph after pruning. Furthermore, it provides a principled way to prioritize *zero* operations over other *non-zero* operations. In our experiment on CIFAR-10, we found that the variance of the prior, as well as that of the posterior, is several magnitudes smaller than the posterior mean, which renders a good metric for architecture parameters pruning.

- **Simple and fast search:** Our algorithm is formulated simply as an iteratively re-weighted ℓ_1 type algorithm where the re-weighting coefficients used for the next iteration are computed not only from the value of the current solution but also from its posterior variance. The update of posterior variance is based on Laplace approximation in Bayesian learning which requires computation of the inverse Hessian of log likelihood. To make the computation for large networks feasible, a fast Hessian calculation method is proposed. In our experiment, we train the model for only *one* epoch before calculating the Hessian to update the posterior variance. Therefore, the search time for very deep neural networks can be kept within 0.2 GPU days.

- **Network compression:** As a byproduct, our approach can be extended directly to Network Compression by enforcing various structural sparsities over network parameters. Extremely sparse models can be obtained at the cost of minimal or no loss in accuracy across all tested architectures. This can be effortlessly integrated into BayesNAS to find sparse architecture along with sparse kernels for resource-limited hardware.

6.2. SEARCH SPACE DESIGN

The search space defines which neural architectures a NAS approach might discover in principle. Designing a good search space is a challenging problem for NAS. Some works [6, 7, 20, 29, 40, 43, 44] have proposed that the search space could be represented by a Directed Acyclic Graph (DAG). We denote e_{ij} as the edge from node i to node j and o_{ij} stands for the operation that is associated with edge e_{ij}.

Similar to other one-shot based NAS approaches [2, 6, 11, 20, 40], we also include (different or same) scaling scalars over all operations of all edges to control the information flow, denoted as w_{ij}^{o} which also represent the architecture parameters. The output of a

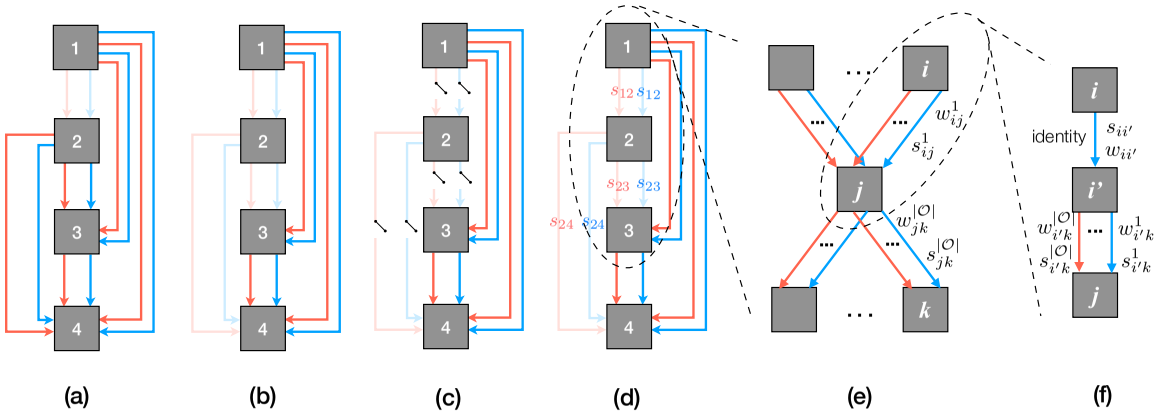


Figure 6.1: An illustration of BayesNAS: (a) disconnected graph with isolated node 2 caused by disregard for dependency; (b) expected connected graph with no connection from node 2 to 3 and from node 2 to 4; (c) illustration about dependency with predecessor's (e_{12}) superior control over its successors (e_{23} and e_{24}) (d) designed switches realizing the dependency and determining "on or off" of the edge; (e) elementary multi-input-multi-output motif for a graph; (f) prioritized *zero* operation over other *non-zero* operations.

mixed operation $o_{ij}, i < j$ is defined based on the outputs of its edge

$$o_j(z_i) = \sum_{o \in \mathcal{O}} w_{ij}^o o_{ij}(z_i). \tag{6.1}$$

Then z_j can be obtained as $\sum_{i<j} o_j(z_i)$.

To this end, the objective is to learn a sparse subgraph while maintaining the accuracy of the over-parameterized DAG [2]. Let us formulate the search problem as an optimization problem. Given a dataset $\mathbf{D} = (\mathbf{X}, \mathbf{Y}) = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ and the desired sparsity level κ (*i.e.*, the number of non-zero edges), one-shot NAS problem can be written as an optimization problem with the following constraints:

$$\begin{aligned} \min_W L(\mathbf{W}; \mathbf{D}) = \min_W \frac{1}{N} \sum_{n=1}^{N} \ell(\mathbf{y}_n, \mathrm{Net}(\mathbf{x}_n, \mathcal{W}, \mathbf{w})) \\ \text{s.t.} \quad W \in \mathbb{R}^{m^{\mathrm{net}} + m^{\mathrm{edge}}}, \quad \|\mathbf{w}\|_0 \leq \kappa^{\mathrm{edge}} \end{aligned} \tag{6.2}$$

where $\mathbf{W}$ are split into two parts: network parameters $\mathcal{W} = [\mathcal{W}_{ij}^o]$ and architecture parameters $\mathbf{w} = [w_{ij}^o]$ with dimension of m^{net} and m^{edge} respectively, and $\|\cdot\|_0$ is the standard ℓ_0 norm. The formulation in Eq. (6.2) can be substantiated by incorporating *zero* operations into $\mathcal{O}$ to allow removal of w_{ij}^o [6, 20] aiming to further reduce the size of cells and improve the design flexibility.

To alleviate the negative effect induced by the dependency and magnitude-based metric whose issues have been discussed in Introduction, for each w_{ij}^o, we introduce a switch s_{ij}^o that is analogous to the one used in an electric circuit. There are four features associated with these switches. First, the "on-off" status is not solely determined by its magnitude. Second, dependency will be taken into account, *i.e.*, the predecessor has superior control over its successors as illustrated in Fig. 6.1c. Third, s_{ij}^o is an auxiliary variable that will not be updated by gradient descent but computed directly to switch on or off the edge. Lastly, s_{ij}^o should work for both proxy and proxyless scenarios and can be better embedded into existing algorithmic frameworks [6, 11, 20]. The calculation method will be introduced later in Section 6.3.1.

Inspired by the hierarchical representation of a DAG [20, 21], we abstract a single motif as the building block of DAG, as shown in Fig. 6.1e. Apparently, any derived motif, path, or network can be constructed by such a multi-input-multi-output motif. It shows that a successor can have multiple predecessors and each predecessor can have multiple operations over each of its successors. Since the representation is general, each directed edge can be associated with some primitive operations (*e.g.*, convolution, pooling, etc.) and a node can represent output of motifs, cells, or a network.

6.3. IDENTIFICATION METHOD

6.3.1. DEPENDENCY-BASED PERFORMANCE ESTIMATION STRATEGY

Encoding the Dependency Logic In the following, we will formally state the criterion to identify the redundant connections in Proposition 2. The idea can be illustrated by Fig. 6.1b in which both the blue and red edges from node 2 to 3 and from node 2 to 4

might be non-zeros but should be removed as a consequence. To enable this and refer to the proposition 1 in Chapter. 5, we have the following proposition for NAS.

Proposition 2 *There is information flow from node j to k under operation o' as shown in Fig. 6.1e if and only if at least one operation of at least one predecessor of node j is non-zero and $w_{jk}^{o'}$ is also non-zero.*

Remark 4 *The same expression for Proposition 2 is: there is **no** information flow from node j to k under operation o' if and only if all the operation of all the predecessors of node j are zeros or $w_{jk}^{o'}$ is zero. This explains the incompleteness of the problem 6.2 as well as the possible phenomenon that non-zero edges become dysfunctional in Fig. 6.1b.*

Remark 5 *The expression to encode Proposition 2 is not unique. Some examples include but not limited to, e.g., $w_{jk}^{o'}\sum_{i<j}|w_{ij}^{o}|$, $w_{jk}^{o'}\sum_{i<j}\alpha_{ij}^{o}|w_{ij}^{o}|, \forall \alpha_{ij}^{o} \in (0,1]$, $w_{jk}^{o'}\sum_{i<j}(w_{ij}^{o})^2$. Apparently, ℓ_0 norm of these quantities are difficult to be included in a constraint in the optimization problem formulation in 6.2.*

As can be seen in Remark 5, we will construct a probability distribution jointly over $w_{jk}^{o'}$, $w_{ij}^{o}, \forall i<j$ in the sequel, denoted as

$$p(c(w_{jk}^{o'}, w_{ij}^{o})), \forall i<j. \tag{6.3}$$

where c is a possible expression like in Remark 5 to encode Proposition 2.

In the following, we will show how the "switch" s can be used to implement Proposition 2. If we assume s has two states $\{\text{ON}, \text{OFF}\}$, $w_{jk}^{o'}$ is redundant when $s_{jk}^{o'}$ is OFF *or* all s_{ij}^{o} are OFF, $\forall i<j, o \in \mathcal{O}$. How to use s to encode the redundancy of $w_{jk}^{o'}$, *i.e.*, $w_{jk}^{o'}\sum_{i<j}|w_{ij}^{o}| = 0$? One possible solution is

$$\bigcup_{i<j}\bigcup_{o\in\mathcal{O}} s_{ij}^{o} \cap s_{jk}^{o'} \quad \text{or} \quad \overline{\overline{\bigcup_{i<j}\bigcup_{o\in\mathcal{O}} s_{ij}^{o}} \cup \overline{s_{jk}^{o'}}} \tag{6.4}$$

If s is a continuous variable with $s=\infty$ for ON and 0 for OFF, set union and intersection can be arithmetically represented by addition and multiplication, respectively. s does not directly determine the magnitude of w but plays a role as uncertainty or confidence for zero magnitude.

A straightforward way to encode this logic is to assign a probability distribution, for example, Gaussian distribution, over $w_{jk}^{o'}$

$$p(w_{jk}^{o'}) = \mathcal{N}(w_{jk}^{o'}|0, s_{jk}^{o'}), \ \sum_{i<j} p(w_{ij}^{o}) = \sum_{i<j}\mathcal{N}(w_{ij}^{o}|0, s_{ij}^{o})$$

Since $w_{ij}^{o}, \forall i,j,o$ are independent to each other, we construct the following distribution to express Eq. (6.3):

$$
\begin{aligned}
p(c(w_{jk}^{o'}, w_{ij}^{o})) &\triangleq \mathcal{N}(w_{jk}^{o'}|0, s_{jk}^{o'}) \sum_{i<j} \mathcal{N}(w_{ij}^{o}|0, s_{ij}^{o}) \\
&= \mathcal{N}(w_{jk}^{o'}|0, s_{jk}^{o'}) \mathcal{N}\left(\sum_{i<j} \frac{s_{ij}^{o} w_{ij}^{o}}{\sum_{i<j} s_{ij}^{o}} |0, s_{ij}^{o} \right) \\
&= \mathcal{N}\left(w_{jk}^{o'} \sum_{i<j} \frac{s_{ij}^{o} w_{ij}^{o}}{\sum_{i<j} s_{ij}^{o}} |0, \gamma_{jk}^{o'} \right)
\end{aligned}
\tag{6.5}
$$

where

$$
\gamma_{jk}^{o'} \triangleq \left(\frac{1}{\sum_{i<j} \sum_{o \in \mathcal{O}} s_{ij}^{o}} + \frac{1}{s_{jk}^{o'}} \right)^{-1} . \tag{6.6}
$$

Since $s_{ij}^{o} > 0$ in Eq. (6.5) always holds, regardless of what s_{ij}^{o} is, we can use the following simpler alternative to substitute Eq. (6.5) to encode Proposition 2:

$$
p(c(w_{jk}^{o'}, w_{ij}^{o})) \triangleq \mathcal{N}\left(w_{jk}^{o'} \sum_{i<j} w_{ij}^{o} |0, \gamma_{jk}^{o'} \right). \tag{6.7}
$$

Interestingly, Eq. (6.7) and (6.4) are equivalent. This means that we may find an algorithm that is able to find the sparse solution in a probabilistic manner. However, Gaussian distribution, in general, does not promote sparsity. Fortunately, some classic yet powerful techniques in Bayesian learning are applicable, *i.e.*, sparse Bayesian learning (SBL) [28, 34] and automatic relevance determination (ARD) prior [25, 27] in Bayesian neural networks.

Zero Operation Ruling All In this paper, we do not include *zero* operation as a primitive operation. Instead, between node i and j we compulsively add one more node i' and allow only a single *identity* operation (see Fig. 6.1f). The associated weight $w_{ii'}$ is trainable and initialized to 1 as well as its switch $s_{ii'}$. The idea is that if $s_{ii'}$ is OFF, all the operations from i' to j will be disabled as a consequence. Then $\gamma_{jk}^{o'}$ in Eq. (6.6) can be substituted by

$$
\gamma_{jk}^{o'} \triangleq \left(\frac{1}{s_{ii'}} + \frac{1}{\sum_{i'<j} \sum_{o \in \mathcal{O}} s_{i'j}^{o}} + \frac{1}{s_{jk}^{o'}} \right)^{-1} . \tag{6.8}
$$

6.3.2. BAYESIAN LEARNING SEARCH STRATEGY

Bayesian Neural Network The likelihood for the network weights $\mathcal{W}$ and the noise precision σ^{-2} with data $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$ is

$$
p(\mathbf{Y}|\mathcal{W}, \mathbf{w}, \mathbf{X}, \sigma^2) = \prod_{n=1}^{N} \mathcal{N}(y_n | \mathrm{Net}(\mathbf{x}_n; \mathcal{W}, \mathbf{w}); \sigma^2) . \tag{6.9}
$$

To complete our probabilistic model, we specify a Gaussian prior distribution for each entry in each of the weight matrices in $\mathscr{W}$. In particular,

$$p(\mathscr{W} \mid \lambda) = \prod_{i<j} \prod_{o \in \mathscr{O}} \mathscr{N}(\mathscr{W}_{ij}^{o} \mid 0, \lambda^{-1}) \tag{6.10}$$

$$p(\mathbf{w} \mid \mathbf{s}) = \prod_{j<k} \prod_{o \in \mathscr{O}} \prod_{o' \in \mathscr{O}} \mathscr{N}\left(w_{jk}^{o'} \sum_{i<j} w_{ij}^{o} | 0, \gamma_{jk}^{o'} \right) \tag{6.11}$$

where $\gamma_{jk}^{o'}$ is defined in Eq. (6.8). σ^{-2}, λ and $\mathbf{s}$ are *hyperparameters*. Importantly, there is an individual hyperparameter associated independently with every edge weight and a single one with all network weight. Follow Mackay's evidence framework [24], 'hierarchical priors' are employed on the latent variables using Gamma priors on the inverse variances. The hyper-priors for σ^{-2}, λ and $\mathbf{s}$ are chosen to be a gamma distribution [3], *i.e.*, $p(\lambda) = \text{Gam}(\lambda \mid a^{\lambda}, b^{\lambda}), p(\beta) = \text{Gam}(\beta \mid a^{\beta}, b^{\beta})$ with $\beta = \sigma^{-2}$, and $p(s_{ij}^{o}) = \text{Gam}(s_{ij}^{o} \mid a^{s_{ij}^{o}}, b^{s_{ij}^{o}})$. Essentially, the choice of Gamma priors has the effect of making the marginal distribution of the latent variable prior the non-Gaussian Student's t therefore promoting the sparsity [34, Section 2 and 5.1]. To make these priors non-informative (*i.e.*, flat), we simply fix a and b to zero by assuming uniform scale priors for analysis and implementation. This formulation of prior distributions is a type of *hierarchically constructed automatic relevance determination* (HARD) prior which is built upon the classic ARD prior [27, 34].

The posterior distribution for the parameters $\mathscr{W}$, γ and λ can then be obtained by applying Bayes' rule:

$$\begin{aligned} & p(\mathscr{W}, \mathbf{w}, \lambda, \mathbf{s}, \sigma^2 \mid \mathscr{D}) \\ = & \frac{p(\mathbf{Y} \mid \mathbf{X}, \mathscr{W}, \mathbf{w}, \lambda, \mathbf{s}, \sigma^2) p(\mathscr{W} \mid \lambda) p(\mathbf{w} \mid \mathbf{s}) p(\lambda) p(\gamma) p(\sigma^2)}{p(\mathbf{Y} \mid \mathbf{X})}, \end{aligned} \tag{6.12}$$

where $p(\mathbf{Y} \mid \mathbf{X})$ is a normalization constant. Given a new input vector $\mathbf{x}_\star$, we can make predictions for its output $\mathbf{y}_\star$ using the predictive distribution given by

$$\begin{aligned} & p(\mathbf{y}_\star \mid \mathbf{x}_\star, \mathscr{D}) \\ = & \int p(\mathbf{y}_\star \mid \mathbf{x}_\star, \mathscr{W}, \mathbf{w}, \lambda, \mathbf{s}, \sigma^2) p(\mathscr{W}, \mathbf{w}, \lambda, \mathbf{s}, \sigma^2 \mid \mathscr{D}) \\ & d\sigma^2 \, d\lambda \, d\mathbf{s} \, d\mathscr{W} \, d\mathbf{w}, \end{aligned} \tag{6.13}$$

where $p(\mathbf{y}_\star \mid \mathbf{x}_\star, \mathscr{W}, \mathbf{w}, \lambda, \mathbf{s}, \sigma^2) = \mathscr{N}(\mathbf{y}_\star \mid \text{Net}(\mathbf{x}_\star), \sigma^2)$. However, the exact computation of $p(\mathscr{W}, \mathbf{w}, \lambda, \mathbf{s}, \sigma^2 \mid$ and $p(\mathbf{y}_\star \mid \mathbf{x}_\star)$ is not tractable in most cases. Therefore, in practice, we have to resort to approximate inference methods.

It should be noted that λ is the same for all network parameters. However, it can be different for $\mathscr{W}$ or constructed to represent the structural sparsity for Convolutional kernels in NN aiming for Network Compression, which is related to Bayesian compression [23] and structural sparsity compression [35]. Since our main focus is on architecture parameters, without breaking the flow, we will fix λ which is equivalent to the weight decay coefficient in SGD and $\sigma^2 = 0.01$ that is equivalent to the regularization coefficient for network parameters.

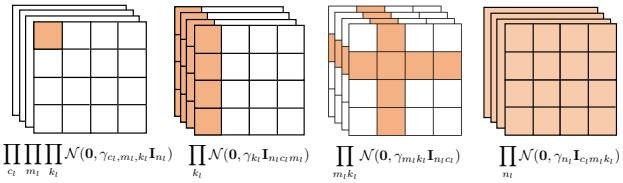


Figure 6.2: Structural Sparsity

In case of uniform hyperpriors, we only need to maximize the term $p(\mathbf{Y}|\lambda,\mathbf{s},\sigma^2)$ [3, 24]

$$\int\int p(\mathbf{Y}|\mathscr{W},\mathbf{w},\mathbf{X},\sigma^2)p(\mathscr{W}|\lambda)p(\mathbf{w}|\mathbf{s})d\mathscr{W}d\mathbf{w}. \tag{6.14}$$

We assume that the distribution of data likelihood belongs to the exponential family

$$p(\mathbf{Y}|\mathscr{W},\mathbf{w},\mathbf{X},\sigma^2) \sim \exp\Big(-E_D(\mathbf{Y};\mathrm{Net}(\mathbf{X};\mathscr{W},\mathbf{w});\sigma^2)\Big) \tag{6.15}$$

where $E_D(*)$ is the *energy function* over data.

Hessian Calculation of Architecture Parameter After we have the computation method for the Hessian of a convolutional layer, we need to consider the Hessian of an architecture parameter. Now the output from node i to j under operation o becomes $w_{ij}^o B_i$, where w_{ij}^o is the architecture parameter and B_i stands for the input vector.

Inspired by [4], the Hessian for w_{ij}^o could be computed recursively as $\mathbf{H}_{ij}^o = \mathbb{E}(\sum(B_i)^2 H_j)$, where H_j is supposed to be the known pre-activation Hessian for B_j and H_i is the pre-activation Hessian for B_i.

$$H_i = \sum_{o\in\mathscr{O}} (w_{ij}^o)^2 H_j. \tag{6.16}$$

Since B_i and H_j are independent of each other, the Hessian $\mathbf{H}_{ij}^o$ could also be calculated more efficiently:

$$\mathbf{H}_{ij}^o = (\mathbb{E}(|B_i|)^2 H_j \tag{6.17}$$

where $\mathbb{E}$ will return the mean.

Algorithm As analyzed before, the optimization objective of searching architecture becomes removing redundant edges. The training algorithm is iteratively indexed by t. Each iteration may contain several epochs. The pseudo code is summarized in Algorithm 6. The cost function is simply the maximum likelihood over the data D with regularization whose intensity is controlled by the re-weighted coefficient ω

$$\mathscr{L}_D = E_D(\cdot) + \lambda_w \sum_{j<k}\sum_{o'\in\mathscr{O}} \|\omega_{jk}^{o'}(t) w_{jk}^{o'}\|_1 + \lambda\|\mathscr{W}\|_2^2 \tag{6.18}$$

Algorithm 6 BayesNAS Algorithm for Proxyless Tasks.

Input: $\boldsymbol{\gamma}(0), \boldsymbol{\omega}(0), \mathbf{w}(0) = \mathbf{1}$; $\lambda = 0.01$; sparsity intensity $\lambda_w^o \in \mathbb{R}^+$
Output:
for $t = 1$ to $T_{\max}$ **do**
1. Update $\mathbf{w}$ and $\mathscr{W}$ by minimizing $\mathscr{L}_D$ in Eq. (6.18)
2. Compute Hessian for $\mathbf{w}$ (Eq. (6.16), Eq. (6.17))
3. Update variables associated with $\mathbf{w}$
while $i < j < k, o, o' \in \mathscr{O}$ **do**

$$C_{jk}^{o'}(t) = \left(\frac{1}{\gamma_{jk}^{o'}(t-1)} + \mathbf{H}_{jk}^{o'}(t) \right)^{-1} \tag{6.19}$$

$$\omega_{jk}^{o'}(t) = \frac{\sqrt{\gamma_{jk}^{o'}(t-1) - \mathbf{C}_{jk}^{o'}(t)}}{\gamma_{jk}^{o'}(t-1)} \tag{6.20}$$

$$s_{jk}^{o'}(t) = \left| \frac{w_{jk}^{o'}(t)}{\omega_{jk}^{o'}(t)} \right| \tag{6.21}$$

$$\gamma_{jk}^{o'}(t) \text{ is given by } Eq. (6.6) \text{ or } Eq. (6.8) \tag{6.22}$$

end while
4. Prune the architecture if the entropy $\frac{\ln\left(2\pi e \gamma_{jk}^{o'}\right)}{2} \leq 0$
5. Fix $\mathbf{w} = \mathbf{1}$, train the pruned net in the standard way
end for

The algorithm mainly includes five parts. The first part is to jointly train $\mathscr{W}$ and $\mathbf{w}$. The second part is to freeze the architecture parameters and prepare to compute their Hessian. The third part is to update the variables associated with the architecture parameters. The fourth part is to prune the architecture parameters and the pruned net will be trained in a standard way in the fifth part. As discussed previously on the drawback of magnitude-based pruning metrics, we propose a new metric based on the properties of differential entropy of the distribution. Since $p(w_{jk}^{o'})$ in Eq. (6.5) is Gaussian with zero mean $\gamma_{jk}^{o'}$ variance, the differential entropy is $\frac{1}{2}\ln\left(2\pi e \gamma_{jk}^{o'}\right)$. We set the threshold for $\gamma_{jk}^{o'}$ to prune related edges when $\frac{1}{2}\ln\left(2\pi e \gamma_{jk}^{o'}\right) \leq 0$, *i.e.*, $\gamma_{jk}^{o'} \leq 0.0585$.

The algorithm can be easily transferred to other scenarios. One scenario involves proxy tasks to find the cell. Suppose a network is assembled by stacking O different kinds of cells together, such as $\aleph_1$ normal cells and $\aleph_O$ reduction cells in [20]. Then optimal O cells are required to be designed in a NAS task. As explained before, we design a switch $\mathbf{s}$ for each architecture parameter w to determine the "on-off" of the corresponding edge in our method. In order to find such optimal cells, we propose that switches on the same position of the identical kind of cells should also be same. Based on this, the architecture parameters could be divided into different groups. The general grouped architecture

parameters are given as follows:

$$\mathbf{w}_{jk}^{o'}(t) = \Big[\underbrace{\mathbf{w}_{jk,1}^{o'1}(t),\ldots,\mathbf{w}_{jk,1}^{o'\aleph_1}(t)}_{\aleph_1 \textbf{ elements}} \Big| \ \ldots \ \Big| \underbrace{\mathbf{w}_{jk,O}^{o'1}(t),\ldots,\mathbf{w}_{jk,O}^{o'\aleph_O}(t)}_{\aleph_O \textbf{ elements}} \Big]. \tag{6.23}$$

If the group o is consist of $\aleph_o$ elements, where $o = 1,\ldots,O$, the optimal $s_{jk,o}^{o'}$ can be obtained as:

$$\sum_{i=1}^{\aleph_o} \left(\frac{\mathbf{w}_{jk,o}^{o'}{}^{\top}(t)\mathbf{w}_{jk,o}^{o'}(t)}{s_{jk,o}^{o'}(t)} + \omega_{jk,o}^{o'i}(t)^2 s_{jk,o}^{o'}(t) \right) \geq 2 \left\| \sqrt{\sum_{i=1}^{\aleph_o} \omega_{jk,o}^{o'i}(t)^2} \cdot \mathbf{w}_{jk,o}^{o'}(t) \right\|_{\ell_2}, \tag{6.24}$$

then

$$s_{jk,o}^{o'}(t) = \frac{\left\| \mathbf{w}_{jk,o}^{o'}(t) \right\|_{\ell_2}}{\sqrt{\sum_{i=1}^{\aleph_o} \omega_{jk,o}^{o'i}(t)^2}}, \forall i. \tag{6.25}$$

The calculation of $\omega_{jk,g}^{o'}$ for group o is:

$$\omega_{jk,o}^{o'}(t) = \sqrt{\sum_{i=1}^{\aleph_o} \frac{\sqrt{\gamma_{jk,o}^{o'i}(t-1) - \mathrm{C}_{jk,o}^{o'i}(t)}}{\gamma_{jk,o}^{o'i}(t-1)^2}} \tag{6.26}$$

and both $\mathbf{s}$ and ω for the different elements in the identity group should keep the same:

$$\mathbf{s}_{jk}^{o'}(t) = \Big[\underbrace{\mathbf{s}_{jk,1}^{o'}(t),\ldots,\mathbf{s}_{jk,1}^{o'}(t)}_{\aleph_1 \textbf{ elements}} \Big| \ \ldots \ \Big| \underbrace{\mathbf{s}_{jk,O}^{o'}(t),\ldots,\mathbf{s}_{jk,O}^{o'}(t)}_{\aleph_O \textbf{ elements}} \Big] \tag{6.27}$$

$$\omega_{jk}^{o'}(t) = \Big[\underbrace{\omega_{jk,1}^{o'}(t),\ldots,\omega_{jk,1}^{o'}(t)}_{\aleph_1 \textbf{ elements}} \Big| \ \ldots \ \Big| \underbrace{\omega_{jk,O}^{o'}(t),\ldots,\omega_{jk,O}^{o'}(t)}_{\aleph_O \textbf{ elements}} \Big] \tag{6.28}$$

Similar to Eq. (6.18), we group the same edge/operation in the repeated stacked cells, where g is the index. The cost function for proxy tasks is then given as follows in the form of re-weighted group Lasso:

$$\mathscr{L}_D = E_D(\cdot) + \lambda_w \sum_g \sum_{j<k} \sum_{o' \in \mathscr{O}} \|\omega_{jk,g}^{o'}(t) w_{jk,g}^{o'}\|_2 + \lambda \|\mathscr{W}\|_2^2 \tag{6.29}$$

The pseudo code is summarised in Algorithm 7.

6.4. EXPERIMENT

6.4.1. NEURAL ARCHITECTURE SEARCH

The experiments focus on two scenarios in NAS: proxy NAS and proxyless NAS. For proxy NAS, we follow the pipeline in DARTS [20] and SNAS [37]. First BayesNAS is applied to

Algorithm 7 The proposed Algorithm is transferable for cell selection of proxy tasks.

Input: $\boldsymbol{\gamma}_{jk}^{o'}(0), \boldsymbol{\omega}_{jk}^{o'}(0), \mathbf{w}(0) = \mathbf{1}$; sparsity intensity $\lambda_w^o \in \mathbb{R}^+$; $\lambda = 0.01$; cost function $\mathscr{L}_D$ in Eq. (6.18)

Output:

for $t = 1$ to $T_{\max}$ **do**

1. Maximum likelihood with regularization:

$$\min_{\mathscr{W},\mathbf{w}} E_D(\cdot) + \lambda_w \sum_g \sum_{j<k} \sum_{o' \in \mathscr{O}} \|\omega_{jk,g}^{o'}(t)\mathbf{w}_{jk,g}^{o'}\|_2 + \lambda \|\mathscr{W}\|_2^2 \tag{6.30}$$

2. Compute Hessian for $\mathbf{w}$ (Eq. (6.17))

3. Update variables associated with $\mathbf{w}$

while $g \in (1, O); i < j < k; o, o' \in \mathscr{O}$ **do**

$$C_{jk}^{o'}(t) = \left(\frac{1}{\gamma_{jk}^{o'}(t-1)} + \mathbf{H}_{jk}^{o'}(t) \right)^{-1} \tag{6.31}$$

$$\omega_{jk}^{o'}(t) \text{ is given by (6.26) and (6.28)} \tag{6.32}$$

$$\mathbf{s}_{jk}^{o'}(t) \text{ is given by (6.25) and (6.27)} \tag{6.33}$$

$$\gamma_{jk}^{o'}(t) \text{ is given by (6.6) or (6.8)with}: \gamma_{jk}^{o'}(t) = \Big[\underbrace{\gamma_{jk,1}^{o'}(t), \ldots}_{\aleph_1 \textbf{ elements}} \Big| \ \ldots \ \Big| \underbrace{\gamma_{jk,O}^{o'}(t), \ldots}_{\aleph_O \textbf{ elements}} \Big] \tag{6.34}$$

end while

4. Prune the architecture if the entropy $\frac{\ln\left(2\pi e \gamma_{jk}^{o'}\right)}{2} \leq 0$

5. Fix $\mathbf{w} = \mathbf{1}$, train the pruned net in the standard way

end for

search for the best convolutional cells in a complete network on CIFAR-10. Then a network constructed by stacking the learned cells is retrained for performance comparison. For proxyless NAS, we follow the pipeline in ProxylessNAS [6]. First, the tree-like cell from [7] with multiple paths is integrated into the PyramidNet [12]. Then we search for the optimal path(s) within each cell by BayesNAS. Finally, the network is reconstructed by retaining only the optimal path(s) and retrained on CIFAR-10 for performance comparison.

ARCHITECTURE EVALUATION ON CIFAR-10

Proxy Search Unlike DARTS and SNAS that rely on validation accuracy during or after search, we use γ in BayesNAS as a performance evaluation criterion which enables us to achieve it in an one-shot manner. Our setup follows DARTS and SNAS, where convolutional cells of 7 nodes are stacked for multiple times to form a network. The input nodes, *i.e.*, the first and second nodes, of cell k are set equal to the outputs of cell $k-1$ and cell $k-2$ respectively, with 1×1 convolutions inserted as necessary, and the output node is the depthwise concatenation of all the intermediate nodes. Reduction cells are located at 1/3

Table 6.1: Classification errors of BayesNAS and state-of-the-art image classifiers on CIFAR-10.

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	Search Method
DenseNet-BC [16]	3.46	25.6	-	manual
NASNet-A + cutout [44]	2.65	3.3	1800	RL
AmoebaNet-B + cutout [31]	2.55 ± 0.05	2.8	3150	evolution
Hierarchical Evo [21]	3.75 ± 0.12	15.7	300	evolution
PNAS [19]	3.41 ± 0.09	3.2	225	SMBO
ENAS + cutout [29]	2.89	4.6	0.5	RL
Random search baseline + cutout [20]	3.29 ± 0.15	3.2	1	random
DARTS (2nd order bi-level) + cutout [20]	2.76 ± 0.09	3.4	1	gradient
SNAS (single-level) + moderate con + cutout [37]	2.85 ± 0.02	2.8	1.5	gradient
DSO-NAS-share+cutout [40]	2.84 ± 0.07	3.0	1	gradient
Proxyless-G + cutout [6]	2.08	5.7	-	gradient
BayesNAS + cutout + $\lambda_w^o = 0.01$	3.02±0.04	2.59±0.23	0.2	gradient
BayesNAS + cutout + $\lambda_w^o = 0.007$	2.90±0.05	3.10±0.15	0.2	gradient
BayesNAS + cutout + $\lambda_w^o = 0.005$	2.81±0.04	3.40±0.62	0.2	gradient
BayesNAS + TreeCell-A + Pyrimaid backbone + cutout	2.41	3.4	0.1	gradient

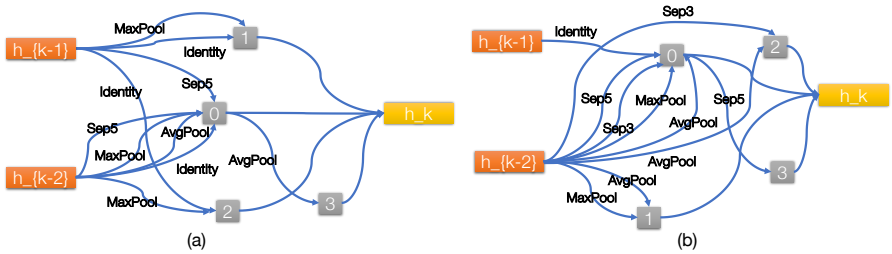


Figure 6.3: Normal and reduction cell found by BayesNAS with $\lambda_w^o = 0.01$.

and 2/3 of the total depth of the network to reduce the spatial resolution of feature maps. Unlike DARTS and SNAS, we exclude *zero* operations. In the searching stage, we train a small network stacked by 8 cells using BayesNAS with different λ_w. This network size is determined to fit into a single GPU. Since we cache the feature maps in memory, we can only set batch size as 18. The optimizer we use is SGD optimizer with momentum 0.9 and fixed learning rate 0.1. Other training setups follow DARTS and SNAS. The search takes about 3 hours on a single GPU[1]. The normal and reduction cells learned on CIFAR-10 using BayesNAS are shown in Figure 6.3a and 6.3b. A large network of 20 cells where cells at 1/3 and 2/3 are reduction cells is trained from scratch with the batch size of 128. The validation accuracy is presented in Table 6.1. The test error rate of BayesNAS is competitive against state-of-the-art techniques and BayesNAS is able to find convolutional cells with fewer parameters when compared to DARTS and SNAS.

[1]All the experiments were performed using NVIDIA TITAN V GPUs

Table 6.2: Comparison with state-of-the-art image classifiers on ImageNet in the mobile setting.

Architecture	Test Error (%)		Params	Search Cost	Search Method
	top-1	top-5	(M)	(GPU days)	
Inception-v1 [33]	30.2	10.1	6.6	–	manual
MobileNet [15]	29.4	10.5	4.2	–	manual
ShuffleNet 2× (v1) [39]	29.1	10.2	~5	–	manual
ShuffleNet 2× (v2) [39]	26.3	–	~5	–	manual
NASNet-A [44]	26.0	8.4	5.3	1800	RL
NASNet-B [44]	27.2	8.7	5.3	1800	RL
NASNet-C [44]	27.5	9.0	4.9	1800	RL
AmoebaNet-A [31]	25.5	8.0	5.1	3150	evolution
AmoebaNet-B [31]	26.0	8.5	5.3	3150	evolution
AmoebaNet-C [31]	24.3	7.6	6.4	3150	evolution
PNAS [19]	25.8	8.1	5.1	~225	SMBO
DARTS [20]	26.9	9.0	4.9	4	gradient
BayesNAS ($\lambda_w^o = 0.01$)	28.1	9.4	4.0	0.2	gradient
BayesNAS ($\lambda_w^o = 0.007$)	27.3	8.4	3.3	0.2	gradient
BayesNAS ($\lambda_w^o = 0.005$)	26.5	8.9	3.9	0.2	gradient

Proxyless Search Using an existing tree-like cell, we apply BayesNAS to search for the optimal path(s) within each cell. Varying from proxy search, cells do not share architecture in proxyless search. The backbone used is PyramidNet with three layers, each consisting of 18 bottleneck blocks and $\alpha = 84$. All 3×3 convolution in bottleneck blocks are replaced by the tree-cell that has in total 9 possible paths within. The groups for grouped convolution are set to 2. For the detailed structure of the tree-cell, we refer to [7]. In the searching stage, we set the batch size to 32 and the learning rate to 0.1. We use the same optimizer as for proxy search. The λ of BayesNAS for each possible path is set to 1×10^{-2}. Because each cell can have a different structure in the proxyless setting, we demonstrate only two typical types of cell structure among all of them in Figure 6.4a and Figure 6.4b. The first type is a chain-like structure where only one path exists in the cell connecting the input of the cell to its output. The second type is an inception structure where divergence and convergence both exist in the cell. Our further observation reveals that some cells are dispensable with respect to the entire network. After the architecture is determined, the network is trained from scratch with the batch size of 64, learning rate as 0.1, and cosine annealing learning rate decay schedule [22]. The validation accuracy is also presented in Table 6.1. Although the test error increases slightly compared to [6], there is a significant drop in the number of model parameters to be learned, which is beneficial for both training and inference.

TRANSFERABILITY TO IMAGENET

For ImageNet mobile setting, the input images are of size 224×224. A network of 14 cells is trained for 250 epochs with batch size 128, weight decay 3×10^{-5} and initial SGD learning rate 0.1 (decayed by a factor of 0.97 after each epoch). Results in Table 6.2 show that

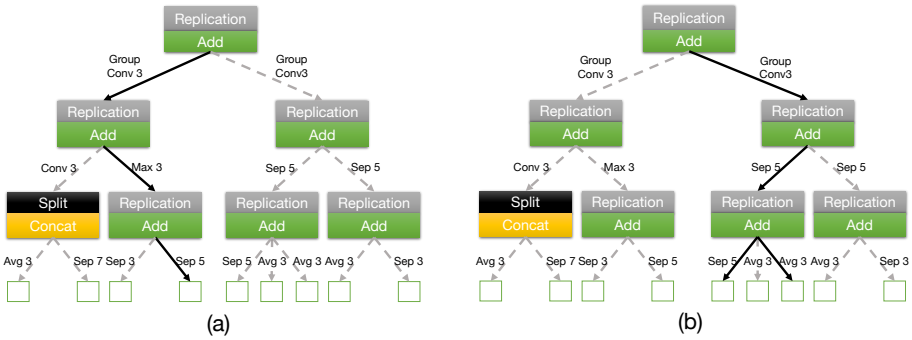


Figure 6.4: The pruned tree-cell: (a) The chain-like where only one path exists in the cell connecting the input of the cell to its output. (b) The inception structure where divergence and convergence both exist in the cell. The solid directed lines denote the path found by BayesNAS while the dashed ones denote the paths discarded.

the cell learned on CIFAR-10 can be transfered to ImageNet and is capable of achieving competitive performance.

6.4.2. NEURAL NETWORK COMPRESSION

In addition to applying the proposed sparse Bayesian deep learning algorithm to address NAS problems, we also explore the possibility of our method on the network structural compression problem. In this section, we summarise the calculation details about applying various sparse structured priors over the network weights of a convolution neural network in Table 6.3. The regularization term and the calculation of hyper-parameters (e.g. ω, ψ) are included. Specifically, for the weight in the l-th convolutional layer $W^l \in \mathbb{R}^{N_l \times C_l \times m_l \times k_l}$, some examples of the structured sparsity are shown in Fig.6.5. The corresponding sparse prior is given in the second column of Table 6.3.

It should be noted that Table 6.3 is an extension of Table 2.1 which explains the calculation for a Fully-Connected neural network. And the prior for the weight in a FC layer can also be represented as this table with $m_l = k_l = 1$, N_l and C_l stand for the size of input feature and output features, respectively. The proposed network compression algorithm is generic for the weights in fully connected and convolutional neural networks. Similar to the pruning criterium, which has been illustrated in Section. 4.2.4 of Chapter. 4, the network pruning criteria also include both the magnitude of weight and the uncertainty of weight. The equation of pruning connection can be referred to Eq. (4.11).

LENET-300-100 AND LENET-5 ON MNIST
RESNET-18 ON CIFAR-10

We also evaluate our algorithm on the Cifar10 dataset using ResNet-18 as the initialized backbone [13]. In addition to the input Conv layer and output FC layer, the other 16 Conv layers are separated into 8 blocks with 2 layers each block. We apply shape-wise and filter-wise regularization to the Conv layer as shown in Fig. 6.5(a) and 6.5(j); row-wise and column-wise regularization to the FC layer as shown in Fig. 6.5(b) and 6.5(c). The re-

Table 6.3: Hyper-parameter update rule for CNN

Category	Prior Formulation	$R(\omega^l, W^l)$	ω^l	ψ^l
(a) Shape-wise	$\prod_{c_l}\prod_{m_l}\prod_{k_l}\mathcal{N}(\mathbf{0}, \psi_{c_l,m_l,k_l}\mathbf{I}_{n_l})$	$\sum_{c_l=1}^{C_l}\sum_{m_l=1}^{M_l}\sum_{k_l=1}^{K_l}\|\omega^l_{:,c_l,m_l,k_l} \circ \boldsymbol{W}^l_{:,c_l,m_l,k_l}\|_2$	$\omega^l_o = \sqrt{\sum_{c_l}\sum_{m_l}\sum_{k_l}\|\alpha^l_{:,c_l,m_l,k_l}\|}$ $\omega^l_{:,c_l,m_l,k_l} = \omega^l_o \cdot \mathbf{I}^l_{:,c_l,m_l,k_l}$	$\psi^l_o = \frac{\|\boldsymbol{W}^l_{:,c_l,m_l,k_l}\|_2}{\omega^l_{:,c_l,m_l,k_l}(t-1)}$ $\psi^l_{:,c_l,m_l,k_l} = \psi^l_o \cdot \mathbf{I}^l_{:,c_l,m_l,k_l}$
(b) Row-wise	$\prod_{c_l}\prod_{m_l}\mathcal{N}(\mathbf{0}, \psi_{c_l,m_l}\mathbf{I}_{n_l k_l})$	$\sum_{c_l=1}^{C_l}\sum_{m_l=1}^{M_l}\|\omega_{:,c_l,m_l,:} \circ \boldsymbol{W}^l_{:,c_l,m_l,:}\|_2$	$\omega^l_o = \sqrt{\sum_{c_l}\sum_{m_l}\|\alpha^l_{:,c_l,m_l,:}\|}$ $\omega^l_{:,c_l,m_l,:} = \omega^l_o \cdot \mathbf{I}^l_{:,c_l,m_l,:}$	$\psi^l_o = \frac{\|\boldsymbol{W}^l_{:,c_l,m_l,:}\|_2}{\omega^l_{:,c_l,m_l,:}(t-1)}$ $\psi^l_{:,c_l,m_l,:} = \psi^l_o \cdot \mathbf{I}^l_{:,c_l,m_l,:}$
(c) Column-wise	$\prod_{c_l}\prod_{k_l}\mathcal{N}(\mathbf{0}, \psi_{c_l,k_l}\mathbf{I}_{n_l m_l})$	$\sum_{c_l=1}^{C_l}\sum_{k_l=1}^{K_l}\|\omega^l_{:,c_l,:,k_l} \circ \boldsymbol{W}^l_{:,c_l,:,k_l}\|_2$	$\omega^l_o = \sqrt{\sum_{c_l}\sum_{k_l}\|\alpha^l_{:,c_l,:,k_l}\|}$ $\omega^l_{:,c_l,:,k_l} = \omega^l_o \cdot \mathbf{I}^l_{:,c_l,:,k_l}$	$\psi^l_o = \frac{\|\boldsymbol{W}^l_{:,c_l,:,k_l}\|_2}{\omega^l_{:,c_l,:,k_l}(t-1)}$ $\psi^l_{:,c_l,:,k_l} = \psi^l_o \cdot \mathbf{I}^l_{:,c_l,:,k_l}$
(d) Row & column-wise	$\prod_{c_l}\prod_{m_l k_l}\mathcal{N}(\mathbf{0}, \psi_{c_l,m_l k_l}\mathbf{I}_{n_l})$	$\bar{W}^l = [\boldsymbol{W}^l_{:,c_l,m_l,:}, \boldsymbol{W}^l_{:,c_l,:,k_l}]$ $\tilde{\omega}^l = [\omega^l_{:,c_l,m_l,:}, \omega^l_{:,c_l,:,k_l}]$ $\sum_{c_l=1}^{C_l}\sum\|\tilde{\omega}^l \circ \bar{W}^l\|_2$	$\omega^l_o = \sqrt{\sum_{c_l}\sum_{m_l}\|\alpha^l_{:,c_l,m_l,:}\| + \sum_{c_l}\sum_{k_l}\|\alpha^l_{:,c_l,:,k_l}\|}$ $\omega^l_{:,c_l,m_l,:} = \omega^l_o \cdot \mathbf{I}^l_{:,c_l,m_l,:}$ $\omega^l_{:,c_l,:,k_l} = \omega^l_o \cdot \mathbf{I}^l_{:,c_l,:,k_l}$	$\psi^l_o = \frac{\|\bar{W}^l\|_2}{\tilde{\omega}^l(t-1)}$ $\psi^l_{:,c_l,m_l,:} = \psi^l_o \cdot \mathbf{I}^l_{:,c_l,m_l,:}$ $\psi^l_{:,c_l,:,k_l} = \psi^l_o \cdot \mathbf{I}^l_{:,c_l,:,k_l}$
(e) Channel-wise	$\prod_{c_l}\mathcal{N}(\mathbf{0}, \psi_{c_l}\mathbf{I}_{n_l m_l k_l})$	$\sum_{c_l=1}^{C_l}\|\omega^l_{:,c_l,:,:} \circ \boldsymbol{W}^l_{:,c_l,:,:}\|_2$	$\omega^l_o = \sqrt{\sum_{c_l}\|\alpha^l_{:,c_l,:,:}\|}$ $\omega^l_{:,c_l,:,:} = \omega^l_o \cdot \mathbf{I}^l_{:,c_l,:,:}$	$\psi^l_o = \frac{\|\boldsymbol{W}^l_{:,c_l,:,:}\|_2}{\omega^l_{:,c_l,:,:}(t-1)}$ $\psi^l_{:,c_l,:,:} = \psi^l_o \cdot \mathbf{I}^l_{:,c_l,:,:}$
(f) Group shape-wise	$\prod_{m_l}\prod_{k_l}\mathcal{N}(\mathbf{0}, \psi_{m_l,k_l}\mathbf{I}_{n_l c_l})$	$\sum_{m_l=1}^{M_l}\sum_{k_l=1}^{K_l}\|\omega^l_{:,:,m_l,k_l} \circ \boldsymbol{W}^l_{:,:,m_l,k_l}\|_2$	$\omega^l_o = \sqrt{\sum_{m_l}\sum_{k_l}\|\alpha^l_{:,:,m_l,k_l}\|}$ $\omega^l_{:,:,m_l,k_l} = \omega^l_o \cdot \mathbf{I}^l_{:,:,m_l,k_l}$	$\psi^l_o = \frac{\|\boldsymbol{W}^l_{:,:,m_l,k_l}\|_2}{\omega^l_{:,:,m_l,k_l}(t-1)}$ $\psi^l_{:,:,m_l,k_l} = \psi^l_o \cdot \mathbf{I}^l_{:,:,m_l,k_l}$
(g) Group row-wise	$\prod_{m_l}\mathcal{N}(\mathbf{0}, \psi_{m_l}\mathbf{I}_{n_l c_l k_l})$	$\sum_{m_l=1}^{M_l}\|\omega^l_{:,:,m_l,:} \circ \boldsymbol{W}^l_{:,:,m_l,:}\|_2$	$\omega^l_o = \sqrt{\sum_{m_l}\|\alpha^l_{:,:,m_l,:}\|}$ $\omega^l_{:,:,m_l,:} = \omega^l_o \cdot \mathbf{I}^l_{:,:,m_l,:}$	$\psi^l_o = \frac{\|\boldsymbol{W}^l_{:,:,m_l,:}\|_2}{\omega^l_{:,:,m_l,:}(t-1)}$ $\psi^l_{:,:,m_l,:} = \psi^l_o \cdot \mathbf{I}^l_{:,:,m_l,:}$
(h) Group column-wise	$\prod_{k_l}\mathcal{N}(\mathbf{0}, \psi_{k_l}\mathbf{I}_{n_l c_l m_l})$	$\sum_{k_l=1}^{K_l}\|\omega^l_{:,:,:,k_l} \circ \boldsymbol{W}^l_{:,:,:,k_l}\|_2$	$\omega^l_o = \sqrt{\sum_{k_l}\|\alpha^l_{:,:,:,k_l}\|}$ $\omega^l_{:,:,:,k_l} = \omega^l_o \cdot \mathbf{I}^l_{:,:,:,k_l}$	$\psi^l_o = \frac{\|\boldsymbol{W}^l_{:,:,:,k_l}\|_2}{\omega^l_{:,:,:,k_l}(t-1)}$ $\psi^l_{:,:,:,k_l} = \psi^l_o \cdot \mathbf{I}^l_{:,:,:,k_l}$
(i) Group row & column-wise	$\prod_{m_l k_l}\mathcal{N}(\mathbf{0}, \psi_{m_l k_l}\mathbf{I}_{n_l c_l})$	$\bar{W}^l = [\boldsymbol{W}^l_{:,:,m_l,:}, \boldsymbol{W}^l_{:,:,:,k_l}]$ $\tilde{\omega}^l = [\omega^l_{:,:,m_l,:}, \omega^l_{:,:,:,k_l}]$ $\sum\|\tilde{\omega}^l \circ \bar{W}^l\|_2$	$\omega^l_o = \sqrt{\sum_{m_l}\|\alpha^l_{:,:,m_l,:}\| + \sum_{k_l}\|\alpha^l_{:,:,:,k_l}\|}$ $\omega^l_{:,:,m_l,:} = \omega^l_o \cdot \mathbf{I}^l_{:,:,m_l,:}$ $\omega^l_{:,:,:,k_l} = \omega^l_o \cdot \mathbf{I}^l_{:,:,:,k_l}$	$\psi^l_o = \frac{\|\bar{W}^l\|_2}{\tilde{\omega}^l(t-1)}$ $\psi^l_{:,:,m_l,:} = \psi^l_o \cdot \mathbf{I}^l_{:,:,m_l,:}$ $\psi^l_{:,:,:,k_l} = \psi^l_o \cdot \mathbf{I}^l_{:,:,:,k_l}$
(j) Filter-wise	$\prod_{n_l}\mathcal{N}(\mathbf{0}, \psi_{n_l}\mathbf{I}_{c_l m_l k_l})$	$\sum_{n_l=1}^{N_l}\|\omega^l_{n_l,:,:,:} \circ \boldsymbol{W}^l_{n_l,:,:,:}\|_2$	$\omega^l_o = \sqrt{\sum_{n_l}\|\alpha^l_{n_l,:,:,:}\|}$ $\omega^l_{n_l,:,:,:} = \omega^l_o \cdot \mathbf{I}^l_{n_l,:,:,:}$	$\psi^l_o = \frac{\|\boldsymbol{W}^l_{n_l,:,:,:}\|_2}{\omega^l_{n_l,:,:,:}(t-1)}$ $\psi^l_{n_l,:,:,:} = \psi^l_o \cdot \mathbf{I}^l_{n_l,:,:,:}$

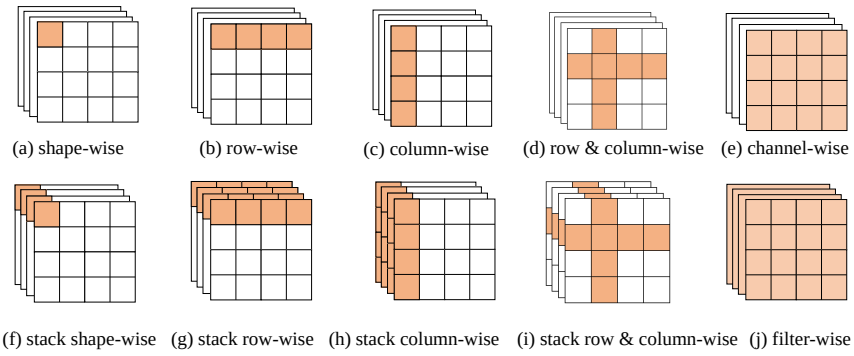


Figure 6.5: some examples of structured sparsity for the 3D filters in Conv layer with extensions of [35]. Coloured squares mean the weights to be pruned. It should be noted that the FC layer can be easily enforced by (a)-(e).

sult is given in Table 6.4. It can be found that the two Conv layers in block 4 are pruned away, which shows the potential of our method to reduce the number of layers.

Table 6.4: Model Sparsity of ResNet-18 on Cifar10 dataset

conv1	conv2-5	conv6-9	conv10-13	con14-17	FC layer	Total	Test error
22.80%	10.06%	9.24%	20.64%	1.43%	10.33%	2.96%	6.58% (baseline)
	22.87%	5.45%	15.04%	0.19%			
	20.05%	4.94%	10.77%	0			6.23% (our method)
	12.99%	10.73%	4.61%	0			

RESNET-33 ON ATRIAL FIBRILLATION DATASET

Atrial fibrillation (AF) is one typical symptom of cardiac arrhythmia. The pre-diagnosing of AF based on electrocardiography (ECG) data has been a hot research topic in recent years. In the PhysioNet and Computing in Cardiology Challenge 2017 [8], the work [14] used a ResNet to extract deep features from ECG data and achieved an impressive result. However, despite its accurate predicted performance, the complexity of the model also makes it run slow. In this section, we try to apply network compression techniques to help the model provided in [14] being more lightweight and more efficient. The network compression experiments are performed on the PhysioNet/Computing in Cardiology Challenge 2017 dataset [8], containing 8528 records of short single-lead 300 Hz ECG data. The data are labelled with four classes: (1) normal sinus rhythm (N for short, 5076 records), (2) AF (A for short, 758 records), (3) alternative rhythm (O for short, 2415 records), and (4) noisy recordings (P for short, 279 records).

We apply channel-wise regularization to the Conv layer as shown in Fig. 6.5(e); row-wise and column-wise regularization to the FC layer as shown in Fig. 6.5(b) and 6.5(c). The result is given in Table 6.5. The structural sparsity for each layer is in Table 6.6. During the training process, the dynamic gamma pruning with threshold 0.001 is used. From

Table 6.5, it can be found that 24 Conv layers are pruned away. And the overall sparsity is only 1.03%. The model size reduced from 167.4M to 1.4M, which is very promising to be implemented on resource-constrained mobile devices (e.g. smartphones and handheld devices).

Table 6.5: Sparsity for each layer in ResNet-33 on Atrial fibrillation dataset

conv1	conv2-9	conv10-17	conv18-25	26-33	FC layer	Total	F1 Score	Model size
9.37%	4.25% 22.85% 0 0 0 0 0 0	23.5% 19.13% 0 0 0 0 0 0	16.04% 9.11% 0 0 0 0 0 0	6.53% 0.35% 0 0 0 0 0 0	2.93%	1.03%	0.74 (baseline) 0.75 (our method)	167.4M (baseline) 1.4M (our method)

The structural sparsity for each layer is in Table 6.6.

Table 6.6: Structural sparsity for each layer in ResNet-33 on Atrial fibrillation dataset

Channel category	conv1	conv2-9	conv10-17	conv18-25	26-33	FC layer
In channel	100%	9.35% 40.63% 0 0 0 0 0 0	54.69% 42.97% 0 0 0 0 0 0	56.25% 28.13% 0 0 0 0 0 0	50.78% 11.91% 0 0 0 0 0 0	100%
Out channel	9.37%	45.31% 56.25% 0 0 0 0 0 0	42.97% 44.53% 0 0 0 0 0 0	28.52% 32.42% 0 0 0 0 0 0	12.89% 2.93% 0 0 0 0 0 0	2.93%

6.5. CONCLUSION

We introduce BayesNAS that can directly learn a sparse neural network architecture based on high-dimensional data. We significantly reduce the search time by using only one epoch to get the candidate architecture. The proposed method can also be successfully applied in the neural network compression tasks.

BIBLIOGRAPHY

[1] Bowen Baker et al. “Designing Neural Network Architectures using Reinforcement Learning”. In: *International Conference on Learning Representations* (2017).

[2] Gabriel Bender et al. “Understanding and simplifying one-shot architecture search”. In: *International Conference on Machine Learning*. 2018, pp. 549–558.

[3] James O Berger. *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013.

[4] Aleksandar Botev, Hippolyt Ritter, and David Barber. “Practical Gauss-Newton Optimisation for Deep Learning”. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML’17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 557–565.

[5] Andrew Brock et al. “SMASH: One-Shot Model Architecture Search through HyperNetworks”. In: *International Conference on Learning Representations*. 2018.

[6] Han Cai, Ligeng Zhu, and Song Han. “ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware”. In: *International Conference on Learning Representations*. 2019. URL: https://arxiv.org/pdf/1812.00332.pdf.

[7] Han Cai et al. “Path-Level Network Transformation for Efficient Architecture Search”. In: *ICML*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 677–686.

[8] G. D. Clifford et al. “AF classification from a short single lead ECG recording: The PhysioNet/computing in cardiology challenge 2017”. In: *2017 Computing in Cardiology (CinC)*. Sept. 2017, pp. 1–4. DOI: 10.22489/CinC.2017.065-469.

[9] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. “Efficient Multi-Objective Neural Architecture Search via Lamarckian Evolution”. In: *International Conference on Learning Representations*. 2019.

[10] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. “Neural Architecture Search: A Survey”. In: *Journal of Machine Learning Research* 20.55 (2019), pp. 1–21.

[11] Ariel Gordon et al. “Morphnet: Fast & simple resource-constrained structure learning of deep networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1586–1595.

[12] Dongyoon Han, Jiwhan Kim, and Junmo Kim. “Deep pyramidal residual networks”. In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, pp. 6307–6315.

[13] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[14] Shenda Hong et al. “Combining deep neural networks and engineered features for cardiac arrhythmia detection from ECG recordings”. In: *Physiological measurement* 40.5 (2019), p. 054009.

[15] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV].

[16] Gao Huang et al. “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[17] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. “Snip: Single-shot network pruning based on connection sensitivity”. In: *arXiv preprint arXiv:1810.02340* (2018).

[18] Chenxi Liu et al. *Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation*. 2019. arXiv: 1901.02985 [cs.CV].

[19] Chenxi Liu et al. “Progressive neural architecture search”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 19–34.

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. “DARTS: Differentiable Architecture Search”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=S1eYHoC5FX.

[21] Hanxiao Liu et al. “Hierarchical Representations for Efficient Architecture Search”. In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=BJQRKzbA-.

[22] Ilya Loshchilov and Frank Hutter. “SGDR: Stochastic Gradient Descent with Warm Restarts”. In: *International Conference on Learning Representations*. 2017.

[23] Christos Louizos, Karen Ullrich, and Max Welling. “Bayesian compression for deep learning”. In: *arXiv preprint arXiv:1705.08665* (2017).

[24] David JC MacKay. “Bayesian interpolation”. In: *Neural computation* 4.3 (1992), pp. 415–447.

[25] David JC MacKay. “Bayesian methods for backpropagation networks”. In: *Models of neural networks III*. Springer, 1996, pp. 211–254.

[26] Risto Miikkulainen et al. “Evolving deep neural networks”. In: *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 293–312.

[27] Radford M Neal. “Bayesian learning for neural networks”. In: (1995).

[28] Wei Pan. “Bayesian Learning for Nonlinear System Identification”. PhD dissertation. Imperial College London, 2017.

[29] Hieu Quang Pham et al. “Efficient Neural Architecture Search via Parameter Sharing”. In: *ICML*. 2018.

[30] Esteban Real et al. “Large-scale evolution of image classifiers”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 2902–2911.

[31] Esteban Real et al. “Regularized Evolution for Image Classifier Architecture Search”. In: *AAAI*. 2019.

[32] Shreyas Saxena and Jakob Verbeek. “Convolutional neural fabrics”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 4053–4061.

[33] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[34] Michael E Tipping. “Sparse Bayesian learning and the relevance vector machine”. In: *Journal of machine learning research* 1.Jun (2001), pp. 211–244.

[35] Wei Wen et al. “Learning structured sparsity in deep neural networks”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 2074–2082.

[36] Lingxi Xie and Alan Yuille. “Genetic cnn”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE. 2017, pp. 1388–1397.

[37] Sirui Xie et al. “SNAS: stochastic neural architecture search”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=rylqooRqK7.

[38] Chris Zhang, Mengye Ren, and Raquel Urtasun. “Graph HyperNetworks for Neural Architecture Search”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=rkgW0oA9FX.

[39] Xiangyu Zhang et al. “Shufflenet: An extremely efficient convolutional neural network for mobile devices”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 6848–6856.

[40] Xinbang Zhang, Zehao Huang, and Naiyan Wang. *Single Shot Neural Architecture Search Via Direct Sparse Optimization*. 2019. URL: https://openreview.net/forum?id=ryxjH3R5KQ.

[41] Zhao Zhong et al. “Practical Block-wise Neural Network Architecture Generation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2423–2432.

[42] Hongpeng Zhou et al. “Bayesnas: A bayesian approach for neural architecture search”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 7603–7613.

[43] Barret Zoph and Quoc V. Le. “Neural Architecture Search with Reinforcement Learning”. In: *International Conference on Learning Representations*. 2017.

[44] Barret Zoph et al. “Learning Transferable Architectures for Scalable Image Recognition”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 8697–8710.

7

CONCLUSION AND FUTURE WORK

This thesis investigates the applications of the developed sparse Bayesian deep learning algorithms with different priors on DNN and DNN-like models. This chapter concludes the thesis in Section 7.1 and outlines some possible avenues for future research in Section 7.2.

7.1. CONCLUSION

To address the main research challenges about system identification (SYSID) using deep neural networks (DNNs) as illustrated in Section **??**, this thesis proposes several sparse Bayesian deep learning algorithms which can be implemented in diverse SYSID applications. The scientific and technical implications of the research findings can be summarised as follows.

Chapter 1 illustrates the research background, research motivation, and research challenges of using DNNs for SYSID. It highlights two main challenges, i.e., the overfitting issue caused by model complexity and the infeasibility for white-box modelling. By regarding these research challenges as sparse regression problems, the Bayesian approach becomes a potentially more effective solution than the conventional sparse regression techniques, such as lasso and group lasso. Chapter 2 presents three sparse Bayesian deep learning (SBDL) algorithms that enforce different priors on model parameters, i.e., single prior, group prior, and fused prior. This thesis adopts the Laplace approximated method to approximate the posterior distribution since it can scale better to DNNs than other approximate inference methods, e.g., variational inference. However, the inverse Hessian is a necessity for the Laplace approximated method. The Hessian calculation is intractable for DNNs, especially for the indirect convolutional and recurrent operation. To address this challenge, we develop efficient Hessian calculation methods by extracting the diagonal values of Hessian. The proposed Hessian calculation methods can accelerate the optimization process and turn the intractable training problem into a tractable one. The detailed procedures for the Hessian calculation are presented in Section 2.2 of Chapter 2.

In Chapter 3, the SBDL algorithm with the single prior is implemented to identify a repressilator model. We designed a combined neural network that includes both a linear and nonlinear sub-network. The linear sub-network is a fully connected (FC) neural network without hidden layers. The nonlinear sub-network is also a FC neural network with its activation function replaced by the typical form of Hill function. Finally, the structure and parameters of the mathematical expressions of the repressilator model, especially the topology and coefficients of the Hill function, can be identified precisely.

To address several typical problems in SYSID, including input feature selection, easily overfitting to the training dataset, we apply the SBDL algorithm with the group prior in Chapter 4. In this way, the input features of DNN models (i.e., MLP and LSTM) can be reduced clearly. Moreover, structural sparsity can be achieved to alleviate the overfitting issue. Besides, the parameter and prediction uncertainty is also derived with the Monte-Carlo sampling method. The proposed method can achieve good and competitive simulation accuracy on several linear and nonlinear system identification benchmarks.

To realize the white-box modelling which can discover the governing equations from data, we design a DNN-like hierarchical network, termed as Mathematical Operation Network (MathONet) in Chapter 5. The MathONet consists of unary and binary mathematical operations, which can formulate closed-form mathematical expressions. An initialized MathONet can be regarded as an over-redundant graph, whose sub-graph is the true underlying solution. To extract the essential sub-graph, this chapter uses the SBDL algorithm with the fused prior (single prior & group prior) on model parameters. The single prior is used to select the mathematical operation, and the group prior is used to select the input features. The proposed approach can identify ordinary differential equations (ODEs) or

partial differential equations (PDEs) for several dynamic systems. We also extend the proposed SBDL algorithms in two deep learning topics with high-dimensional datasets, i.e., neural architecture search (NAS) and neural network compression in Chapter 6.

7.2. FUTURE WORK

7.2.1. MODEL TYPE SELECTION

In SYSID, the model type selection is an important research aspect, which still lacks a specific definition to the best of our knowledge. The most relevant concept is the "model set selection", which can be found in [4, 9]. In these works, the system model is normally regarded as a transfer function $y(t) = G(q)u(t) + H(q)e(t)$. And the model set selection means the choice of $[G(q), H(q)]$, which decides the model structure and characterizes the model as ARX, OE, ARMAX and FIR, etc. Another relevant concept is the "type of models" [9], which distinguishes models between four types, i.e., mental, software, graphical, and mathematical models. Obviously, these two primitive terminologies describe the model with two polarizations. The first one confines a model in the range described by transfer functions, deprived of discussion with other typical models such as neural networks. On the contrary, the second one covers a wide range of model categories that can even be applied beyond the engineering field (e.g., mental model).

In the future, we think it is necessary to discuss the "model" in the range between these two polarizations. First, with the prosperous development of machine/deep learning, some models, such as deep neural networks and symbolic trees, are indispensable and should be included in the discussion. Second, we limit our focus only to the mathematical models, which are the most prevalent model type for SYSID in the control and engineering field. Other types of models (e.g., mental models) are excluded from the consideration. To unify the nomenclature, the "model type selection" refers to building a mathematical model which can match the measurements by training a model selected from or combined by four typical model types, i.e., neural network model, basis function model, linear model and symbolic regression model. The main challenge of this research direction is to develop an algorithm to select proper model types automatically. With multiple choices of model types, an intuitive solution to decide the proper model structure is to implement all of these model types, and then the one that best satisfies the criteria (e.g., simulation accuracy) will be singled out. However, such a brute-force selection method is not intelligent and may cost heavy computing resources, especially for the large-scale system modelling tasks, requiring a lot of time and resources to train a model. An efficient solution should be a selection algorithm that can select the proper model type from different choices before training. A possible solution is to regard this challenge as a sparse regression problem, which is similar to addressing the overfitting issues of DNNs in this thesis.

7.2.2. CONVERGENCE OF THE TRAINING

The convergence of DNN training is challenging to analyze, which is influenced by many aspects, i.e., the pre-processing of training data [3], the initialization of weight matrices [10], proper selection of learning rate and batch size, and the complexity of NN [3, 10]. A local convergence theory was developed for mildly over-parameterized two-layer

DNN, which shows the gradient descent can converge to zero with the initial loss below a threshold in [10]. [2] proposed using the efficient conjugate gradient (CG) algorithm to train the RNN, which can accelerate the convergence procedure and help find the optimal solution. In this thesis, although there is no guarantee that the absolute global minimum can be achieved during the training process, the experimental result shows that the convergence trend is noticeable. This is also consistent with the research findings in [3] that state that the backpropagation process can always make it possible to meet practical stopping criteria.

7.2.3. WHITE-BOX MODELLING ON HIGH-DIMENSIONAL SYSTEM

In this thesis, we propose a DNN-like model structure to learn the governing equations of several dynamic systems. Although these systems include both linear and nonlinear ones and can be described by either ordinary or partial differential equations, they still belong to polynomials, which lack operations such as $\sin, \cos, \log$. In the future, we will try more complex experiments on several aspects: a) high-dimensional dynamic system. Actually, most of the current symbolic regression techniques confine the identified systems to low-order systems. For example, [8] made the assumption that the system can be described by a polynomial of low degree (normally less than 3). However, a high-dimensional system such as fluid dynamics [1] is the cornerstone of many applications. We should explore applying the proposed white-box modelling method to high-dimensional dynamic systems, e.g., biology, physics and fluid dynamics, etc. b) model with periodic activation functions *e.g.*, $\sin, \cos$. It is well known that $\sin, \cos$ are non-monotonic periodic functions that can explain the dynamics of many systems, such as the motion of a swinging pendulum. However, previous research has verified that the inclusion of *e.g.*, $\sin, \cos$ in a DNN cannot contribute to the model training process because of their easy convergence to local minima [5]. Besides, the conventional neural networks cannot approximate the periodic dynamics even after a lot of tuning work [11]. In the future, we can try to address this issue by encoding periodic functions in the proposed model structures. c) including division in the model structure. The division operation is also a basic mathematical operation. However, previous works which tried to include the division in the DNN/DNN-like model found that the division performed poorly when the denominator was near 0 [6]. [7] proposed the equation learner ($EQL^{\div}$) structure, which includes the division in the output layer. However, this approach precludes learning simple expressions such as $\sin\left(\frac{x}{y}\right)$. In the future, we should try more practical approaches to encode the division or represent the division using an equivalent expression, e.g., $\frac{1}{x} = \exp\{-\log(x)\}$.

ACKNOWLEDGEMENTS

Time flies. Looking back many years ago, I never considered the possibility of doing a PhD in such a university and such a beautiful country. Since setting foot on the land of Holland four years ago, I have embarked on a new journey. A new country, a new culture, a new language, and a new research topic. All these "new" make the past every day not only unknown, tense, and stressful but also fresh, novel, and exciting. I like this challenging feeling. And I believe the past four years' adventures would be one of the most extraordinary and unforgettable experiences of my life. Now, I am close to the end of this journey. I want to say a heartfelt thank you to everyone I met in those years.

First, I would like to express my special appreciation to Prof.dr.Horváth Imre and Dr. Rusák Zoltán who are the guides leading me to the PhD stage. There is an old Chinese saying that ''teach a man to fish and you feed him for a lifetime". The research methodology I learned from you is such a pillar for my research, which teaches me the proper steps to start a scientific study. Your words and deeds will benefit me all my life.

My deepest gratitude is given to Prof.dr.ir.Martijn Wisse and Dr. Wei Pan. Thank you very much for providing me with the opportunity to conduct my PhD in 3ME. Martijn, you are always a good listener. Every time we had a meeting or a discussion, I could feel my words being heard by you. You managed to make me feel like an equal despite your academic accomplishments. Your warm, humble, considerate, and tolerant personality has set me a goal that I can learn throughout my life. Wei, limited words cannot express my gratitude to you. Thank you very much for your patience and responsibility in supervising my PhD project. As my supervisor, you instructed me to explore a research idea, carry out experiments, write a scientific paper, and prepare for a presentation. The discussions with you were always exciting and rewarding. I admire your academic enthusiasm and many incredible inspirations. I think in the future, I will always miss the scene where you suddenly dropped by to discuss a new idea about my project. The four years I worked with you have had a significant impact on my research and personality and pointed out a way to develop my future career.

I want to express my appreciation to my office mates, who were my first point of reference about whether I was in the right direction. Many thanks to Carlos and Tim, who always encouraged me to be brave and optimistic. Their high-quality research and self-discipline also give me examples that I should strive for. I will not forget the discussions and many parties with Jihong, who reminded me to consider problems from reality. The talents and accomplishments of Linda inspired me that there is a broader sky beyond scientific research. Giovanni and Ajith set the bar of overcoming difficulties with a positive attitude. Chengwei's critical attitude towards research also becomes a mirror of my self-reflection.

I also want to thank friendly colleagues in the CoR department. They are Hai, Rodrigo, Yujie, Desong, Bruno, Alvaro, Tugrul, Luzia, Corrado, Vishal, Maximilian, Thomas, Zhaochong, Max, Yanggu, Xiaolin. I was always happy to see and chat with you guys in the cor-

ridor and social room. Special thanks to Hai, Rodrigo, Tugrul, and Thomas, who helped me a lot to organize experiments. I would also like to thank Hanneke, Karin, Rosanne, Noortje, and Hans. They manage administration matters and provide us with such a pleasant working environment.

I am also very grateful that many friends accompany me on the four year's journey. There were Tao Hou, Na Wu, Jun Xu, Wenting Ma, Sheng Yu, Na Li, Shan Jing, Zimin Xia, Fan Li, Siqi Song, Wei Chang, Qizhi Wei, who gave me a lot of care in study and life. Special thanks to my friends Rui Yan and Yuxuan Feng, who organized many parties especially during the depressing pandemic time. Every time being with you and Langzi Chang, Xiangcou Zheng, Zhaoyinig Ding, Renan Gong, Huan Wang, Biyue Wang, Shuhong Li, Hai Gong always put a smile on my face. Many thanks to Xiangcou, Huan, Maolong, Chengwei, Yusheng, and Shuai Yuan, who also provided me with many opportunities for collaborative scientific research besides friendship.

I want to express my most sincere and deep gratitude to my family. Thanks to my parents, Yuanfeng Zhao and Guangyin Zhou, who gave me life and supported me all way. My father, you are the first person to shed a PhD's dream in my heart when I was only in primary school. You passed away eight years ago. But your expectations for me are always one of the most powerful driving forces in my heart. I dedicate my most profound respect to my mother, who is a great woman. I cannot have a stable life and complete my PhD studies without your support and effort. You are always my strong backing, and I promise I will be the backing of our family in the future. I also want to thank my little sister, Xuanyi Zhou, every time I hear your laughter, it always makes me feel warm and healed.

Finally, I want to thank my country, China, who provided me with the scholarship to finish the PhD. All best wishes to my supervisors, colleagues, friends, family, and my country.

CURRICULUM VITÆ

Hongpeng ZHOU

26-10-1994	Born in Nanyang, China.

EDUCATION

2011–2015	Bachelor in Detection Guidance and Control Technology Harbin Institute of Technology, China
2015–2017	Master in Control science and Technology Harbin Institute of Technology, China
2017–2022	PhD in Mechanical Engineering Delft University of Technology, the Netherlands

AWARDS

2013	National Scholarship (Undergraduate)
2014	Merit Student of Heilongjiang Province
2014	National Endeavor Fellowship
2015	Excellent Graduate (Bachelor) in Harbin Institute of Technology
2015	National Scholarship (Postgraduate)
2017	Excellent Graduate (Master) in Harbin Institute of Technology
2019	Travel Award for the International Conference on Machine Learning

LIST OF PUBLICATIONS

Papers Published During PhD Stage:

1. **Hongpeng Zhou**, Minghao Yang, Jun Wang and Wei Pan, *BayesNAS: A Bayesian Approach for Neural Architecture Search*, International Conference on Machine Learning, 7603-7613, 2019.

2. **Hongpeng Zhou**, Chahine Ibrahim, and Wei Pan, *A Sparse Bayesian Deep Learning Approach for Identification of Cascaded Tanks Benchmark*, arXiv preprint arXiv:1911.06847, 2019.

3. **Hongpeng Zhou**, Chahine Ibrahim, Wei Xing Zheng and Wei Pan, *Sparse Bayesian Deep Learning for Dynamic System Identification*, Automatica, under review, 2021.

4. Yusheng Yang, **Hongpeng Zhou**, Yu Song and Peter Vink, *Identify Dominant Dimensions of 3D Hand Shapes Using Statistical Shape Model and Deep Neural Network*, Applied Ergonomics, 96:103462, 2021. (This paper made minor contribution to the thesis. Its content is not included in this thesis.)

Papers Published During Master Stage:

1. **Hongpeng Zhou**, Ao Chen, Chengming Yang, Jiapeng Yin, Han Yu, *A data-driven KPI prediction method for vehicular cyber physical system*, IEEE 25th International Symposium on Industrial Electronics, 72-77, 2016.

2. **Hongpeng Zhou**, Hao Ju, Tianyu Tan, Tianyi Gao, *A KPI prediction approach with JITL for vehicular Cyber Physical System*, Seventh International Conference on Intelligent Control and Information Processing, 85-90, 2016.

3. Ao Chen, **Hongpeng Zhou**, Yujian An, Wei Sun, *PCA and PLS monitoring approaches for fault detection of wastewater treatment process*, IEEE 25th International Symposium on Industrial Electronics, 1022-1027, 2016.

4. Hengbo Ma, Tianyu Tan, **Hongpeng Zhou**, Tianyi Gao, *Support vector machine-recursive feature elimination for the diagnosis of Parkinson disease based on speech analysis*, Seventh International Conference on Intelligent Control and Information Processing, 34-40, 2016.

5. Hongyan Yang, Lei Liu, **Hongpeng Zhou**, Tianyi Gao, *Multivariate statistic methods for predicting electricity consumption of Beijing*, IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society, 7197-7202, 2016.

6. Zelin Ren, Jian Hou, **Hongpeng Zhou**, *Fault Detection and process monitoring of industrial process based on spherical kernel T-PLS*, IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society, 7161-7166, 2016.

7. Jianqiao Zheng, Hongfang Wang, **Hongpeng Zhou**, Tianyi Gao, *A using of just-in-time learning based data driven method in continuous stirred tank heater*, Seventh International Conference on Intelligent Control and Information Processing, 98-104, 2016.

8. Mingyang Yang, Xuebo Yang, Chengming Yang, **Hongpeng Zhou**, *Key data set selection algorithm based on PLS regression in industrial process*, IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society, 7179-7184, 2016.