A Study on Deep Learning Based Packet Transmission Strategy for Intelligent Network Traffic Control

A dissertation presented
by

Bomin Mao

submitted to
Tohoku University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

Supervisor: Professor Nei Kato

Department of Applied Information Sciences
Graduate School of Information Sciences
Tohoku University

January, 2019

A Study on Deep Learning Based Packet Transmission Strategy for Intelligent Network Traffic Control

A dissertation presented by

Bomin Mao

approved as to style and content by

Professor Nei Kato,
Graduate School of Information Sciences _____

Professor Ayumi Shinohara,
Graduate School of Information Sciences _____

Professor Masanori Hariyama,
Graduate School of Information Sciences _____

Associate Professor Zubair Md. Fadlullah,
Graduate School of Information Sciences _____

Tohoku University
Sendai, Japan.

To My Family

Abstract

Recent years, an increasing number of devices are connected to the Internet for providing users with various kinds of services. Accompanying the era of Internet of Things (IoT), the number of devices connected to the Internet will be three times as high as the global population in 2021. To offer users different services, the connected devices generate and receive the traffic. Therefore, the significant increase in the quantity of the connected devices usually leads to the exponential growth of the global IP traffic. It is forecast by the industry that the global IP traffic will increase nearly threefold over the next 5 years and reach 3.3 ZB annually by 2021. The surging traffic demand does not mean the growing profits for the Internet Service Providers (ISPs). On the other hand, the ISPs are confronted with the problem of declining profits due to the traffic explosion. This is because the main idea behind the routing algorithms has traditionally been remarkably similar and the manner in which the Internet core and the wired/wireless heterogeneous backbone networks are constructed have largely remained unchanged over the years. To accommodate the tremendous growth of network traffic, the ISPs have to reconsider the core network structure and the packet transmission strategy instead of just adding more/larger routers and more/faster links to scale up the Internet core infrastructure, which on the other hand results in a huge cost.

As we know, the computation capacities of different platforms, such as the Central Processing Unit (CPU) and the Graphic Processing Unit (GPU), are enlarged significantly driving by the Moore's Law. For example, the single precision processing power of V100 GPU accelerator launched by Nvidia in June 2017 reached as high as 14028 GFLOPS, while that of S870 GPU in May 2007 was only 1382.4 GFLOPS. Besides the much better experience realized by the increasing computation capacities, some existing technologies have benefited from the more powerful computation platforms and achieved some breakthrough. For instance, as one of the machine learning techniques, the deep neural networks can be trained with much lower time consumption, which lays the solid foundation for the wide applications. Currently, deep learning, emerged from the deep neural networks, has shown its predominant intelligence in many complex activities. Moreover, researchers have also considered this technique to develop the networking management algorithms to improve the performance.

Inspired by the development in the computing hardware and the Artificial Intelligence (AI) technology, in this dissertation, we explore new opportunities in packet processing with deep learning to inexpensively shift the computing needs from rule-based route computation to deep learning based route estimation for high-throughput packet processing. Also, driven by the development of the computation platforms, Software Defined

Routers (SDRs) (programmable routers) have emerged as a viable solution to provide a cost-effective packet processing platform with easy extensibility and programmability. Moreover, multi-core platforms significantly promote SDRs' parallel computing capacities, enabling them to adopt artificial intelligent techniques to manage routing paths. This dissertation first envisions a supervised deep learning system to construct the routing tables and show how the proposed method can be integrated with programmable routers using both CPUs and GPUs. We demonstrate how our uniquely characterized input and output traffic patterns can enhance the route computation of the deep learning based SDRs through both analysis and extensive computer simulations. In particular, the simulation results demonstrate that our proposal outperforms the benchmark method in terms of delay, throughput, and signaling overhead.

Since the labeled data is usually unavailable in some Software Defined Communication Systems (SDCSs) with heterogeneous networks as the data plane, the supervised training manner is not suitable. To alleviate the congestion in the SDCSs with varying traffic patterns, in this dissertation, we utilize Convolutional Neural Networks (CNNs) to intelligently compute the paths according to the input real-time traffic traces. To reduce the computation overhead of the central controller and improve the adaptation of CNNs to the changing traffic pattern, we consider an online training manner. Analysis shows that the computation complexity can be significantly reduced through the online training manner. Moreover, the simulation results demonstrate that our proposed CNNs are able to compute the appropriate paths combinations with high accuracy. Furthermore, the adopted periodical retraining enables the deep learning structures to adapt to the traffic changes.

The above research focuses on the static network scenarios. However, it has been forecast that the traffic generated by the wireless and mobile devices will jump to more than 63% of the global IP traffic by 2021. Therefore, we further propose a Value Iteration Architecture based Deep Learning (VIADL) method to conduct routing design in order to address the limitations of existing deep learning based routing algorithms in dynamic networks. Besides the network performance analysis, this dissertation also studies the complexity of our proposal as well as the resource consumption in different deployment manners. Moreover, this dissertation adopts the Heterogeneous Computing Platform (HCP) to conduct the training and running of the proposed VIADL since the theoretical analysis demonstrates the significant reduction of the time complexity with the multiple GPUs in HCPs. Furthermore, simulation results demonstrate that compared with the existing deep learning based method, our proposal can guarantee more stable network performance when network topology changes.

Acknowledgments

Firstly, this dissertation would have never been completed without the continuous support and guidance from my supervisor, Prof. Nei Kato. I have been really fortunate to have him as my supervisor since he provided me enough freedom and resource to conduct the research I feel interested in. His kindness, prudence, and work of ethics have made my PhD period one of the best periods of my life. I am also grateful to Associate Prof. Zubair Md. Fadlullah for closely watching, directing, and guiding me throughout each stage of this work. His enthusiasm and prompt suggestion always inspire me to work hard towards my goal. I could not find works strong enough to express my gratitude for him. I would like to thank also the other two professors in my thesis committee, Prof. Ayumi Shinohara and Prof. Masanori Harayama, for their interest, concern, and valuable comments on both the writing of the thesis and the defense.

During the three and half years' study overseas, I always feel fortunate to meet my wife, Ms. Wei Zou. She has offered me so much support which enables me to concentrate on my research. Being accompanied by her, to study overseas becomes more meaningful and interesting. I wish to thank my dear parents who were always supporting and encouraging me to strict on my choice. I am deeply grateful for their strong faith in me. I cannot imagine how much hardships they have gone through to bring up me, my brother and sister. I would also link to sincerely thank my brother and sister for that they take my responsibility to look after my parents when I was studying overseas. I would repay them during my rest life.

This dissertation would not be possible without the help of my colleagues and friend, Fengxiao Tang. I would never forget his valuable suggestions, discussions, and help over the years. Special thanks go to Motoko Shiraishi, Keisuke Miyanabe, and Yibo Zhou for sincerely helping me with all the necessary documents. Acknowledgments are also given to Chinese Scholarship Council (CSC) for the strong support that enabled me to pursue the challenging route during my doctoral years. Last, but not least, special thanks to all of my lab-mates for their responsiveness, technical help, and mainly for having contributed to this long journey I had to make to pursue my Ph.D degree.

Contents

Abstract i

Acknowledgments iii

1 Introduction 1

1.1 Background . . . 1
1.2 Breakthrough of Deep Learning . . . 3
1.3 Development of Hardware Computation Capacities . . . 4
1.4 Purpose of Research . . . 6
1.5 Summary and Organization of the Thesis . . . 8

2 Overview of Deep Learning and Traffic Control 10

2.1 Introduction . . . 10
2.2 Overview of Deep Learning Technologies . . . 10
2.2.1 Preliminaries of Deep Learning . . . 12
2.2.2 Two Commonly Utilized Deep Learning Architectures . . . 15
2.2.2.1 Deep Belief Architecture . . . 15
2.2.2.2 Convolutional Neural Networks . . . 19
2.2.3 Different Training Manners . . . 23
2.2.4 Survey of Deep Learning Based Networking . . . 24
2.2.4.1 Network Parameter Prediction . . . 24
2.2.4.2 Intelligent Resource Allocation . . . 25
2.2.4.3 Smart Anomaly Detection . . . 25
2.3 Overview of Traffic Control . . . 26
2.3.1 Traditional Traffic Control Strategies . . . 26
2.3.1.1 Data Link Layer . . . 27
2.3.1.2 Network Layer . . . 27
2.3.1.3 Transport Layer . . . 28
2.3.2 Research on Deep Learning Based Traffic Control . . . 28
2.3.2.1 Network Scenarios and Problem Analysis . . . 29

2.3.2.2 Deep Learning Structure Construction 29
2.3.2.3 Network Performance Analysis 29
2.3.2.4 Computation Analysis and Proposal Deployment 30
2.4 Summary 30

3 Deep Learning Based Routing Algorithm for Core Networks Running on GPU Accelerate SDRs 31
3.1 Introduction 31
3.2 Design of Deep Learning based Routing Strategy 32
3.2.1 Input and Output Design 32
3.2.2 Deep Learning Structure Design 34
3.2.3 Considered Router Architecture 35
3.3 The Procedures of the Proposed Deep Learning based Routing Strategy . . 37
3.3.1 Initialization Phase 37
3.3.2 Training Phase 37
3.3.3 Running Phase 39
3.4 Computation Performance Analysis 40
3.4.1 DBA Precision Analysis 41
3.4.2 Complexity Analysis of the Training Phase 42
3.4.3 Complexity Analysis of the Running Phase 45
3.5 Network Performance Evaluation 47
3.6 Summary 51

4 Online Learning Based Routing Strategy for Software Defined Communication Systems 53
4.1 Introduction 53
4.2 Problem Statement and Model Design 55
4.3 Procedures of Our Proposal 58
4.3.1 Initial Phase 58
4.3.2 Running Phase 60
4.3.2.1 Data Collection 60
4.3.2.2 Routing Judgement 61
4.3.2.3 Retraining Phase 62
4.4 Complexity Analysis 63
4.5 Performance Evaluation 64
4.6 Summary 68

5 Value Iteration based Deep Learning Architecture for Routing in Dynamic Networks **69**
5.1 Introduction . . . 69
5.2 Problem Formulation . . . 70
5.3 Preliminaries . . . 73
5.3.1 Markov Decision Process (MDP) . . . 73
5.3.2 Value Iteration . . . 75
5.4 Design of the Deep Reinforcement Learning Based Routing Strategy . . . 75
5.5 Complexity Analysis From the HCP Perspective . . . 79
5.6 Performance Evaluation . . . 81
5.6.1 Deployment Analysis . . . 83
5.6.2 Performance with Link Failures . . . 85
5.7 Summary . . . 86

6 Conclusion **87**

Appendix **90**
Method to Adjust the Weights and Biases of RBMs . . . 90

Bibliography **93**

Publications **101**

List of Figures

1.1 The Global IP Traffic per Month. 2
1.2 The next generation network paradigm. 5
1.3 The Nvidia GPU processing power roadmap. 6
1.4 Recent inter-disciplinary trends indicate an inter-disciplinary area involving computing systems, computer networks, and machine intelligence. Particularly, network traffic control systems are becoming robust and intelligent owing to the advancement in CPU/GPU technologies and deep learning, respectively. 7
1.5 The research contents of this dissertation. 8

2.1 The architecture of deep neural networks. 11
2.2 The commonly utilized deep learning architectures. 12
2.3 Considered model of the proposed deep learning system. 15
2.4 The process of convolution between two three-dimensional matrices. 20
2.5 The timescales of approaches to congestion control. 27

3.1 Considered system model and problem statement. 32
3.2 Considered input and output design. 33
3.3 The architecture of GPUs and steps of how packets are passed in the GPU-accelerated SDR. 35
3.4 Mean Square Errors (MSEs) of different DBAs. 40
3.5 The time cost of training phase on the chosen GPU and CPU-based SDRs. 45
3.6 The time cost of running phase on the chosen GPU and CPU-based SDRs. 46
3.7 Comparison of network performance under different network loads in our proposal and the bencmark method (OSPF) in terms of signaling overhead, throughput, and average delay per hop. 48
3.8 Comparison of network performance under different signaling intervals in our proposal and the bencmark method (OSPF) in terms of signaling overhead, throughput, and average delay per hop. 49

4.1 The considered structure of SDCS. 54

4.2 An illustrative example: when switches S_1, S_2, and S_3 choose S_5 as the next node to destination S_8, S_5 will be the bottleneck, which means that traffic congestion will easily happen to S_5. 55
4.3 The input of the CNN in our proposal. 56
4.4 The process in the running phase. 62
4.5 The network performance before and after training. 65
4.6 The network performance comparison between the conventional routing protocol and our proposal in terms of packet loss rate and average packet delay. 65
4.7 The throughput comparison in the considered SDCS. 66
4.8 Comparison of SDCS performance under different packet generation rates in our proposal and the bencmark methods (OSPF) in terms of packet loss rate, average packet delay, and throughput. 67

5.1 The considered network topology. 70
5.2 The Heterogeneous Computing Platform (HCP) and the applications built on it. 71
5.3 The proposed Value Iteration Architecture (VIA). 76
5.4 The log time cost of training VIA with the single GPU HCP and the multiple GPUs HCP. 79
5.5 The log time cost of running VIA with the single GPU HCP and the multiple GPUs HCP. 80
5.6 Training computation consumption of networks with different percentages of centralized controlled switches. 82
5.7 Running time cost for networks with different percentages of centralized controlled switches. 83
5.8 Considered network performance for different cases. 84

List of Tables

2.1 Comparison of three training methodologies. 23
2.2 Existing networking research based on deep learning 24
2.3 Some congestion control protocols in the transport layer. 28

3.1 Routing Table Built in R_3. 39
3.2 Effect of different input and output characterization strategies on the network control accuracy for N=16. 42

4.1 The parameters of the considered CNN structure 63

6.1 Comparison of the three deep learning based strategies. 88

List of Acronyms

3G The third Generation of wireless mobile telecommunications technology

4G The forth Generation of wireless mobile telecommunications technology

5G The fifth Generation of wireless mobile telecommunications technology

ADD Addition

AI Artifical Intelligence

AP Access Point

CAIDA Center for Applied Internet Data Analysis

CD Contrastive Divergence

CIFAR Canadian Institute for Advanced Research

CNN Convolutional Neural Network

CPU Central Processing Unit

DBA Deep Belief Architecture

D2D Device to Device

DIV Division

DMA Direct Memory Access

DNN Deep Neural Network

DRAM Dyanmic Random Access Memory

ECN Explicit Congestion Notification

EXP Exponention

FiWi Fiber-Wireless

HTTP Hyper Text Transfer Protocol

IoT Internet of Things

ISP Internet Service Provider

GB Giga Byte

Gbps Giga byte per second

GHz Giga Hertz

GPU Graphic Processing Unit

HCP Heterogeneous Computing Platform

IEEE Institute of Electrical & Electronics Engineers

IETF Internet Engineering Task Force

IP Internet Protocol

LSTM Long-Short Term Memory network

MDP Markov Decision Process

MSE Mean Square Error

ML Machine Learning

MOS Mean Opinion Score

MUL Multiplication

NAPI Northbound Application Programming Interface

NEG Negation

NIC Network Interface Card

NN Neural Network

OD Origin Destination

OSPF Open Shortest Path First

QoS Quality of Service

QoE Quality of Experience

RAM Random Access Memory

RBM Restricted Boltzmann Machine

ReLu Rectified Linear Units

SAPI South Application Programming Interface

SDCS Software Defined Communication System

SDN Software Defined Networking

SDR Software Defined Router

SIMD Single Instruction Multiple Data

SM Streaming Multiprocessor

SP Shortest Path

SQRT Square Root Operation

SUB Subtraction

SVM Support Vector Machine

TCP Transmission Control Protocol

V2X Vehicle to Everything

VIA Value Iteration Architecture

VIADL Value Iteration Architecture based Deep Learning

WMN Wireless Mesh Network

WLAN Wireless Local Area Network

XCP eXplicit Control Protocol

ZB ZettaByte

Chapter 1

Introduction

1.1 Background

In recent decades, it has been witnessed that the significant improvement has been fulfilled in the communication field to provide people with services of better quality and more convenience due to the development of various technologies, such as Fiber-Wireless (FiWi) [1], Device to Device (D2D) [2], and 5G [3]. For example, the emerging mobile telecommunication technology, 5G, can offer people Internet connections with a speed of 1 Gbit/s [4], while 3G proposed in 2000 can only provide an information transfer rate of lower than 1 Mbit/s [5]. Inspired by the development, new Internet services requiring much higher packet transfer rate become a reality. The 5G network can transfer the real-time road information to automatic vehicles in time for avoiding the potential accidence [6]. Another more encouraging application is the Internet of Things (IoT), which enables everything to be connected to the Internet [7]. The IoT technology creates opportunities for more direct integration of the physical world into computer-based systems, resulting in efficiency improvements, economic benefits, and reduced human exertions [8].

On the one hand, the development of communication technologies is beneficial to human being's life. On the other hand, as more Internet services are emerging, the Internet Service Providers (ISPs) are confronted by several challenges [9]. Recent years, the network traffic is increasing tremendously. The global Internet Protocol (IP) traffic per annum exceeded the ZettaByte (ZB) threshold at the end of 2016, and is expected to increase up to 3.3 ZB by 2021 as shown in Fig. 1.1 [10]. It becomes critically urgent to improve the traffic control performance in order to provide the Internet services of guaranteed quality. At the same time, it is expected that the number of devices connected to the IP networks will be three times as high as the global population in 2020 [10]. As the packets are generated by various devices for different services, the heterogeneous packet configurations as well as the corresponding different requirements for the Quality of Service (QoS) further complicate the problem.

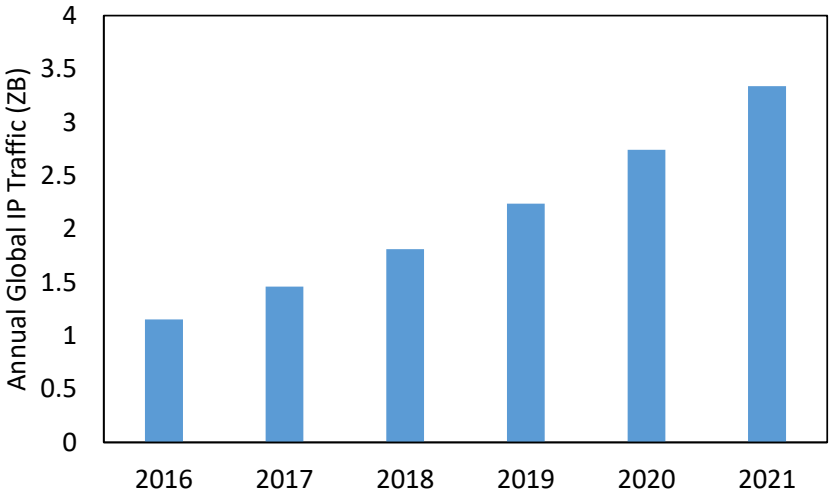


Figure 1.1: The Global IP Traffic per Month.

To accommodate the tremendous growth of network traffic, the Internet core infrastructure has simply continued to scale up by adding more/larger routers and more/faster links [11]. The increasingly larger core networks have driven the architectures of core routers to be more powerful in computation and switching capacities. Even with the recent surge in the data traffic, the network operators are confronted by the challenges of traffic management for ensuring the QoS as well as dealing with the declining profits [12], which is because the hardware solution results in a massive investment splurge. Besides the implementation of more infrastructure, scholars have conducted a lot of meaningful research on the traffic control algorithms, which can be utilized in specified scenarios for some performance improvement [12, 13]. However, most of these strategies cannot be practically applied, for which two reasons can be summarized. First, current routers and switches still consist of proprietary hardware, meaning that the routing strategies are integrated into the specified hardware to fulfill the packet forwarding tasks. Therefore, the software aspect of traffic management mainly focuses on the application of new routing strategies that may not be possible until a new generation of capable hardware architectures emerge [14]. Second, current packet transmission strategies lack the ability of reconfiguration to fit for the changing network environment. Due to the limitations of fixed hardware architectures, many factors are neglected when designing the algorithms to reduce the computation overhead. Therefore, the current packet transmission strategies still follow the traditional manner (e.g., the Shortest Path (SP) algorithm and so forth) which chooses the paths according to the maximum or minimum metric values [15]. To design an efficient traffic control strategy, it is necessary to improve the packet forwarding algorithms and reconsider the hardware architecture.

1.2 Breakthrough of Deep Learning

Recent years, the field of Artificial Intelligence (AI), dictated by deep learning, is drawing the attentions from the academia and industry. Technology giants such as Google, Microsoft, Facebook, Amazon, Nvidia, and others are investing heavily with their powerful computing resource to drive AI research, particularly aiming at deep learning breakthroughs [16]. As the most efficient and promising AI technique, deep learning is now a thriving field with a widely covered active research topics and relevant applications ranging from speech recognition to driver-less smart cars [17]. In 2006, a group of researchers brought together by the Canadian Institute for Advanced Research (CIFAR) introduced an unsupervised learning method to pretrain the deep forward neural networks, which enables the hidden layers of deep neural networks to extract the features from the input data without requiring labeled data [18]. Since this method pretrains the neural networks layer by layer with the reconstruction objective, the weights of a deep neural network can be initialized to sensible values, which significantly overcomes the difficulty in training a deep architecture. As the deep architectures can learn more complex features, in March 2016, Google's DeepMind AI program firstly adopted the deep learning technique in the board game "Go", which is concerned with more than 2×10^{170} legal positions [19]. And the developed programs, AlphaGo and AlphaGo Zero managed to beat the world top players, Lee Sedol and Ke Jie, in October 2015 and May 2017, respectively [19, 20]. The breakthrough of the deep learning technology in board games inspires people to realize its huge potential and also encourages researchers in different fields to develop and discover the intelligent solutions of complex problems. Nowadays, deep learning has been widely studied and applied in the fields of medical diagnose, automatic drive, and industrial control [21, 22, 23].

One important reason for the wide applications of deep learning is because of its flexibility resulting from the various architectures, different training manners, and corresponding numerous algorithms. For instance, as a machine learning method, the constructed deep learning architectures can be trained with different manners according to the purpose. Specifically, the supervised learning is usually applied for classification and regression problems, and the unsupervised learning is suitable for clustering, dimensionality reduction problems, while the reinforcement learning is mainly utilized to learn a policy [24]. To fit for the various application scenarios and purposes, different deep learning architectures have also been developed, such as the Deep Belief Architecture (DBA), Convolutional Neural Network (CNN), Long-Short Term Memory network (LSTM), which significantly increases the flexibility and efficiency of this technique. Therefore, it can be concluded that deep learning is one of the most important paradigm technologies in the future [25].

As mentioned above, the deep learning technique can be utilized to effectively analyze the complex relationships among multiple inputs through training with example data. The trained deep learning architecture can predict the values of some parameters when we input the necessary information. Since the deep learning technique has exhibited superior performance in extremely difficult applications which have traditionally been dominated by humans [19, 20, 25], e.g. board games, it is a promising technology to address the challenges of network traffic control. Moreover, considering the increasing complexities and surging demand in current communication networks, the deep learning technique provides an efficient tool to analyze the network condition and improve the performance. For instance, the deep learning technique can be adopted to accurately predict the traffic changes in heterogeneous networks, which can be considered to improve the packet transmission paths and allocate the network resource, resulting in reduced probabilities of network congestion.

1.3 Development of Hardware Computation Capacities

Besides the endeavors in software for improving the algorithms, it is also necessary to rethink the core networks. If we want to deploy the deep learning based network traffic control strategies, the network architectures as well as the routers/switches need to be taken into account. As we know, the infrastructures of the Internet backbone networks have remained largely unchanged since the invention [11]. As one of the main components in the core networks, the practically deployed routers still rely on the circuit structure to accomplish the packet switching tasks [26]. Even though the circuit switching can achieve a throughput more than 100 Gbps, the hardware-based architecture lacks flexibility to fit for different routing algorithms, for which the main idea behind the routing algorithms has traditionally been remarkably similar [27]. On the other hand, if we want to apply some new networking algorithms to accommodate the increasing traffic, it is necessary to redesign the hardware architecture of routers and replace existing architecture with the newly developed one, which leads to invaluable expense and time cost.

To address the problems in traditional core network architectures, researchers have proposed the Software Defined Networking (SDN) [28]. Different from the traditional networks which integrate the algorithms into the proprietary hardware architectures to fulfill management in high efficiency, as shown in Fig. 1.2, the proposed SDN consists of three planes: the data plane, control plane, and the application plane, in which the complex network control logics are separated from the data plane to the central controllers composed by various computation platforms [29, 30, 31]. Moreover, since they

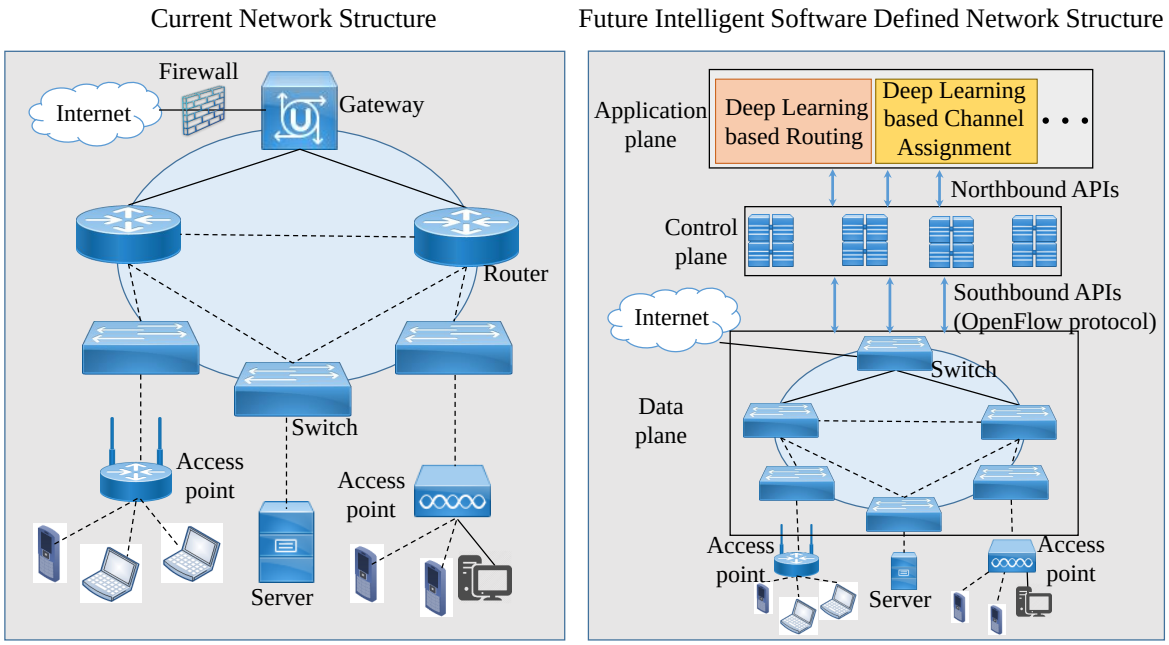


Figure 1.2: The next generation network paradigm.

are based on the general architecture, the SDN controllers allow the upgrade of network management algorithms to be fulfilled by just updating the corresponding applications, which is more flexible than conventional networks [12, 32]. Specifically, any new network management application can be installed in the controller through the Northbound Application Programming Interfaces (NAPIs), while the communications between the controller and switches are fulfilled via the Southbound Application Programming Interfaces (SAPIs) [29, 33]. Due to the advantages in flexibility and simplifications, the SDN technology has been regarded as the next network paradigm as shown in Fig. 1.2 [34]. Moreover, encouraged by the idea of the SDN, researchers have considered the use of software-defined infrastructure which provides the commodity hardware architecture to install the programmable routing strategies to carry out packet processing and transmission, such as the Software Defined Routers (SDRs) [14] and Heterogeneous Computation Platforms (HCPs) [35]. As the critical components of SDNs these software-defined architectures are required not only to support the software-based packet transmission but also to flexibly execute other functions according to the network operators' needs.

To enable the software-defined architectures to accomplish the complex network management, a general hardware architecture with enough computation capacity is necessary. Therefore, the software-defined architecture integrating modern computation platforms is a promising choice to redesign our backbone networks. As we know, the hardware computation capacity has been significantly improved driven by the Moore's Law [36]. Fig. 1.3 gives the processing power roadmap of Nvidia's Graph Processing Units (GPUs), which clearly shows the step-by-step improvement of the computation capacity. The improve-

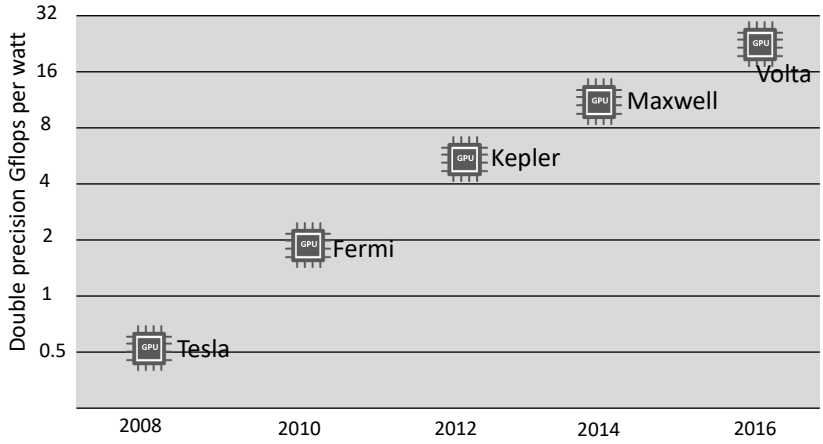


Figure 1.3: The Nvidia GPU processing power roadmap.

ment makes it possible to consider the commodity hardware for manufacturing network infrastructure. Moreover, to improve the processing throughput performance comparable to that of the proprietary-hardware-based routers, researchers as well as networking manufacturers have explored multi-core-based architectures which consist of the Central Processing Units (CPUs) and Graphics Processing Units (GPUs). As we know, the GPUs can execute the same program to process different sets of data in a parallel fashion, while the CPUs undertakes different instructions at the same time [37]. The cooperation of GPUs and CPUs can significantly promote the efficiency to conduct the network management work. Therefore, the software-defined architectures can be regarded as a competent candidate for conducting the deep learning based routing strategies in modern backbone networks.

The improvement of network architectures and the hardware computation capacities enables the routers/switches to forward packets more efficiently [26]. Also, the general hardware architectures driven by the GPUs pave the way to adopt deep learning in networking field. Different from conventional traffic control methods which neglect many factors to reduce the analyze complexities, once the computing requirement is satisfied, the deep learning technique enables researchers to take more parameters into account for improving the calculation accuracy. Furthermore, the rich computation resource allows the researchers to learn the traffic control strategies from past network traces through the deep learning technique, which significantly overcomes the difficulty in analyzing the exact relationship among multiple parameters.

1.4 Purpose of Research

After introducing the increasing traffic overhead in current networks as well as the development in the deep learning and hardware, it is expected that the future network traffic

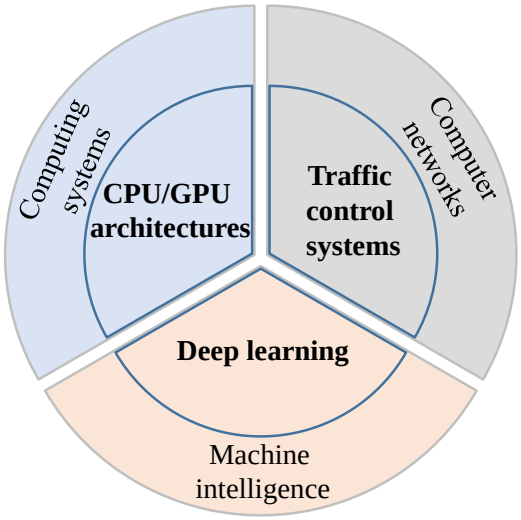


Figure 1.4: Recent inter-disciplinary trends indicate an inter-disciplinary area involving computing systems, computer networks, and machine intelligence. Particularly, network traffic control systems are becoming robust and intelligent owing to the advancement in CPU/GPU technologies and deep learning, respectively.

control will be an integration of the state-of-art techniques in the communication, computer, and computing fields as shown in Fig. 1.4. However, in this paper, we will still survey the existing traffic control methods to analyze the shortcomings. And inspired by the advantages of deep learning as well as the development of computation capacities, we will attempt to adopt the emerging techniques to improve the traffic control performance. Even though various algorithms based on deep learning to optimize the network performance have been proposed, these approaches do not focus on the network traffic control. In this dissertation, we will focus on the challenges of network traffic control as shown in Fig. 1.5 and extend our research from the following four fields.

Firstly, in this paper, to address the increasing traffic overhead, the existing traffic control strategies need to be studied and analyzed. Since the traffic control needs to be cooperatively fulfilled by different layers, in this paper, we need to do some analysis about the promising directions. Also, the deep learning technique should be discussed since this topic is still new in the networking field. The functions and characteristics of different training manners and architectures need to be studied so as to choose the best one to improve the accuracy and reduce the computation overhead.

Secondly, most of existing research simply utilizes the supervised learning to train a neural network for future predictions, which does not carefully consider the network characteristics. As the input and output of the deep learning architectures impact on the structure design and the prediction accuracy, the characterizations of input and output should be decided according to the considered factors. The purpose of my research is to analyze the factors deeply concerned with the routing decision of core networks and characterize the input and output.

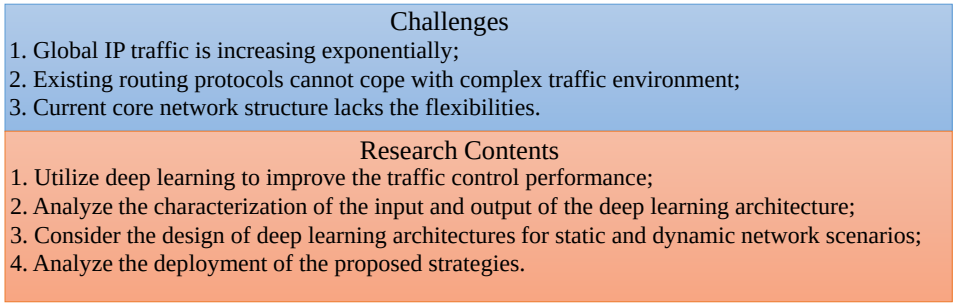


Figure 1.5: The research contents of this dissertation.

The third problem is that most of existing deep learning based research just focuses on static network topology and the trained architectures cannot be applied in a different network scenario. Once the network topology changes, such as some links fail, the trained architectures have to be retrained with new data, which consumes a lot of computation resource and causes some delay. On the other hand, if we do not retrain the deep learning architectures when network changes, the prediction accuracy decreases sharply, losing the advantages of deep learning. In this dissertation, we consider different network scenarios and analyze how to design the intelligent routing algorithms to improve the traffic control for both static and dynamic network scenarios.

Furthermore, current research focuses on the design of algorithms and neglects analysis of the computation resource consumption. Since the deep learning methods are concerned with more computation overhead compared with traditional strategies, our research also discusses the computation consumption. Moreover, the adopted hardware architecture significantly impacts on the computation complexity, while the deployment is a critical problem for the practical application of the proposed deep learning based traffic control methods. Therefore, in this paper, the adopted hardware architectures and the deployment to efficiently conduct the deep learning based methods are also analyzed considering the requirement of the deep learning architecture and the characteristics of network structures.

1.5 Summary and Organization of the Thesis

To explore the deep learning technique to alleviate the increasing network traffic overhead, in this dissertation, we attempt to develop the intelligent packet transmission strategies. We first discuss the main challenges for the Internet networks and survey the existing research to tackle these challenges. The new emerging technique, deep learning, is also introduced. In this part, besides introducing the preliminary knowledge, we focus on some commonly adopted deep learning architectures and the existing intelligent approaches. Based on these introductions, we explain our proposals for different network scenarios.

The remainder of this paper consists of five chapters.

Chapter 2 mainly includes two parts. Firstly, the deep learning technique is introduced in this part. The preliminaries of deep learning are explained, including the forward and backward propagations. Then, the three training manners as well as several deep learning architectures are discussed. Moreover, we survey some existing deep learning based research in the networking field. Secondly, we analyze the global network traffic burden and the conventional strategies to alleviate the overhead. Specifically, the characteristics of heterogeneity and complexities of current networks are emphasized, which leads to extreme difficulties in designing the traffic control algorithms. Considering the promising application of deep learning, we discuss the research contents to adopt the deep learning to design the packet transmission strategy for improving the traffic control. A summary of this chapter is finally given.

Chapter 3 proposes a deep learning based routing strategy for the static backbone networks. In this chapter, we analyze the characterizations of the input and output for constructing the deep learning architectures to improve the traffic control performance. The DBAs are utilized to predict the next nodes and a corresponding path construction strategy is proposed in this chapter. To run the intelligent routing strategy efficiently, we consider the GPU accelerated router architecture and give detailed analysis of the packet processing steps. Simulations also evaluate the improvement in terms of the computation efficiency and network performance.

Chapter 4 proposes an online learning based routing strategy for the considered Software Defined Communication System (SDCS). Since the traffic pattern in some network keeps varying, the trained deep learning architecture are not fit for the changed surrounding. Also, it is impossible to collect enough training data to cover all the potential traffic patterns for supervised learning. To tackle this problem, we consider collecting the real-time traffic trace to perodically retrain the utilized CNNs periodically. Then, we conduct the simulations to evaluate the performance.

Chapter 5 studies the deep learning based routing algorithms for dynamic network topology. To address the limitations that most of the adopted deep learning architectures are related to the network topology, we attempt to utilize the deep reinforcement learning method to learn the routing policy beyond the network shape. We consider the Value Iteration Architecture (VIA) to predict the next node with the network topology and the Origin-Destination (OD) information as the input. The HCP is utilized to run the proposed intelligent routing method. We evaluate the network and computation performance through simulations.

Chapter 6 finally concludes the thesis. The network performance improvement brought by the deep learning technique is summarized in this chapter.

Chapter 2

Overview of Deep Learning and Traffic Control

2.1 Introduction

As we mentioned in the Chapter 1, the global networks are confronted by increasing traffic overhead and growing complexity. Since it has been illustrated the superiority over human beings in complex activities, such as the games, image classification, and speech recognition, the technology of deep learning is promising to alleviate the traffic overhead. Before discussing the proposed deep learning based traffic control strategies, it is necessary to introduce some preliminaries of this new technique. We will study the mainly concerned calculations as well as the three training manners. After that, several commonly utilized deep learning architectures are discussed, of which the related applications in the networking field are explained to evaluate the advantages of deep learning in performance optimization. Since our purpose is to address the traffic control challenge, we will also survey the traditional strategies in different layers. Then, we analyze the research contents of adopting the emerging AI technology to alleviate the traffic challenge in current networks.

2.2 Overview of Deep Learning Technologies

As one of the most important and basic machine learning architecture, neural networks are one of the most beautiful programming paradigm ever invented to solve complex problems via data analysis. However, this method has not aroused so much attention compared with other machine learning strategies [38], such as Support Vector Machine (SVM) [39], for more than 20 years. Unitl 2006, a Canadian research team improved the training method, which significantly increased the prediction accuracy of neural networks [17].

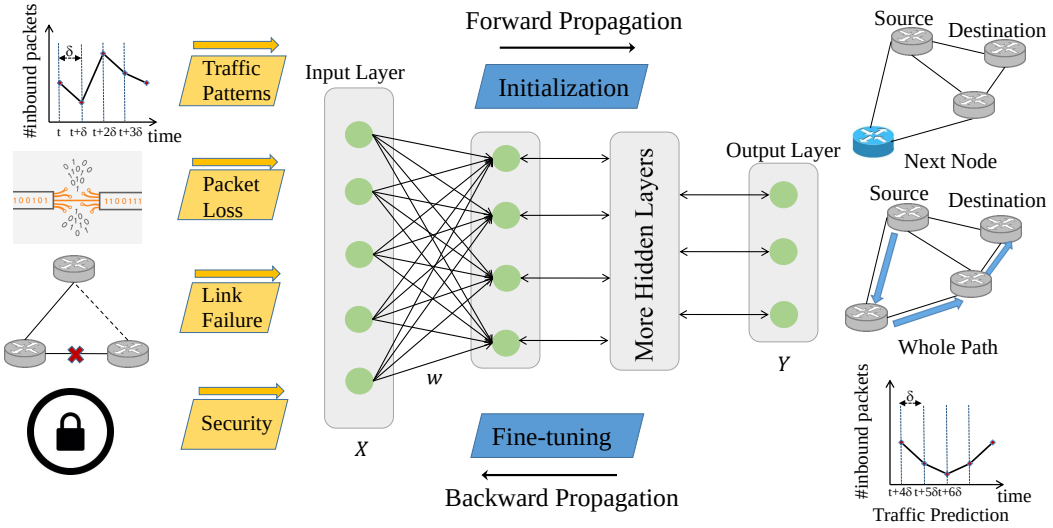


Figure 2.1: The architecture of deep neural networks.

Therefore, it became practical to deploy a large scale of neural networks, referred as deep learning, resulting in the high efficiency in solving complex problems. Fig. 2.1 shows the architecture of deep neural networks, which can be regarded as the most basic deep learning structure [40]. It can be found that it is composed by the input layer, the output layer, and multiple hidden layers. Each layer consists of several units, and each unit is connected to all the units in the adjacent layers through weighted links. If we input some values to the first layer, we can obtain the output through the layer-by-layer propagation of the deep neural networks. The meaning of the output is defined according to our purpose, which can be the values of some predictable parameters, or binary values for classification purpose, or the possibilities of different policies. And the output values are not only dependent on the input layer, but also impacted by the weighted links and the hidden layers. Therefore, we can conclude that the deep neural network represents the relationship between the input and output and the architecture is re-configurable through adjusting the weighted links and hidden layers.

To adopt the deep neural network for definite problems, we first need to characterize the input and output according to our considered scenario and purpose. For example, if the deep learning is utilized to judge whether the network will be congested or not in the next time interval, then the input can be current network traffic patterns, while the output can consist of binary units, of which 1 represents yes and 0 denotes not. Since the characterizations of input and output as shown in Fig. 2.1 impact on the prediction accuracy as well as the design of other layers, it is one of the most important steps for the application of deep learning [41]. As we mentioned in the last paragraph, the architecture of the deep neural network represents the relationship between the input and output, how to adjust the weighted links and hidden layers is vitally critical for the final performance. Moreover, beneficial from the widely meaningful research from the academia and industry,

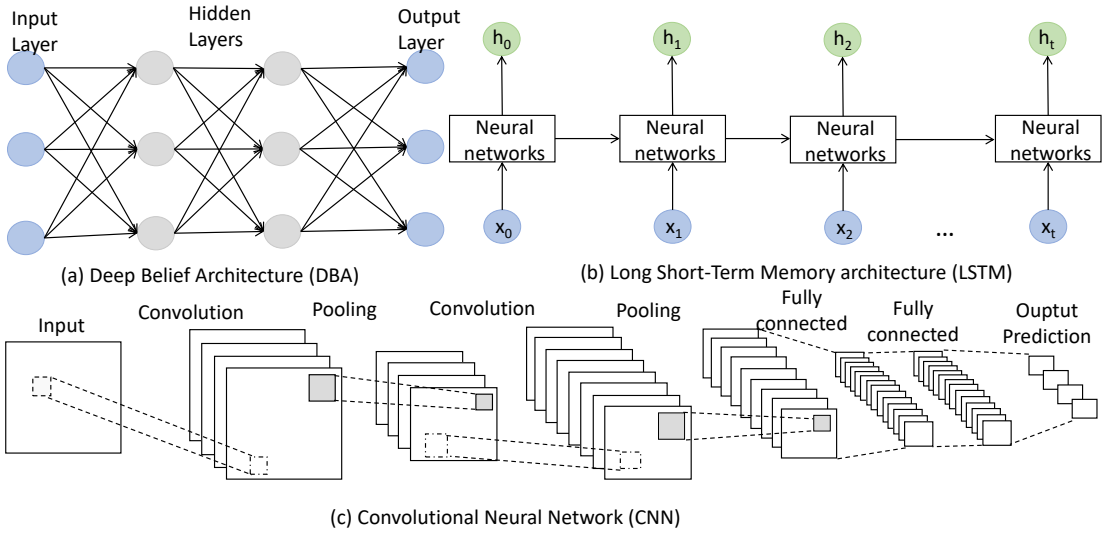


Figure 2.2: The commonly utilized deep learning architectures.

scholars have constructed various deep learning architectures, such as the DBA, LSTM, and CNN as shown in Fig. 2.2, to address different scenarios and problems. Since a suitable deep learning architecture can improve the prediction accuracy and reduce the computation overhead, it also needs to be carefully considered when applying this new technique. Furthermore, similar to other machine learning methods, the deep learning architectures can be trained with the manners of supervised/unsupervised/reinforcement learning with different data formats [42]. And the three training manners can be adopted for different purposes. For example, the unsupervised learning is usually adopted for classification, while the supervised learning can be applied to predict the values of some parameters. This section will be expanded from these aspects.

2.2.1 Preliminaries of Deep Learning

Before applying the deep learning technique in our research, we need to know how it works. Since the calculation process varies from architectures, we focus on the most common steps. We still choose the deep neural network as shown in 2.1 as an example. For describing the computations more clearly, we choose two adjacent layers, the $(l-1)^{th}$ and l^{th} layers as an example. And l^{th} has $n^{(l)}$ units, denoted as $U^{(l)} = \{u_i | i = 1, 2, \cdots, n^{(l)}\}$. Therefore, for the first layer, $X = U^{(1)}$, while $Y = U^{(L)}$ for the final output layer, if we assume the deep neural network has L layers. For each two adjacent layers, the units satisfy the following relationship [40]:

$$u_j^{(l)} = f\Big(\sum_{i=1}^{i=n^{(l-1)}} u_i^{(l-1)} w_{ij}^{(l)} + b_j^{(l)}\Big), \tag{2.1}$$

where $w_{ij}^{(l)}$ represents the weight of the link connecting the units $u_i^{(l-1)}$ and $u_j^{(l)}$. $b_j^{(l)}$ is the bias of unit $u_j^{(l)}$. $f(x)$ is the activation function. It can be found that to get the values of units in l^{th} layer, we usually utilize a transfer function to activate the sum of the weighted units in the $(l-1)^{th}$ layer. And the most popular activation functions are given as follows [43]:

$$Sigmoid: \quad f(x) = \frac{1}{1+exp(-x)}, \tag{2.2}$$

$$Tanh: \quad f(x) = \frac{1-exp(-2x)}{1+exp(-2x)}, \tag{2.3}$$

$$ReLu: \quad f(x) = max(0, x). \tag{2.4}$$

It can be found that if we know the values of input layer and initialize the weights and biases, we can calculate the units layer by layer, of which the process is usually called the forward propagation as shown in Fig 2.1. Here, note that the methods to initialize the values of weights and biases can be referred to [44]. After obtaining the predicted output denoted as $Y^{'}$ ($Y^{'} = U^{(L)}$), we can define a loss function to measure how good a prediction model is and the function can be minimized to optimize the prediction. And for the given input X, since the value of the loss function is related to the weights and biases, it can be denoted as $C(W, B)$, where W and B are the weight matrix and bias matrix, respectively. The process to minimize the value of the loss function is termed the training. A most commonly used method of finding the minimum point of $C(W, B)$ is "gradient descent". Moreover, to define a loss function depends on a number of factors, such as the presence of outliers, choice of machine learning algorithm, time efficiency of gradient descent, ease of finding the derivatives and confidence of prediction [45]. We can utilize the supervised learning as an example to define the loss function. As we mentioned earlier, the training data of the supervised learning is labeled, denoted as (X, Y). And the purpose of training in supervised learning manner is to adjust the weights and biases to minimize the distance between the predicted output $Y^{'}$ and given output Y. Then, the loss function for the supervised learning can be denoted as below:

$$C(W, B) = \frac{\sum_{i=1}^{i=N}(Y_i - Y_i^{'})^2}{N}. \tag{2.5}$$

N denotes the number of training data.

To minimize $C(W, B)$, we apply the gradient descent method to adjust the values of weights and biases. We use Equations 2.6 and 2.7 to explain the key ideas.

$$w_{ij}^{(l)} := w_{ij}^{(l)} - \eta \frac{\partial C(W, B)}{\partial w_{ij}^{(l)}}, \tag{2.6}$$

$$b_j^{(l)} := b_j^{(l)} - \eta \frac{\partial C(W,B)}{\partial b_j^{(l)}}, \tag{2.7}$$

where η is the learning rate which can be adjusted to balance the convergence speed and prediction accuracy. Be repeatedly applying the update rule, we can hopefully find a minimum value of the cost function. However, since $C(W,B)$ is a complex function of the weights and biases, the problem is how to calculate the values of $\frac{\partial C(W,B)}{\partial w_{ij}^{(l)}}$ and $\frac{\partial C(W,B)}{\partial b_j^{(l)}}$. To solve this problem, we need to analyze the propagation process of the errors in the deep neural networks. We utilize δ to denote the error, then we can obtain the following equation [46].

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}}, \tag{2.8}$$

where $z_j^{(l)}$ represents the unit's value before activation in the deep neural networks and can be calculated according to the following equation:

$$z_j^{(l)} = \sum_{i=1}^{n_{l-1}} w_{ij}^{(l)} u_i^{(l-1)} + b_j^{(l)}. \tag{2.9}$$

According to chain rule described in Equation 2.9, we can rewrite the Equation 2.8 as below:

$$\delta_i^{(l)} = \frac{\partial C}{\partial z_i^{(l)}} = \sum_j \frac{\partial C}{\partial z_j^{(l+1)}} \frac{\partial z_j^{(l+1)}}{\partial u_i^{(l)}} \frac{\partial u_i^{(l)}}{\partial z_i^{(l)}} = \sum_j \delta_j^{(l+1)} w_{ij}^{(l+1)} f'(z_i^{(l)}), \tag{2.10}$$

From Equation 2.10, it can be found that the error can be calculated from the back layer to the front layer, which is usually call the back propagation as shown in Fig. 2.1. After obtaining the unit error of each layer, we can deduce the partial differential values of weights and biases as shown in Equation 2.11 and 2.12.

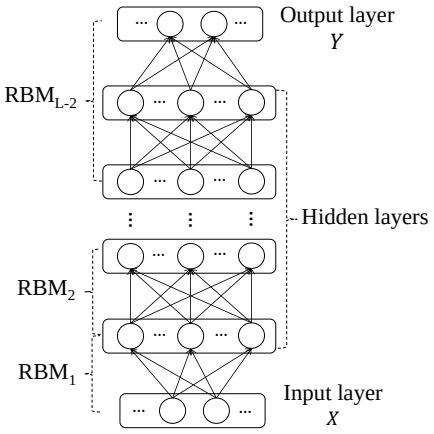
$$\frac{\partial C}{\partial w_{ij}^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} \frac{\partial (w_{ij}^{(l)} u_i^{(l-1)} + b_j^{(l)})}{\partial w_{ij}^{(l)}} = u_i^{(l-1)} \delta_j^{(l)} \tag{2.11}$$

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \frac{\partial (w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)})}{\partial b_j^{(l)}} = \delta_j^{(l)} \tag{2.12}$$

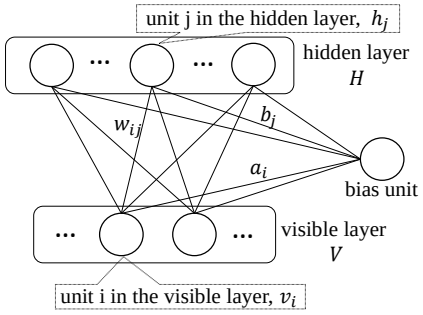
The training process can be conducted by using Equations 2.11 and 2.12 to repeatedly update the weights and biases according to Equations 2.6 and 2.7 till the value of cost function converges. In this way, we can optimize the prediction accuracy of the deep neural networks.

2.2.2 Two Commonly Utilized Deep Learning Architectures

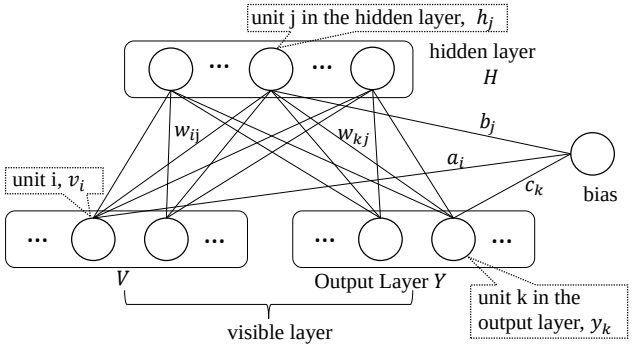
As we mentioned earlier, researchers have developed various deep learning architectures for different application scenarios. As we cannot cover every architecture in this paper, we concentrate on two widely applied structures which will be utilized in our proposals: DBA and CNN.



(a) Considered L-layers DBA.



(b) RBM



(c) Last RBM.

Figure 2.3: Considered model of the proposed deep learning system.

2.2.2.1 Deep Belief Architecture

The DBA is one of the most commonly utilized deep learning models as shown in Fig. 2.3a. As shown in the figure, we assume the DBA consists of L layers, the input layer, X, the output layer, Y, and the $(L-2)$ hidden layers. And the DBA can be also seen as a stack of $(L-2)$ Restricted Boltzmann Machines (RBMs) and a logistic regression layer as the top layer [18]. The structure of each RBM is shown in Fig. 2.3b. It can be seen that each RBM consists of two layers, the visible layer, V, and the hidden layer, H. The units in the two layers are connected through weighted links while those in the same layer are not connected. It should be noted that a weighted bias is given to each unit in both layers. The term w_{ij} denotes the weight of the link connecting the unit i in the

visible layer and the unit j in the hidden layer. Also, a_i and b_j represent the bias of unit i in the visible layer and that of unit j in the hidden layer, respectively. The learned units' activated values in the hidden layer are used as the "visible data" for the upper RBM in the DBA. As we mentioned in Sec. 2.2, researchers made great breakthrough in training the deep learning architectures. And according to the research work [18], the deep learning training process consists of two steps: the Greedy Layer-Wise training to initialize the structure and the backward propagation process to fine-tune the structure. For a DBA, the initial process is to train every RBM which is an unsupervised learning process for the reason that an RBM is an undirected graphical model where the units in the visible layer are connected to stochastic hidden units using symmetrically weighted connections as depicted in Fig. 2.3b [18]. While training an RBM, sets of unlabeled data are given to the visible layer, and the values of the weights and biases are repeatedly adjusted until the hidden layer can reconstruct the visible layer. Therefore, the hidden layer after training can be seen as the abstract features of the visible layer. Training an RBM is the process to minimize the reconstruction error with the hidden layer. To mathematically model the training process, we use a log-likelihood function of the visible layer given as follows. Then, the training process is to update the values of weights and biases to maximize the value of the log-likelihood function.

$$l(\Theta, A) = \sum_{t=1}^{N} \log p(V^{(t)}), \tag{2.13}$$

where Θ denotes the vector consisting of all the values of the weights and biases of the hidden layer. Θ can be written as $\Theta = (W, B)$. W and B represent the vectors consisting of all the weights, w_{ij}, and biases of the hidden units, b_j, respectively. If we use θ to denote the unit in Θ, then θ can be any w or b. A consists of the biases of the visible units, a_i. N denotes the number of training data. $V^{(t)}$ is the t^{th} training data, probability of which is $p(V^{(t)})$.

To maximize $l(\Theta, a)$, we can use the gradient descent of $l(\Theta, A)$ to adjust W, A, and B, which can be described as in Equations 2.14 and 2.15.

$$\theta_i := \theta_i + \eta \frac{\partial l(\Theta, a)}{\partial \theta_i}, \tag{2.14}$$

$$a_i := a_i + \eta \frac{\partial l(\Theta, a)}{\partial a_i}, \tag{2.15}$$

where η is the learning rate in deep learning.

To calculate the value of $p(V)$ (representing any $p(V^{(t)})$), we need to model the RBM as an energy model since the RBM is a particular form of log-linear Markov Random Field (MRF) [25]. The energy function, $E(V, H)$, and the joint probability function, $p(V, H)$,

are defined as follows.

$$E(V,H) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_i \sum_j h_j w_{ij} v_i \tag{2.16}$$

$$p(V,H) = \frac{e^{-E(V,H)}}{\sum_V \sum_H e^{-E(V,H)}} \tag{2.17}$$

where v_i and h_j are the unit i in the visible layer and the unit j in the hidden layer shown in Fig. 2.3b, respectively. Also, the relationship between $p(V)$ and $p(V,H)$ can be expressed as follows.

$$p(V) = \sum_H p(V,H) \tag{2.18}$$

We can use Equations 2.13 to 2.18 to obtain the values of θ [18]. However, the complexity of the calculation of $\sum_V \sum_H$ in Equation 2.17 is $2^{n_V+n_H}$, which is extremely high (n_V and n_H represent the dimensions of vectors V and H, respectively). Another problem is that to calculate Equation 2.17, it is necessary but impossible to consider all the possible values of V and H instead of only the obtained training data. To solve these problems, Hinton *et al.* proposed the Contrastive Divergence (CD) method [47]. The main idea of CD is to use the Gibbs Sampling method to sample the values of V and H to approximate the real values since the conditional distribution probability of one layer (while the value of the other layer is given), e.g., $p(V|H;\Theta,A)$, can be calculated. The detailed procedures of CD is given in the Appendix 6 [47]. As the value of every unit is independent on the other units in the same layer, when one layer is fixed, the conditional distribution probability of the other layer can be calculated as follows,

$$p(V|H;\Theta,A) = \prod_i p(v_i|H;\Theta,A), \tag{2.19}$$

$$p(H|V;\Theta,A) = \prod_j p(h_j|V;\Theta,A), \tag{2.20}$$

where $p(V|H;\Theta,A)$ and $p(H|V;\Theta,A)$ are the conditional probability of V given H and the conditional probability of H given V, respectively. $p(v_i|H;\Theta,A)$ is the conditional probability distribution of unit i in the visible layer when the hidden layer is fixed. Also, $p(h_j|V;\Theta,A)$ is the conditional probability distribution of the unit j in the hidden layer when the visible layer is fixed.

If the values of the units in the visible layer and the hidden layer are all binary, then

$p(v_i = 1|H; \Theta, A)$ and $p(h_j = 1|V; \Theta, A)$ are given as follows.

$$p(v_i = 1|H; \Theta, A) = f(\sum_j w_{ij} h_j + a_i) \tag{2.21}$$

$$p(h_j = 1|V; \Theta, A) = f(\sum_i w_{ij} v_i + b_j) \tag{2.22}$$

where $f(x)$ represents the activation activation function.

Since the values of the DBA's input units representing the numbers of inbound packets are continuous and affected by many factors, we use the Gaussian probability distribution to model the traffic patterns [48]. Therefore, for RBM_1 in our proposed DBA, Equation 2.16 and Equation 2.21 should be revised as follows.

$$E(V, H) = -\sum_i \frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_j b_j h_j - \sum_i \sum_j \frac{v_i}{\sigma_i} h_j w_{ij}, \tag{2.23}$$

$$p(v_i|H; \Theta, A) = N(a_i + \sigma_i \sum_j h_j w_{ij}, \sigma_i^2), \tag{2.24}$$

where σ_i is the value of the variance for the unit v_i. $N(a_i + \sigma_i \sum_j h_j w_{ij}, \sigma_i^2)$ denotes the Gaussian distribution with mean $(a_i + \sigma_i \sum_j h_j w_{ij})$ and variance σ_i.

The forward propagation follows the above-mentioned equations for the DBA. However, if the DBA is trained with the supervised learning manner, since the output layer has a given label, the last RBM, RBM_{L-2}, consists of three layers as shown in Fig. 2.3c [49]. Therefore, the visible layer of RBM_{L-2} consists of not only RBM_{L-3}'s hidden layer but also the output layer of the DBA, Y. And its hidden layer is the top hidden layer of the DBA. The structure of RBM_{L-2} is shown in Fig. 2.3c and its energy function is expressed as follows. To keep consistent with the other RBMs, we use V and H to denote RBM_{L-3}'s hidden layer and the top hidden layer, respectively.

$$E(V, H, Y) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_k c_k y_k - \sum_i \sum_j h_j w_{ij} v_i - \sum_j \sum_k h_j w_{kj} y_k, \tag{2.25}$$

where Y represents the vector in the output layer. c_k is the bias of the unit y_k. w_{kj} represents the weight of the link connecting the units h_j and y_k.

As the units in V and Y are independent on each other, the conditional distribution

of the concatenated vector consisting of V and Y is,

$$\begin{aligned} p(V, Y|H; \Theta, A) &= p(V|H; \Theta, A)p(y|H; \Theta, A) \\ &= \prod_i p(v_i|H; \Theta, A) \prod_k p(y_k|H; \Theta, A) \end{aligned} \tag{2.26}$$

We use the method mentioned above to train each RBM. The value of the visible layer of the first RBM is X in the given training data. And after training each RBM, the learned activated value of its hidden layer is used as the "data" for the next RBM in the DBA. Here, it can be found that we train one hidden layer of the DBA at one time via training an RBM. In this way, the DBA gets initialized and the value of Θ is nearly optimum. Then the method of backward propagation described in Sec. 2.2.1 is utilized to fine-tune the DBA.

2.2.2.2 Convolutional Neural Networks

CNNs have achieved the most wide applications in many fields, to name a few, pattern recognition, image processing, video analysis, and natural language processing [17]. As shown in Fig. 2.2c, a CNN is composed of several convolutional layers, pooling layers, and fully connected layers [50]. Different from the Deep Neural Network (DNN), the units in the convolutional layer are just connected to part of the units in next layer, which can significantly reduce the number of parameters. The pooling operation is usually located in-between two successive convolutional layers. In the pooling operation, the result of the convolution operation is sampled to progressively reduce the feature size [51].

Fig. 2.2c depicts the structure of a CNN, which mainly consists of three parts: convolutional layers, pooling layers, and fully connected layers. For the purpose of classification, a softmax regression process is usually conducted on the output of the last fully connected layer to get the final output [52, 53]. In the remainder of this section, we will give a detailed introduction on the mathematical formulation of each of these layers.

To describe the calculations more clearly, we consider two layers labeled as the $(l-1)^{th}$ and l^{th} layers as an example to explain every operation. The input and output of the convolution and pooling operations are usually represented by a three dimensional matrix, while those of the fully connected layers and softmax operation are denoted as vectors. Moreover, as an activation operation exists in most layers of the CNNs, we adopt Z and U to denote the values of each layer before and after activation, respectively.

For the convolution operation, it is reasonable to assume that the dimension of $U^{(l_1-1)}$ (also $Z^{(l_1-1)}$) is $P \times Q \times R$. Since the CNNs are usually utilized in the image recognition field, we can consider the three dimensions of the input matrix to denote an image's height, width, and depth, denoted by P, Q, R, respectively [52]. For instance, in the application

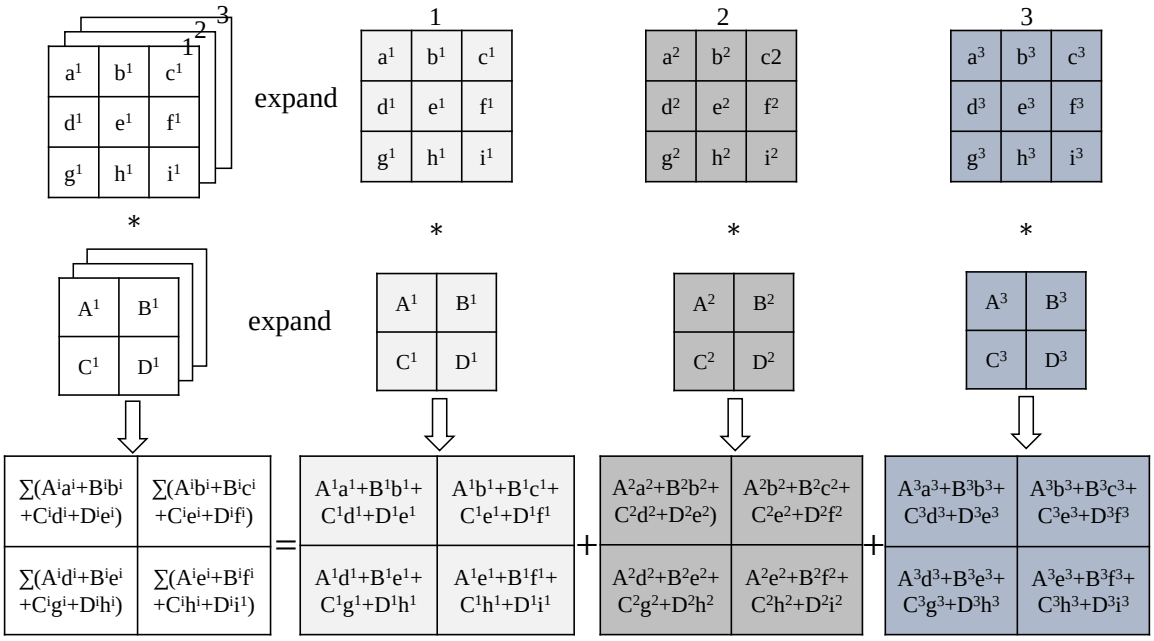


Figure 2.4: The process of convolution between two three-dimensional matrices.

of image classification, researchers usually use a $28 \times 28 \times 3$ matrix to denote a picture, in which 28 is the number of pixels in every edge and 3 is the number of basic color channels which are R, G, and B, respectively. As the purpose of the convolution operation is to extract the distinguished features of the input, the parameters (weights and biases) of the convolution operation consist of a set of learnable filters, each of which can be denoted by a $P^{'} \times Q^{'} \times R$ matrix as the depth of each filter should be the same as that of the input volume. Every filter is small spatially, but extends through the full depth of the input volume. During the forward pass, each filter slides across the width and height of the input volume and a convolution operation is conducted between the filter and the area of the input volume covered by the filter which is shown in Fig. 2.4. The convolution result of the input volume and each filter is named as a feature map. If we use $W^{(l_1)}$ to denote the filters and the k^{th} filter is represented by $W_k^{(l_1)}$, the obtained feature map by the convolution operation can be shown as follows.

$$\begin{aligned} z_{i,j,k}^{(l_1)} &= (Z^{(l_1-1)} * W_k^{(l_1)})(i,j) + w_{bk}^{(l_1)} \\ &= \sum_{r=1}^{R} \sum_{p=1}^{P^{'}} \sum_{q=1}^{Q^{'}} w_{p,q,r} u_{i+p,j+q,r}^{(l-1)} + w_{bk}^{(l_1)}, \end{aligned} \tag{2.27}$$

$$a_{i,j,k}^{(l_1)} = f(u_{i,j,k}^{(l_1)}), \tag{2.28}$$

where $f(\cdot)$ is the activation function and $u_{i,j,k}^{(l_1)}$ is the activated value of the unit in the i^{th}

row and j^{th} column of the feature map. Therefore, $z_{i,j,k}^{(l_1)}$ is the value before activation. $w_{bk}^{(l_1)}$ denotes the bias of the k^{th} filter and is usually a single numeric value. $u_{i+p,j+q,d}^{(l_1-1)}$ is the activated value of unit in the $(i+p)^{th}$ row and $(j+q)^{th}$ column. The most commonly used activation function is the Rectified Linear Unit (ReLU) function given in Equation 2.4.

After the convolution operation, the convolutional layer consists of several feature maps. Since the convolution operation between the input volume and each filter results in a feature map, the number of feature maps is equal to that of filters. Moreover, the size of each feature map is dependent on the following factors: the sizes of the input volume and each filter, the stride of each slide, and the number of zero-padding columns or rows. If we use s and t to represent the number of zero-padding and the value of each slide, then the width and height of the feature are as in Equations 2.29 and 2.30. Here, it should be noted that the advantages of zero-padding are two-fold, namely, to sufficiently extract the features located in the edge of the input volume, and make the height and width of each feature map to be integers.

$$w_{fm}^{(l_1)} = (P - P^{'} + 2s)/t + 1, \tag{2.29}$$

$$h_{fm}^{(l_1)} = (Q - Q^{'} + 2s)/t + 1. \tag{2.30}$$

To progressively reduce the spatial size of the representation to reduce the amount of parameters and computation, it is common to periodically insert a pooling layer in-between the successive convolutional layers. The pooling layer is the result of the down-sampling of the convolutionle layer. To make the pooling operation learnable, we usually add two scalar parameters and the mathematical expression is shown in Equation 2.31.

$$Z^{(l_2)} = \beta^{(l_2)} down(U^{(l_2-1)}) + b^{(l_2)}, \tag{2.31}$$

$$U^{(l_2)} = f(Z^{(l_2)}), \tag{2.32}$$

where $\beta^{(l_2)}$ and $b^{(l_2)}$ denote the learnable parameters of the pooling operation. $Z^{(l_2)}$ and $U^{(l_2)}$ denote the values of output matrix of the pooling layer before and after activation, respectively. *down* represents the operation of downsampling. There are many types of downsampling methods, among which the most popular one is the max pooling, which just keeps the maximum activation value in every square region of the $(l_2 - 1)^{th}$ layer. In this way, the height and width of the pooling layer are just half of those of the $(l_2 - 1)^{th}$ layer while their depths are the same.

After several convolutional layers and pooling layers, the size of the input volume can be significantly reduced. Then, the final pooling layer is connected to the fully connected

layer, in which every unit is connected to all units in the previous layer. If we assume the $(l_3-1)^{th}$ and l_3^{th} layers are the pooling layer and fully connected layer, respectively, the units in these two layers satisfy the following relationship.

$$z_i^{(l_3)} = \sum_j u_j^{(l_3-1)} w_{ij}^{(l_3)} + b_i^{(l_3)}, \tag{2.33}$$

$$u_i^{(l_3)} = f(z_i^{(l_3)}), \tag{2.34}$$

where $z_i^{(l_3)}$ and $u_i^{(l_3)}$ are the values of unit i in the l_3^{th} layer before and after activation, respectively. As mentioned earlier, the 3-dimensional matrix in the pooling layer needs to be spread into a vector when connected to a fully connected layer. Then, $u_j^{(l_3-1)}$ is the activated value of unit j in the vector. $w_{ij}^{(l_3)}$ is the weight of the connection between unit i in the l_3^{th} layer and unit j in the $(l_3-1)^{th}$ layer. $b_i^{(l_3)}$ is the bias of unit i in the l_3^{th} layer. If there are multiple fully connected layers, the relationship between any two fully connected layers also satisfies Equations 2.33 and 2.34.

As the CNN is usually used for classification, if we assume that there are t kinds of different results, then we can use a vector consisting of t binary values to represent the result [53]. And in the vector, only one unit must have the value of 1, the order of which represents the result. Therefore, a softmax regression process is necessary to be conducted on the final fully connected layer. The first step of softmax regression is to convert the values of units in the fully connected layer into possibilities of the results according to Equation 2.35. Then if p_i is maximum, we can label unit i in the output layer as 1 while each of the other units is labeled as 0. We can adopt y to denote the order of the unit, which is labeled 1, meaning that the result of the CNN is y.

$$p_i(x) = \frac{e^{z_i}}{\sum_j e^{z_j}}, \tag{2.35}$$

where p_i is the possibility of the appearance of result i. And z_i is the value of unit i in the final fully connected layer.

If the CNN is trained in a supervised manner, the purpose of the training is to maximize the possibility of training data, which can be expressed as a loss function as shown in Eq. (2.36) [52].

$$C(W,B) = -\frac{1}{N}\left(\sum_{i=1}^{N}\sum_{j=1}^{k} 1\{y=j\}\log p_i(x_i)\right), \tag{2.36}$$

where W and B represent the weights and biases of the CNN. N is the number of training data. x_i is the i^{th} input training data. Here, $1\{\cdot\}$ is the indicator function and its values satisfy that $1\{a\ true\ statement\} = 1$ and $1\{a\ false\ statement\} = 0$.

Table 2.1: Comparison of three training methodologies.

Methodology	Definition	Purpose
Supervised learning	The machine learning task of inferring a function from an already available labeled training dataset usually specified by human operators.	Approximating the function between the given input and output.
Unsupervised learning	The machine learning task of inferring a function to describe hidden structure from "unlabeled" data.	Clustering the given input to different groups according to their distribution.
Reinforcement learning	The machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward.	Exploring and exploiting solutions with a specific objective.

2.2.3 Different Training Manners

After discussing the two commonly utilized deep learning architectures: DBA and CNN, we introduce the training methods in this part. As one of the machine learning methods, deep learning also has three training manners as shown in Table 2.1: supervised learning, unsupervised learning, and reinforcement learning.

Supervised learning, as shown in Table 2.1, is a kind of machine learning task of learning a function to map an input to an output based on the given input-output examples. Therefore, the training data should consist of the input X and corresponding labels, Y. And the goal of training is to predict Y as accurately as possible. Supervised learning is usually adopted for linear regression or classification. For linear regression, it attempts to model the relationship between X and Y by fitting a linear equation to observed data and the output Y is continuous numerical value. And the classification is the problem of assigning new observations to the class to which they most likely belong, based on a classification model built from labeled training data. Therefore, the classification is to predict a discrete label Y.

Different from the supervised learning method, the unsupervised learning is a kind of machine learning method that learns from the data that have not been labeled, classified, or categorized. Therefore, no correct answer exists in the input data for the unsupervised learning. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. According to the purpose, the unsupervised learning tasks can be further divided into clustering and association. The clustering problem is where you want to group the input data according to their inherent distributions, while the association is to discover the rules that describe large portions of your data.

Alongside supervised learning and unsupervised learning, reinforcement learning is

Table 2.2: Existing networking research based on deep learning

Purpose	Strategy	Architecture	Training manner
Network prediction	Traffic prediction	CNN or LSTM	Supervised learning
Resource allocation	Cache allocation	Deep Q-network	Reinforcement learning
	Channel assignment	DBA & CNN	Supervised learning
Security	Anomaly detection	LSTM	Unsupervised learning

considered as one of the three machine learning paradigms. It is concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward as shown in Table 2.1. The reinforcement learning differs from the supervised learning in that the correct input/output pairs need not to be presented. Moreover, the agent focuses on the final cumulative rewards, meaning that the suboptimal actions need not to be explicitly corrected. Furthermore, at every step, the software agent has two options: take the action most of the time which generates the maximum reward or occasionally explore a new action even though it is walking away from known reward. Therefore, the reinforcement learning is involved in finding a balance between the exploration (of uncharted territory) and exploitation (of current knowledge).

2.2.4 Survey of Deep Learning Based Networking

According to our introduction on the various deep learning architectures and three training manners, it can be easily found that this technique is very flexible to be utilized in different scenarios. The significant performance improvement generated by the breakthrough further increases the perspectives of adopting deep learning to solve complex problems. Networking researchers have also attempted to consider this technique to optimize the performance as shown in Table 2.2. In this part, we will give some introductions about the related research of deep learning based networking.

2.2.4.1 Network Parameter Prediction

As deep learning is efficient in prediction, it is the first idea to adopt this technique to forecast the network parameters. If we can get the predicted values of some parameters, we can adjust the network management in advance to optimize the performance. For instance, if we can predict how the topology of the mobile network changes, the channel resource can be prepared in advance for the new connections. Since the traffic flow is the most direct sign of the network condition, it has aroused the most attention of researchers to predict the network traffic with the deep learning technique [54, 55]. Authors of [54] considers the traffic data of the citywide cellular networks as images. Then, the CNNs can be utilized to extract the spatial and temporal dependence of cell traffic. Experimental

results show that the prediction performance in terms of root mean square error can be significantly improved compared with conventional machine learning methods. J. Zhao et al. utilizes the LSTM to model the spatio-temporal features of network traffic and proposed a linear regression model to predict the traffic flow [55]. Simulation results evaluate the performance of their proposal. It can be found that the network traffic can be predicted with different deep learning architectures. Moreover, compared with conventional method, the deep learning technique has much higher prediction accuracy.

2.2.4.2 Intelligent Resource Allocation

Resource allocation is an important factor which directly impacts on the network performance. In current practical networks, the network resource allocation is usually concerned with two types of resource: the communication resource including channels and bandwidth, and the computation resource consisting of the processing power and memory. The authors of [56] discussed the balance between two conflicting factors: the network cost and users' Mean Opinion Score (MOS) to improve the network Quality of Experience (QoE). In this paper, authors utilized the deep reinforcement learning technique to change the cache location of content. Specifically, the network states which consist of the transmission rates and cache condition are utilized as the input of the deep Q-networks [57]. Then, the agent can choose the best action according to the output Q-values of different actions. Simulation results demonstrate the agent can find the best decision to maximize the proposed reward function after trial and error. As we mentioned above, the traffic prediction is usually conducted for many network management tasks. Our previous work [8] proposed a deep learning based partially overlapped channel assignment strategy for the IoT network. In the proposal, the DBA and CNN are jointly utilized to predict the suitable channel assignment for each link according to the traffic patterns. Since the input considers the properties of the IoT traffic, the prediction accuracy is much higher than conventional methods, which leads to the performance improvement of channel assignment.

2.2.4.3 Smart Anomaly Detection

The IoT has been regarded as one of the paradigms of next generation network future IoT network. Since the IoT traffic is concerned with users' data, it is critically important to increase the security and privacy level of corresponding network services. In existing ground networks, deep learning has also been studied for improving the network security level. In [58], the authors proposed a deep learning based approach to detect anomalous network activity. In the paper, the DNN is adopted to extract the features of users' activities from the system logs. Then, the feature vectors are input to the LSTM as

shown in Fig. 2.2b to measure the anomaly score. Since the users' activities are often unpredictable over seconds to hours, authors in this paper utilized an online unsupervised training fashion, by which the models can adapt to the changing patterns in the data. Simulation results show that the proposed strategy has a very high accuracy rate and can significantly reduce the analyst workloads.

2.3 Overview of Traffic Control

Besides the preliminary knowledge of deep learning, we also need to discuss the basis of traffic control before conducting the related research. We will introduce the conventional traffic control strategies in different layers. The main concept and the corresponding shortcoming will be analyzed. To improve the traffic control performance, the deep learning based traffic control is proposed in this section. We mainly discuss what we should study if adopting this technique for traffic control.

2.3.1 Traditional Traffic Control Strategies

Since the network was constructed, the global traffic overhead has been increasing over forty years. To avoid the network congestion and reduce the end-to-end delay, the industry and academia have devoted various endeavors focusing on different layers [26, 59, 60, 61]. Among these traffic control strategies, the most efficient and obvious manner is to deploy the new generation of infrastructures with more computation and communication capacities [26]. Following the Moore's Law, the network hardware including the routers, switches, and data centers, has experienced the changes of several generations. Another example which impacts our life more clearly is the development of cellular communications. As the most commonly used communication form which is developing towards its fifth generation, the transmission speed has been increased to 1Gbps in nearly 40 years [3]. And compared with the first generation which can only provide the wireless voice service, current 4G technique can offer users fluent Internet services, such as the high-definition mobile TV, gaming, and IP telephony [62]. Moreover, the emerging 5G technology will meet the needs of new use-cases such as the IoT and autonomous vehicles [4]. It can be found that the development of hardware can meet the traffic demand of new services. However, the evolution of the physical layer usually has a long cycle and extreme high expense. To address this problem, researchers have also proposed many strategies from different layers to alleviate the network congestion. In following paragraphs, we will introduce some existing strategies in the data link layer, network layer, and transport layer.

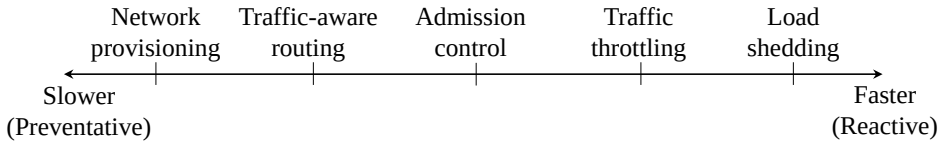


Figure 2.5: The timescales of approaches to congestion control.

2.3.1.1 Data Link Layer

In the data link layer, the design of traffic control is to consider what to do with a sender that systematically wants to transmit frames faster than the receiver can accept them. A common situation in practical networks is when a smartphone requests a service from a far more powerful server. Even if the transmission is error free, the smartphone may not be able to handle the packets sent by the server in time and then lose some. The common strategies to solve this problem can be divided into two groups: the feedback-based schemes and rate-based schemes [63]. In the first one, the feedback-based traffic control, the receiver sends back to the sender some information which can be the permission of more frames or some transmission rules. And the sender sends the frames according to the feedback. This principle rule is followed by various feed-back based traffic control schemes. In the rate-based schemes, the protocol has a built-in mechanism which limits the rate at which the senders may transmit data without utilizing the feedback from the receiver. In existing networks, the rate-based traffic control strategies are regarded as part of the transport layer [63].

2.3.1.2 Network Layer

Besides the data link layer, the network layer also shares the responsibility of traffic congestion avoidance. Since the congestion happens within the network, the network layer directly experiences the performance deterioration. In the network layer, various strategies have been proposed to alleviate and balance the traffic overhead. These strategies consist of network provisioning, traffic-aware routing, admission control, traffic throttling, and load shedding, which are applied on different time scales to either avoid the congestion or react to it once it happens as shown in Fig. 2.5. For each method, some simple explanations are given in the following paragraph.

The method of network provision is to consider some extra resource including routers and switches as backup for dynamical assignment when necessary [63]. Even though this method can effectively alleviate the congestion, it needs to be prepared before constructing the network. The second method, traffic-aware routing is to optimize the path design for traffic balance [64]. It can be fulfilled via different manners, such as splitting traffic across multiple paths [65], choosing the traffic overhead as the link weight [64], or predicting the traffic changes to avoid the heavily used link [66]. Furthermore, once traffic congestion

Table 2.3: Some congestion control protocols in the transport layer.

Protocol	Signal	Precise
XCP	Rate to use	Yes
TCP with ECN	Congestion warning	No
FAST TCP	End-to-end delay	Yes
Compound TCP	Packet loss and end-to-end delay	Yes
CUBIC TCP	Packet loss	No
TCP	Packet loss	No

occurs, the admission control and load shedding methods can be applied. And these two methods have similar ideas, one is to refuse new connections [59] while the other one is to drop some traffic [67]. The traffic throttling is similar to the feedback-based scheme in the data link layer, by which the senders also adopt the feedback to adjust their transmissions [68]. These methods have been widely considered in current practical networks.

2.3.1.3 Transport Layer

Since the congestion is ultimately caused by the traffic sent into the network from the transport layer, the traffic control is also the responsibility of this layer. In current practical networks, various strategies have been adopted to control the traffic in the transport layer. And these strategies can be divided into two groups: bandwidth allocation and regulating the sending rate [63]. The first one is usually fulfilled by running an efficient allocation algorithm to find a good bandwidth assignment to the transport entities that are using the network. And the fairness as well as the network delay and throughput are also considered in the algorithm. In the second group, similar to the feedback-based traffic control schemes in the data link layer, the traffic control protocols utilize some metrics as congestion signals. And once the sender judges that the congestion occurs, it slows down the packet sending rate. How much to slow down can be set a definite value or decided according to the values of the considered congestion signal. Table 2.3 gives several TCP traffic control schemes [69, 70, 71].

2.3.2 Research on Deep Learning Based Traffic Control

After introducing the existing traffic control strategies, we can clearly find that it is the common responsibility of data link layer, network layer, and the transport layer to avoid the network congestion. And these strategies can be conducted at different layers to improve the traffic control performance. However, as the global networks become increasingly complex, the existing strategies need to be improved to fit for the new scenarios. For

example, the performance of traffic aware routing method depends the accuracy of the traffic prediction. Considering the growing heterogeneity of current networks, we need to adopt more efficient traffic prediction technique. Moreover, the rate-based traffic control schemes should also take into account the different service requirements. To start our research, we first focus on the network layer and adopt the deep learning technique for the routing design to alleviate the traffic overhead. Then, the following aspects need to be studied.

2.3.2.1 Network Scenarios and Problem Analysis

As we mentioned earlier, there exist different network scenarios offering various services. It is not realistic to propose only one algorithm utilizing the deep learning technique for improving traffic control for all the networks. Therefore, we need to focus on definite network scenarios and analyze their characteristics, which can impact on our following problem formulation and the deep learning structure construction [24]. For example, for the fiber network, the link information may be neglected due to the large bandwidth. On the other hand, as the D2D networks have dynamic links with limited bandwidth, the link information must be considered in our research.

2.3.2.2 Deep Learning Structure Construction

After analyzing the considered problems for definite scenarios, we can study the construction of deep learning architectures [24]. Firstly, we can characterize the input and output of the deep learning architecture according to our purpose. Since we want to improve the path design method for the purpose of traffic control, the traffic pattern and the next node can be taken as the input and output, respectively. This is because the traffic pattern is the most direct sign of the network situation. Moreover, if we consider the dynamic networks, then the network topology as well as the node information should be considered. The characterizations of the input and output should be firstly considered based on our purpose as shown in Fig. 2.1. Then, we can utilize the input and output to choose a suitable deep learning architecture. For example, the DBA can be chosen if the input is a vector, while the CNN needs to be considered for the matrix input. Moreover, if we want to utilize the deep learning to predict a sequence, the LSTM may be the best structure.

2.3.2.3 Network Performance Analysis

As our goal is to apply the deep learning for traffic control, we need to consider the simulation or experiment to analyze the performance of our proposal. Since the simulation is more efficient and adjustable than the experiment, we conduct simulations to evaluate

the performance. And in the simulation, we utilize the network throughput, average delay, and packet loss rate as the metrics to measure the traffic control performance. To illustrate the improvement more clearly, we utilize some conventional routing methods as the benchmark, such as the Open Shortest Path First (OSPF) protocol [72].

2.3.2.4 Computation Analysis and Proposal Deployment

As deep learning is concerned with massive matrix computations, it generates more computation overhead compared with conventional methods. Also, since the prediction accuracy significantly affects the network performance, to improve the training accuracy is very important for the traffic control. Therefore, in our research, besides the deep learning structure construction, we also need to optimize the considered architectures and training methods. The computation complexity should be studied to analyze the practical deployment [73]. This is because the conventional hardware based communication infrastructure is not suitable to execute the deep learning based proposals [9]. Therefore, we need to consider the suitable hardware platform to efficiently run the proposed strategies.

2.4 Summary

In this chapter, we introduce the preliminary knowledge of deep learning including the propagation process, two deep learning architectures, and three training manners. Moreover, the survey on current deep learning based networking research evaluates the perspectives of this emerging AI technique. To adopt the deep learning for more efficient traffic control, we analyze the existing strategies in different layers. These strategies have been widely applied in practical networks. However, as the global networks have been growing complex, existing traffic control strategies also need to be improved. Therefore, we consider the deep learning technique and give some introductions about what we need to study if adopting deep learning for traffic control.

Chapter 3

Deep Learning Based Routing Algorithm for Core Networks Running on GPU Accelerate SDRs

3.1 Introduction

To adopt the deep learning technique for improving the traffic control, we first need to choose a network scenario in this chapter. Since the routing is concerned with packet forwarding in the Internet, we choose the static backbone network as our considered scenario. As we mentioned in Chapter 1, the proprietary hardware architectures lack the flexibility for the potential update of management strategies. To apply the state-of-the-art software driven routing algorithms developed for different network services, it is necessary to improve the programmability of the core routers. Moreover, since the deep learning technique is concerned with massive matrix computations, we consider the GPU-accelerated SDRs as the routing architecture. Specifically, we adopt the collected data comprising inbound traffic patterns and corresponding subsequent nodes (i.e., routers) to train the DBAs in a supervised manner. Then, the trained DBAs [18] can compute the subsequent nodes (i.e., routers) with the traffic patterns of the edge routers as the input. And the deep learning related training and running operations are executed by the GPU-accelerated SDRs. Furthermore, the packet forwarding operations as well as the processing work are cooperatively conducted by the GPUs and CPUs.

The remainder of the chapter is structured as follows. And in Sec. 3.2, we delineate our proposed DBA structure for routing and how it works in the GPU-accelerated SDR. Then, we introduce the three phases of the deep learning based routing strategy in Sec. 3.3 and Sec. 3.4 analyzes the complexity of our proposal and compares the theoretical time cost for a GPU and a CPU. The network performance evaluation of our proposal is presented

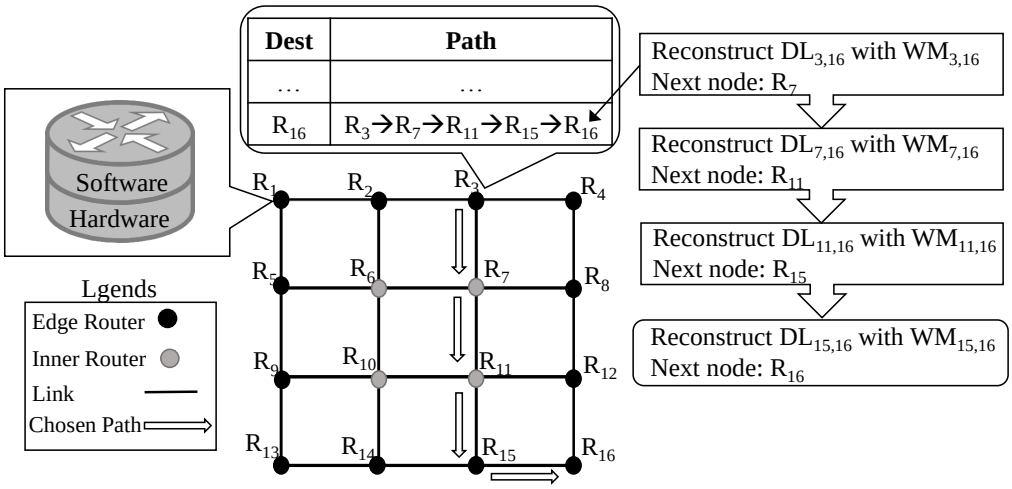


Figure 3.1: Considered system model and problem statement.

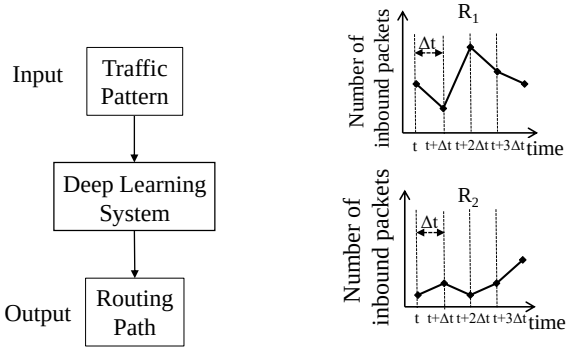
in Sec. 3.5. Finally, Sec. 3.6 concludes the article.

3.2 Design of Deep Learning based Routing Strategy

In this section, we introduce how to design the deep learning structure to construct the routing table on a GPU-accelerated SDR. First, we present the detailed characterization of the input and output of the deep learning structure, then we describe our chosen architecture, DBA. Next, how the proposed routing table construction method works on a GPU-accelerated SDR is discussed.

3.2.1 Input and Output Design

Our considered core network system model is depicted in Fig. 3.1 comprising a number of wired backbone routers. It is worth noting that a wireless backbone network may also be considered. In the considered network, the edge routers are assumed to be connected to different types of networks such as cellular networks, Wireless Mesh Networks (WMNs), and so forth. The data packets generated from the latter networks arrive at the edge routers and are destined for other edge routers for delivery. On the other hand, the inner routers are just responsible for forwarding the packets to the appropriate edge routers. Traditionally, each router periodically forwards the signaling packets to other routers to inform the values of delay or some other metrics of its links to its neighbors. Then, every router can utilize the information to compute the next nodes for sending data packets to the destination routers. This method works well in most cases since every router can make the best decision according to the obtained information of all the network links. However, when some routers in the network are congested because of the overwhelming traffic demand, conventional methods to compute the next nodes suffer from slow convergence.



(a) Characterized input and output.

(b) Considering traffic patterns at each router as input.

Figure 3.2: Considered input and output design.

At the same time, the periodical signaling exchange aggravates the traffic congestion. Furthermore, the traditional routing methods are unable to deal with scenarios where the network environment continues to become more complex, which requires the network operators to consider various unrelated parameters to determine the routing rules. As the deep learning method has been applied to many complex activities to automatically explore the relationships among various inputs, we attempt to adopt deep learning for routing in the remainder of this section.

Since the traffic pattern observed at each router is a direct indication of the traffic situation of that router, we adopt traffic patterns as the input of our deep learning model. As mentioned in Sec. 3.1, the deep learning structure is utilized to compute the routing path. Therefore, we choose the routing path as the output of the model. Accordingly, Fig. 3.2a demonstrates that the traffic pattern is served as the input to the deep learning structure and processed for the routing path decision as the output. Then, the key challenge is to characterize the input and output of the deep learning structure. In order to characterize the input, we use the traffic pattern at each router that may be defined as the number of inbound packets of the router during each time interval as shown in Fig. 3.2b. If we assume that the time interval to count the inbound packets is Δt seconds, then for each router, we can adopt the number of inbound packets in each time interval during the last $\beta\Delta t$ (β is a positive integer) seconds as its traffic pattern. Therefore, by assuming that a network comprises of N routers, we can use a matrix of β rows and N columns to represent the traffic patterns of all the routers in the network and input the values of βN elements in the matrix to the input layer of the deep learning structure. Note that the value of β should not be too large as the traffic patterns long time ago make no difference for current network analysis. Furthermore, if the value of β is too large, the deep learning structure suffers from high complexity and low efficiency. Here, in our

proposed deep learning structure, the simulation results demonstrate that it is accurate enough to set the value of β to 1. As a result, the input of the deep learning structure can be seen as a N dimensional vector, whose i^{th} element is the i^{th} router's traffic pattern during the last Δt seconds. Next, we need to design the output layer. For the purpose of routing, the deep learning structure needs to output the routing path. Consequently, the output layer can be designed to give the whole path like the centralized routing or only the next node similar to the distributed routing strategy. The latter is chosen in our proposal due to its lower complexity and higher tolerance. For a network consisting of N routers, we use a vector consisting of N binary elements to represent the output. In the vector, only a single element has the value of 1, the order of which represents the next node. This means that if the i^{th} element in the N dimensional vector is 1, then the i^{th} router in the considered network is chosen as the next node. In summary, we can use two N dimensional vectors, X and Y, to represent the input and output of the deep learning structure and an example of X and Y is given as follows:

$$X = (tp_1, tp_2, ..., tp_{N-1}, tp_N), \tag{3.1}$$

$$Y = (0, 1, ..., 0, 0), \tag{3.2}$$

where tp_i represents the traffic pattern of the router i which is measured by the number of inbound packets in last time interval. Furthermore, in vector Y, we can find that $y_2 = 1$, which implies that the router 2 is chosen as the next node. Due to the binary value of Y, the deep learning structure is a logistic regression model, which we need to design next.

3.2.2 Deep Learning Structure Design

We utilize the DBA described in Sec. 2.3a as our considered deep learning architectures. Since our purpose of supervised training is to minimize the difference between the output of the DBA (denoted by $h_\Theta(X)$) and the labeled output Y, we use the cross-entropy cost function to measure their difference given in Equation 3.3 [74].

$$C(\Theta) = -\frac{1}{m}\sum_{t=1}^{m}(Y^{(t)}\log\left(h_\Theta(X^{(t)})\right) + (1-Y^{(t)})\log\left(1-h_\Theta(X^{(t)})\right)) + \frac{\lambda}{2}\sum_{l=2}^{L}\sum_{j=1}^{n_l}\sum_{i=1}^{n_{l-1}}(w_{ij}^{(l)})^2. \tag{3.3}$$

Here, $(X^{(t)}, Y^{(t)})$ is the t^{th} training data. $h_\Theta(X^{(t)})$ denotes the output of the DBA when the parameter of the DBA is Θ and the input is $X^{(t)}$. On the right side of the equation, we can find $C(\Theta)$ consists of two parts. The first part represents the difference between the

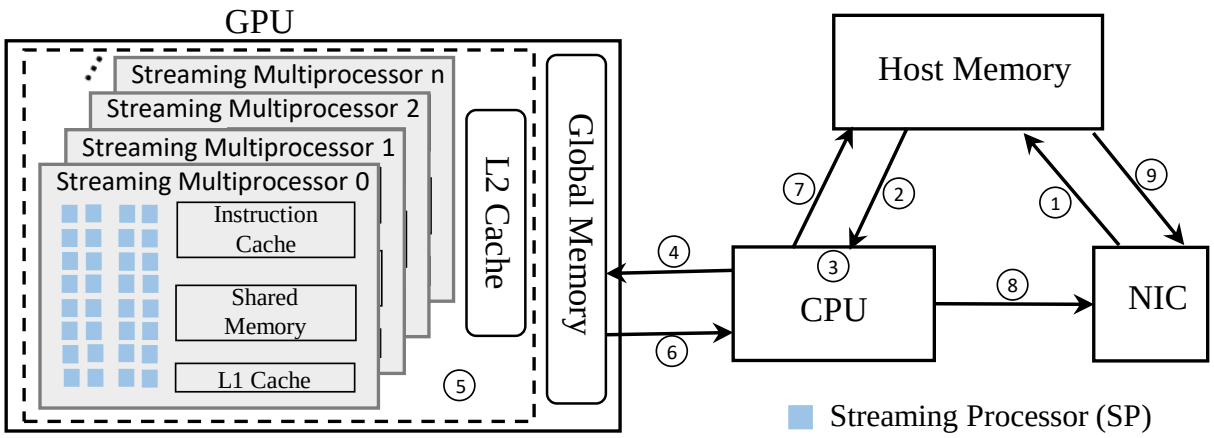


Figure 3.3: The architecture of GPUs and steps of how packets are passed in the GPU-accelerated SDR.

output of DBA and the labeled output, and its value is 0 when $Y^{(t)} = h_\Theta(X^{(t)}) = 0$ *or* 1 for all t, otherwise, bigger than 0. The second part is used to keep the training process from overfitting.

The fine-tuning algorithm works effectively since the value of Θ gets well-initialized through the Greedy Layer-Wise training method instead of being randomly set. After fine-tuning the DBA, we can obtain the optimal values of the parameter $\Theta(W, B)$. The value of A is not trained in the backward propagation step since it does not belong to the final DBA and is just useful in the training of every RBM. In the remainder of the section, we demonstrate how the proposed deep learning structure can be used in a GPU-accelerated SDR.

3.2.3 Considered Router Architecture

In this section, we give a short introduction to the GPU architecture and the procedures of the proposed deep learning based routing strategy working on a general PC platform, which can be regarded as our considered SDR.

As shown in Fig. 3.3, a GPU consists of the global memory, L2 cache, and several Streaming Multiprocessors (SMs), each of which is composed of many Streaming Processors (SPs) [14]. Since a GPU has many computing cores, it launches tens of thousands of threads concurrently when receiving a workload, and each thread runs the same program but on the different set of data. Therefore, the GPU computing is considered as a Single Instruction Multiple Data (SIMD) programming model which is very suitable for running the deep learning.

As mentioned in Sec. 3.1, the reported line rate of the GPU accelerated SDR based on a common PC has reached as high as 40 Gbps, in this chapter, we choose a general PC-based SDR to construct the routing tables and execute our deep learning based rout-

ing algorithm. Fig. 3.3 shows the steps of how the packets are passed through the related four parts in the architecture of the considered SDR, i.e., a GPU, a CPU, Network Interface Cards (NICs), and the main memory. For running the deep learning based routing algorithm, every SDR needs to be initialized in the training phase, during which, SDRs in the network do not need to process any packet and just utilize their GPUs to train their DBAs and record the final values of the parameters of their DBAs. After the training phase, all the routers in the network need to send the parameters' values of their DBAs to all the edge routers. Therefore, every edge router can use the parameters to restore any DBA for building the whole path to any destination router in the running phase, while the inner routers just forward the packets according to the path. As shown in Fig. 3.3, we have given the main architecture of the GPU-accelerated SDR and the labels according to the orders the packets are transferred in the SDR. We can find that (1) packets entering the NIC are copied to the host memory through the Direct Memory Access (DMA). During the whole process, (2) the CPU copies some packets from the main memory to fill its buffer. (3) Then software running on the CPU analyzes these packets and takes some necessary processes like error checking, lifetime reducing, and so on. Moreover, the CPU takes different processes for different types of packets. (4) For data packets, the CPU extracts the headers and sends them to the global memory of the GPU, while the CPU sends the whole signaling packets to the GPU's memory. Note that the CPU needs to buffer the headers of data packets and signaling packets until reaching a given size, and then sends the batch of headers or packets to the GPU instead of dispatching them one by one. Since the GPU can process hundreds of packets in parallel, the batch-processing can improve the throughput, while its adverse effect on latency has been proved to be negligible [27]. (5) After obtaining headers and packets from the CPU, it should be noted that GPUs of edge routers and inner routers execute different packet processing. Software running on the GPUs of the edge routers uses the traffic patterns carried by the signaling packets as the input of the restored DBAs. Then, the DBAs can output the next nodes, with which the GPUs of the edge routers can build the whole paths for data packets and attach the corresponding paths to the received headers. Additionally, the GPUs also need to send the next node information of each packet to the CPUs. On the other hand, the GPUs of the inner routers do not need to compute the paths for the packets and just read the paths in the packets' headers and send the results to the CPUs. Additionally, each GPU processes these headers in parallel and fills them in the buffer. Then, the CPU (6) copies back the processed data packets' headers from the GPU and (7) copies the packets back to the main memory. Meanwhile, (8) the CPU instructs the NIC where to forward the batch, after which, (9) the NIC fetches the packets from the main memory through another DMA. Furthermore, the processes of copying packets to and from the GPU can be deleted since we can take advantage of the mapped memory of the GPU and the CPU,

by which the latency can be further reduced.

In this section, we provided our considered system model, deep learning structure, and explained how the GPU-based SDR can exploit the deep learning structure. In the following section, we present the steps of our proposed deep learning based routing algorithm.

3.3 The Procedures of the Proposed Deep Learning based Routing Strategy

In this section, we focus on the procedures of utilizing the DBAs to compute the next nodes for building the routing paths in the considered core network in Fig. 3.1. The procedures can be divided into three steps, i.e., initialization, training, and running phases. The details of the three phases are provided below.

3.3.1 Initialization Phase

In the initialization phase, we need to obtain the data to train our proposed DBAs. As described in Sec. 3.2, we adopt the supervised learning to train our proposed DBA systems. Therefore, the goal of the initialization phase is to obtain the labeled data which consist of the input vector and the corresponding output vector. As explained in the earlier section, the input vector should be the traffic patterns of the routers in the considered core network. The output vector should indicate the next node corresponding to the given traffic patterns. To gain this kind of training data, we can approach a number of available dataset sources, such as the Center for Applied Internet Data Analysis (CAIDA) [75], and extract the traffic information and relevant routing paths. Another way is to run the traditional routing protocols in our considered network, and record the number of inbound packets of every router and their routing tables.

3.3.2 Training Phase

In the training phase, we use the obtained data to train our designed DBAs. The training process consists of two steps: initializing each DBA with the Greedy Layer-Wise training method and fine-tuning the parameters $\Theta(W, B)$ with the backward propagation method. After the training phase, we can obtain the values of $\Theta(W, B)$.

As described in Sec. 3.2.1, the output of a DBA is a vector representing the next node, which means that it needs several DBAs to build a whole path. Assuming that only one router in the network trains and runs all the DBAs and produces all the paths in the network just like the centralized control strategy in the network, the quantity of

Algorithm 1 Supervised Train DBA
Input: $(X, Y) = \{(X^{(t)}, Y^{(t)})|t = 1, ..., m\}$, η_{CD}, η_{bp}, L (number of layers), $\boldsymbol{n} = (n_1, ..., n_L)$ (the numbers of units in each layer)
Output: Θ

1: **for** $i = 1, ..., L-2$ **do**
2: TrainRBM$(U^{(i)}, \eta_{CD}, n_i, n_{i+1})$
3: **end for**
4: Fine-tuneDBA$((X, Y), \Theta, \eta_{bp})$
5: **return** Θ

computation for the router will be extremely high. Also, such a central router requires a lot of time and resource to compute all the paths, leading to increased delay and unguaranteed accuracy. To reduce the computation requirement on routers and also increase the learning accuracy, we fragment the task of training into several parts and distribute them to every router in the target core/backbone network. This means that every router in the considered network needs to train several DBAs, each of which computes the next node from itself to a destination router. The number of DBAs a router needs to train depends on the number of its destination routers. Let N and I denote the total number of routers and the number of inner routers, respectively. Consequently, the number of destination nodes for each inner router is $(N-I)$, whilst every edge router has $(N-I-1)$ destination nodes since the source and destination routers cannot be the same. Therefore, every inner router needs to train $(N-I)$ DBAs, while all edge routers need to train $(N-I-1)$ DBAs.

For describing the training phase more clearly, we focus on the training procedures of only one DBA, which is also applicable to the other DBAs in our proposal. The main procedures of training a DBA are given in Algorithm 1. The inputs of the training phases are the training data (X, Y) as well as the parameters of the DBA, L and $\boldsymbol{n}$, and the learning rate, η_{CD} and η_{bp}. As shown in Algorithm 1, the training phase mainly consists of two steps: the loop of the Greedy Layer-Wise training to train each RBM as shown in Steps 1 to 3 and the following backward propagation process to fine-tune the weights of links between the layers shown in Step 4. Through the Greedy Layer-Wise training, the DBA is initialized with the values of $\Theta(W, B)$ nearly reaching the global optimum. Then the backward propagation algorithm is used to fine-tune the whole structure to minimize the value of the cost function. The adjusting process does not stop until the cost function is not more than a given value or the number of times reaches an upper bound. Once the backward propagation is finished, the value of $\Theta(W, B)$ of each DBA is recorded.

As mentioned earlier, every edge router needs to train $(N - I - 1)$ DBAs while each inner router needs to train $(N-I)$ DBAs, which means that every edge router can obtain Θ of $(N - I - 1)$ DBAs and each inner router can get Θ of $(N - I)$ DBAs. Then, every edge router needs to send its Θ of $(N - I - 1)$ DBAs to other $(N - I - 1)$ edge routers.

Also, every inner router needs to send its Θ of $(N-I)$ DBAs to all the edge routers. Therefore, each edge router obtains Θ of all the DBAs of all the routers in the network, and the number of sets of Θ is $(N-I)(N-1)$. Let DBA_{ij} represent the DBA in Router i for the destination Router j and Θ_{ij} is its parameter. Since edge routers obtain Θ of all the DBAs in the network, they can construct the corresponding DBA_{ij} with Θ_{ij}. It should be noted that $i \neq j$.

3.3.3 Running Phase

In the running phase, all the routers in the network need to record their numbers of inbound packets as traffic patterns periodically and send them to the edge routers. Then, every edge router can input the traffic patterns to its DBAs to obtain the next nodes to other edge routers. Also, since every edge router obtains the parameters Θ of other routers' DBAs, it can construct any DBA in the network and compute the next node from any router to any destination edge router. Therefore, every edge router can utilize the next node information to construct the whole paths from itself to all the other edge routers. The algorithm is shown in Algorithm 2. Here, we use an array of N elements, $\mathcal{TP}[N]$, to save the numbers of inbound packets of N routers in the network to represent the traffic patterns, and $\Theta[N-I][N-1]$ to save the parameters of all the DBAs in the network. Another array $\mathcal{ER}[N-I]$ is used to save the sequence numbers of the edge routers in the network since they are not continuous. In the real network situation, $\mathcal{ER}[N-I]$ is used to save the IP addresses of all the destination routers. After running Algorithm 2, each edge router can obtain the outputs of DBAs to construct the paths to $(N-I-1)$ edge routers. We can use a matrix, $\mathcal{NR}[N][N-I-1]$ to save the results of these DBAs that can be used to build the whole paths to all the other edge routers. Table 3.1 is the routing table built in router R_3, and Fig. 3.1 shows an example of the process of building the whole path from R_3 to R_{16}.

Table 3.1: Routing Table Built in R_3.

Dest	Path
R_1	$R_3 \rightarrow R_2 \rightarrow R_1$
R_2	$R_3 \rightarrow R_2$
...	...
R_{12}	$R_3 \rightarrow R_7 \rightarrow R_{11} \rightarrow R_{12}$
...	...
R_{16}	$R_3 \rightarrow R_7 \rightarrow R_{11} \rightarrow R_{15} \rightarrow R_{16}$

Algorithm 2 Running Phase
Input: $\mathcal{TP}[N]$, $\Theta[N-I][N-1]$, $\mathcal{ER}[N-I]$, sr (the source router).
Output: $\mathcal{NR}[N][N-I-1]$

$\boldsymbol{D} \leftarrow \mathcal{ER}[N-I] - sr$
while $\boldsymbol{D} \neq \emptyset$ **do**
    $d \in \boldsymbol{D}$
    $s \leftarrow sr$
    repeat
        $\Theta^{'} \leftarrow \Theta[s][d]$
        $nr \leftarrow$ run DBA with $\Theta^{'}$ and $\mathcal{TP}[N]$
        $\mathcal{NR}[s][d] \leftarrow nr$
        $s \leftarrow nr$
    until $nr = d$
    $D \leftarrow D - d$
end whilereturn $\mathcal{NR}[N][N-I-1]$

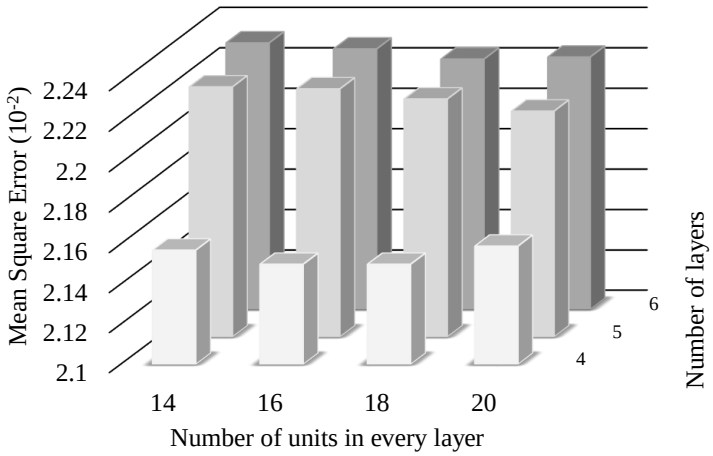


Figure 3.4: Mean Square Errors (MSEs) of different DBAs.

3.4 Computation Performance Analysis

In this section, we analyze the algorithm complexity and the time cost to run the proposed deep learning based routing strategy on the considered SDR. Our analysis mainly focuses on the numerical analysis of the algorithm complexity in the training phase and running phase via calculating how many times of addition operations $+$, subtraction operations $-$, multiplication operations $\times$, division operations $\div$, square root operations $\sqrt{}$, exponentiation operations e^x, and negation operations. To express clearly, the time cost of every kind of operations is denoted by ADD, SUB, MUL, DIV, SQRT, EXP, and NEG. Then, we evaluate and compare the time cost to run the two phases on a GPU and a CPU.

Algorithm 3 TrainRBM(V, η_{CD}, n_v, n_h)
Input: V ($V = \{V^{(t)} | t = 1, ..., m\}$. For RBM_1, $V = X$. For RBM_{L-2}, V is the activated output of RBM_{L-3} and Y. For other RBMs, V of RBM_i is the activated output of RBM_{i-1}), n_v, n_h, η_{CD}.
Output: $W = \{w_{ij} | i = 1, ..., n_v, j = 1, ..., n_h\}$, $A = \{a_i | i = 1, ..., n_v\}$, $B = \{b_j | j = 1, ..., n_h\}$.

for $i = 1, ..., n_v, j = 1, ..., n_h$ **do**
  $\Delta a_i = 0, \Delta b_j = 0, \Delta w_{ij} = 0,$
  $a_i = log\frac{p_i}{1-p_i}, b_j = 0, w_{ij} \sim N(0, 0.01^2)$.
end for
repeat
  for $t = 1, ..., m$ **do**
    for $j = 1, ..., n_h$ **do**
      compute $p(h_j = 1|V) = 1/(1 + e^{-(b_j + \sum_{i=1}^{n_v} w_{ij} v_i)})$
      sample h_j from $p(h_j = 1|V)$
    end for
    for $i = 1, ..., n_v$ **do**
      compute $p(v_i^{'} = 1|H) = 1/(1 + e^{-(a_i + \sum_{j=1}^{n_h} w_{ij} h_j)})$
      sample $v_i^{'}$ from $p(v_i^{'} = 1|H)$
    end for
    for $i = 1, ..., n_v, j = 1, ..., n_h$ **do**
      $\Delta w_{ij} \leftarrow \Delta w_{ij} + p(h_j = 1|V)v_i - p(h_j = 1|V^{'})v_i^{'}$
      $\Delta a_i \leftarrow \Delta a_i + v_i - v_i^{'}$
      $\Delta b_j \leftarrow \Delta b_j + p(h_j = 1|V) - p(h_j = 1|V^{'})$
    end for
  end for
  for $i = 1, ..., n_v, j = 1, ..., n_h$ **do**
    $w_{ij} \leftarrow w_{ij} + \eta_{CD}\Delta w_{ij}/m$
    $a_i \leftarrow a_i + \eta_{CD}\Delta a_i/m$
    $b_j \leftarrow b_j + \eta_{CD}\Delta b_j/m$
  end for
until $iter = r_1$
return W, A, B

3.4.1 DBA Precision Analysis

Since the number of routers in the network shown in Fig. 3.1 is 16, for each DBA, the numbers of units in the input and output layers are both 16. In our simulation, the number of the training data is 100,000. To determine the number of hidden layers and number of units in each hidden layer for each DBA, we train different DBAs. The Mean Square Error (MSE) measuring the prediction error rate of a DBA is given in Fig. 3.4. It can be noticed that the DBAs consisting of 4 layers and 16 or 18 units in each layer have the minimum MSE values. Considering more units in the hidden layers mean more complexity, we choose the DBA which has 2 hidden layers and 16 units in each hidden

layer.

To elucidate the input and output of our proposed deep learning system model, please refer to Table 3.2. This table gives the error rates of five structures in a network with 16 routers. The first row represents the centralized routing control. The input is the traffic pattern in the last time-interval of 16 routers while the output gives the whole paths between any two edge routers in the network. We can find that the number of units in the output layer is more than 30,000, the structure of which becomes extremely complex resulting in significantly poor accuracy. The structures shown in the following two rows are both using one deep learning system to output a whole path. The main difference between them is as follows. The second structure uses a 16×16 matrix to show the path and the elements in the matrix have binary values. On the other hand, the third structure outputs a vector in which the values of some elements represent the routers chosen in the path. We can find that the error rates of the second and third structures are as high as 70% and 45%, respectively. If we choose the next node as the output indicated in the final two rows, we can find that the error rates are just 5%. These two structures use our proposed input and output model, and the fourth structure uses the traffic pattern of only 1 time-interval while the final structure uses three time-intervals' traffic patterns as the input. The output layers in the two structures both consist of 16 units and output a 16 dimensional vector, of which only one element has the value of 1 representing the next router in the path. Even though the two structures have the same performance, the final structure is much more complex than the fourth one because of more units in the input layer. Therefore, the fourth structure we choose has the lowest error rate and the simplest structure compared with other strategies.

3.4.2 Complexity Analysis of the Training Phase

The main process of the training phase consists of training each RBM and fine-tuning the whole DBA as shown in Algorithm 1. The training algorithm of each RBM is shown in Algorithm 3 which is an unsupervised training process. Suppose that the numbers of units

Table 3.2: Effect of different input and output characterization strategies on the network control accuracy for N=16.

Number of input nodes	Number of output nodes	The whole path or next node	Error rate
16	33792	Path	-
16	256	Path	70%
16	16	Path	45%
16	**16**	**Next node**	**5%**
48	16	Next node	5%

in the visible layer and the hidden layer are n_v and n_h, respectively. The number of training sets is m. First, from Step 1 to Step 4 in Algorithm 3, we need to initialize W, A, B and $\Delta W, \Delta A, \Delta B$. Then, we repeatedly utilize all training examples to update the values of $\Delta w_{ij}, \Delta a_i, \Delta b_j$ with the method named CD for adjusting W, A, B [47]. As shown in Step 7 to Step 19, the CD method mainly consists of two periods. The first period is to adopt the method of Gibbs Sampling to get a sample value of h_j and $v_i^{'}$ according to their conditional probability distributions shown in Step 7 to Step 14. Second, the obtained sample value of $v_i^{'}$ is used to update $\Delta w_{ij}, \Delta a_i, \Delta b_j$ according to Step 15 to Step 19. Therefore, the values of $\Delta w_{ij}, \Delta a_i$, and Δb_j can be utilized to update w_{ij}, a_i, and b_j which has been shown in Step 21 to Step 25. The whole training process is repeated r_1 times which takes $r_1((3m+1)n_v n_h + 3mn_v + (2m+1)n_h)$ADD+$r_1(mn_v n_h + (m+1)n_v + mn_h)$SUB+$r_1((4m+6)n_v n_h + n_v + n_h)$MUL+$r_1(2n_v n_h + (m+1)n_v + mn_h)$DIV+$r_1 n_v n_h$(EXP+NEG+SQRT).

Since the visible layer of the first RBM satisfies the Gaussian Distribution as mentioned in Sec. 3.2.2, for training the first RBM, we need to calculate σ_i and $(a_i + \sigma_i \sum_j h_j w_{ij})$ denoting the standard deviation and the mean value of unit i, respectively. This step requires $2N(m-1)$ ADD, Nm SUB, Nm MUL, $2N$ DIV, and N SQRT operations. It should also be noted that when training the first RBM, the conditional probability distribution of the visible layer should be revised to Equation 2.24. The difference of the time cost of the first RBM from other RBMs is negligible.

After finishing the complexity analysis of the first step, we turn to the second step which adopts the stochastic gradient descent of the cost function defined in Equation 3.3 to fine-tune the values of the weights and biases. The detailed procedures of the second step are shown in Algorithm 4 which mainly consists of four operations: forward propagation (Step 3 to Step 8), backward propagation (Step 9 to Step 16), updating the values of W, B (Step 17 to Step 24), and calculating the cost function (Step 26 to Step 37). As shown in the forward propagation from Step 3 to Step 8, the weighted value, $z_j^{(l)}$, and the activated value, $u_j^{(l)}$, of every unit in each layer are calculated. The activation function chosen here is the sigmoid function, then $u_j^{(l)} = 1/(1 + e^{-z_j^{(l)}})$. Consequently, we can get the error of the last layer, $\delta_i^{(L)}$, which is defined as the difference between the activated values of the last layer and the labeled output as shown in Step 10. As the units' values in the last layer are the results of which the units' values in the first layer propagate layer by layer, the error of the last layer is caused by the errors of previous layers. Step 12 to Step 16 show how to utilize the error of the l^{th} layer, $\delta_i^{(l)}$, to calculate the error of the $(l-1)^{th}$ layer, $\delta_i^{(l-1)}$, according to the relationship between the two layers, which is a backward propagation process. Then, the error of each layer, $\delta_i^{(l)}$, can be adopted to update the values of W, B according to Steps 17 to 24. After obtaining the updated values of W, B, we can re-calculate the value of the cost function, C, the procedures for which is shown from Step 26 to 37. Then, it can be confirmed whether a new iteration

Algorithm 4 Fine-Tune the DBA$((X,Y),W,B,\eta_{bp})$
Input: $(X^{(t)},Y^{(t)})$, $W=(W^{(2)},...,W^{(L-1)})$, $W^{(l)}=\{w_{ij}^{(l)}|i=1,...,n_{l-1},j=1,...,n_l\}$, $B=(B^{(2)},...,B^{(L)})$, $B^{(l)}=(b_1^{(l)},...,b_{n_l}^{(l)})$, η_{bp}.
Output: W,B.

repeat
  for $t=1,...,m$ **do**
    for $j=1,...,n_1$ **do**
      initialize the units, $z_j^{(1)}$, of the input layer with $x_j^{(t)}$
    end for
    for all $l\in\{2,...,L\}$ and $j\in\{1,...,n_l\}$ **do**
      $z_j^{(l)}=\sum_{i=1}^{n_{l-1}}w_{ij}^{(l)}u_i^{(l-1)}+b_j^{(l)}$ and $u_j^{(l)}=1/(1+e^{-z_j^{(l)}})$
    end for
    for $i=1,...,n_L$ **do**
      $\delta_i^{(L)}=u_i^{(L)}-y_i^{(t)}$
    end for
    for $l=L-1,...,2$ **do**
      for $i=1,...,n_l$ **do**
        $\delta_i^{(l)}=\sum_{j=1}^{n_{l+1}}w_{ij}^{(l+1)}\delta_j^{(l+1)}u_i^{(l)}(1-u_i^{(l)})$
      end for
    end for
    for $l=2,...,L$ **do**
      for $j=1,...,n_l$ **do**
        $b_j^{(l)}\leftarrow b_j^{(l)}-\eta_{bp}\delta_j^{(l)}$
        for $i=1,...,n_{l-1}$ **do**
          $w_{ij}^{(l)}\leftarrow w_{ij}^{(l)}-\eta_{bp}(\delta_i^{(l)}u_i^{(l-1)}+\lambda w_{ij}^{(l)})$
        end for
      end for
    end for
  end for
  for $t=1,...,m$ **do**
    for all $l\in\{2,...,L\}$ and $j\in\{1,...,n_l\}$ **do**
      $z_j^{(l)}=\sum_{i=1}^{n_{l-1}}w_{ij}^{(l)}u_i^{(l-1)}+b_j^{(l)}$ and $u_j^{(l)}=1/(1+e^{-z_j^{(l)}})$
    end for
    for $j=1,...,n_L$ **do**
      $C_1\leftarrow C_1-(y_j^{(t)}\log u_j^{(L)}+(1-y_j^{(t)})\log(1-u_j^{(L)}))$
    end for
  end for
  for $l=2,...,L,j=1,...,n_l,i=1,...,n_{l-1}$ **do**
    $C_2\leftarrow C_2+(w_{ij}^{(l)})^2$
  end for
  $C=\frac{1}{m}C_1+\frac{\lambda}{2}C_2$
until convergence
return W,B

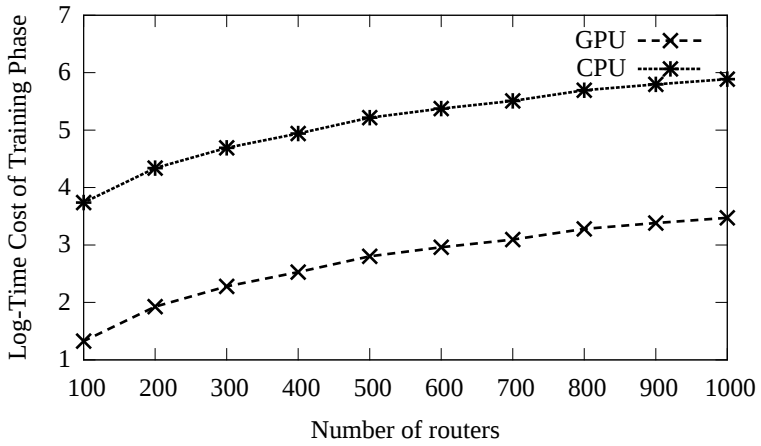


Figure 3.5: The time cost of training phase on the chosen GPU and CPU-based SDRs.

should be executed according to the value of C. If we assume that the algorithm iterates r_2 times, the total time cost is $r_2((4m+1)\sum_{l=2}^{L} n_l n_{l-1} - mn_1 n_2 + m(\sum_{l=2}^{L} n_l + 2n_L) + 1)\text{ADD} + r_2 m(2\sum_{l=2}^{L} n_l n_{l-1} - n_1 n_2 + \sum_{l=2}^{L} n_l + 4n_L)\text{SUB} + r_2((8m+1)\sum_{l=2}^{L} n_l n_{l-1} - mn_1 n_2 + m\sum_{l=2}^{L} n_l + 2mn_L + 1)\text{MUL} + r_2(2\sum_{l=2}^{L} n_l + 2)\text{DIV}$.

After obtaining the number of different operands for training every DBA, we can theoretically analyze the time cost of utilizing a GPU (the Nvidia Titan X Pascal) or a price-comparable CPU (Intel i7-6900K) to execute the calculation. The GPU, Titan X, has 28 SMs, each of which can run 128 times of 32-bit float point arithmetic calculation in a clock cycle. The CPU, Intel i7-6900K, has 8 cores and 16 threads. The latencies for different arithmetic operands that we choose are 3, 5, and 15 clock cycles for ADD/SUB, MUL, and DIV operations, respectively [76]. Since the numbers of the EXP, SQRT, and NEG operands are much fewer than those of other operands, it is reasonable to neglect the time cost of these operands, EXP, SQRT, and NEG. The number of training samples is 100,000 ($m = 100,000$) and the values of r_1 and r_2 are both assumed to be 10,000. Then, we can calculate the values of the time cost of the algorithm running on the GPU and the CPU as shown in Fig. 3.5. It can be found the logarithm value of the time cost of the GPU-based SDR is more than 2 smaller than that of the CPU-based one. This indicates that the GPU-based SDR, for training the proposed deep learning architectures, achieves more than 100 times faster performance than that of the CPU-based SDR. Even though the time cost of the GPU-based SDR is more than 1,000 seconds when the number of routers is 1,000, the training phase of the SDR can be operated offline to avoid network performance degradation.

3.4.3 Complexity Analysis of the Running Phase

In this section, we analyze the time cost of the proposed routing strategy in the running phase. As we mentioned above, the training phase can be regarded as the initialization pe-

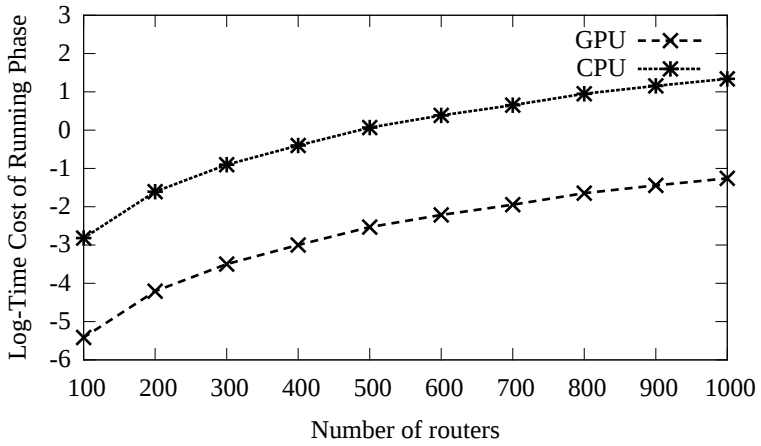


Figure 3.6: The time cost of running phase on the chosen GPU and CPU-based SDRs.

riod of the SDR. Consequently, the SDR mainly works in the running phase. The detailed procedures of the running phase is a feedforward propagation which can be regarded as the same as Steps 3 to 8 in Algorithm 4. As mentioned earlier in Sec. 3.3.3, the running phase is only executed in the edge routers and every edge router only constructs the paths from itself to all the other edge routers. Therefore, in the running phase, every edge router just needs to run the DBAs which compute the next nodes that exist in its paths. It is necessary to assume the average number of nodes in one path to be A due to the uncertainty about the number of routers in one path. Consequently, every edge router needs to run $(N-I-1)A$ DBAs. Therefore, the time cost for every edge router to construct its paths is $(N-I-1)A\sum_{l=2}^{L} n_l(n_{l-1}+1)$ADD+$(N-I-1)A\sum_{l=2}^{L} n_l n_{l-1}$MUL+$(N-I-1)A\sum_{l=2}^{L} n_l$(DIV+EXP+NEG).

Then, we can calculate the values of the time cost of the running phase on the chosen GPU and CPU-based SDRs as demonstrated in Fig. 3.6. The value of A is set to $0.2N$. Since the logarithm value of the time cost of the GPU is about 2 smaller than that of the CPU, it is about 100 times faster to use the GPU than running the algorithm with the CPU-based SDR. We can find that when the number of routers is less than 400, the time cost of the GPU is less than 1 millisecond while that of the CPU is more than 100 milliseconds. This demonstrates that the proposed deep learning based routing strategy runs very fast in the GPU-accelerated SDR.

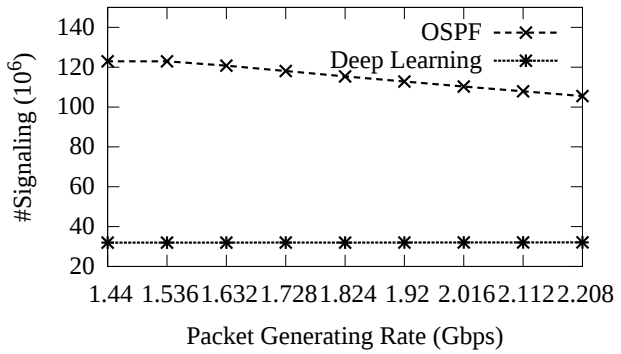
In addition to the complexity analysis, in the following section, we further present a simulation-based network performance evaluation of our proposed deep learning based routing technique on a backbone network constructed with commodity routers.

3.5 Network Performance Evaluation

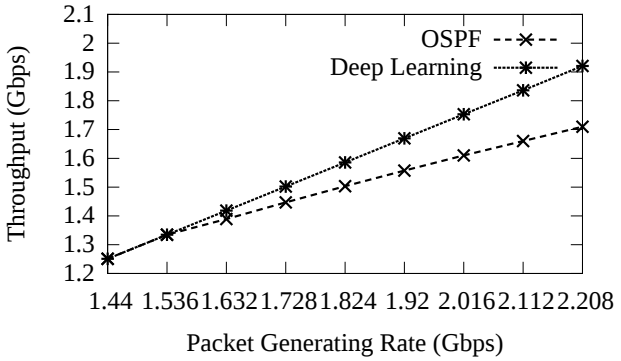
This section evaluates the effectiveness of our proposed deep learning based routing strategy in terms of network performance. In order to accommodate our characterization of the input and output, C++/WILL-API [41] is utilized since it provides the library of DBAs, which is not available in other simulators such as Caffe and Microsoft Cognitive Toolkit [77]. Therefore, we use C++/WILL-API as the simulation framework. In the simulation, all routers' computations are conducted on a workstation with a six-core i7 3.3 Ghz processor and 16 GB RAM. As the computations of all routers in our considered network are outsourced to a single machine, it is reasonable to restrict the simulation to a small size network. Therefore, we consider a medium size wired backbone network as shown in Fig. 3.1 rather than a full-scale core network topology. It is worth noting that this scale of simulation is sufficient enough to demonstrate that the proposed deep learning based routing strategy outperforms the conventional routing strategies such as OSPF. As described in Sec. 3.2.1, only the edge routers generate data packets and these packets are destined for the edge routers, while the inner routers just forward the data packets. On the other hand, all the routers can generate signaling packets. In addition, the signaling packets consist of the traffic patterns and are destined for the edge routers in our proposal, while all the routers flood signaling packets to exchange the routing tables in the OSPF protocol. The sizes of the data packets and the signaling packets are set to 1 kb. The link capacity is set to 20 Gbps. Here, we assume that every router has an unlimited buffer. As mentioned earlier, we need to use supervised training of our DBAs, the training data should consist of the traffic data and the subsequent nodes. However, most realistic traffic traces offered by the public website [75] consist of a mix of routing protocols, which are difficult to use for supervised training. Moreover, as the goal of this chapter is to evaluate the performance of applying deep learning into routing, it is reasonable to choose an existing routing protocol as the benchmark in the simulation. Since the practical traffic data come from the networks using mixed routing protocols, if we use the data to train our deep learning architectures, it is unfair to compare the performance of the proposed routing strategy with our considered benchmark routing protocol. Therefore, in our simulation, we first run the OSPF protocol to build the routing table in the considered network and record the traffic patterns and the corresponding paths. Therefore, we can utilize the recorded traffic patterns and corresponding paths to construct the labeled data for training the DBAs in the training phase.

In this section, we first evaluate the precision of our DBAs for the given core network, before which we decide the number of hidden layers and the number of units required in each hidden layer. We also give a comparison of different characterization strategies of inputs and outputs, and demonstrate that our proposal has the highest precision and the

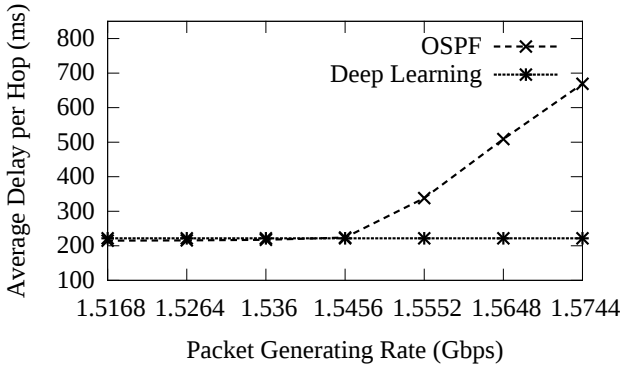
lowest complexity. Then, the network performance with our proposed routing strategy is compared with that of OSPF from three aspects, i.e., the signaling overhead, the network throughput, and the average delay per hop.



(a) Comparison of signaling overhead for the conventional OSPF and the proposed deep learning system.



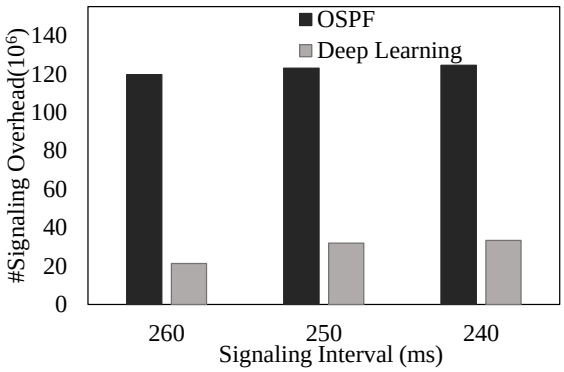
(b) Comparison of aggregate throughput for the conventional OSPF and the proposed deep learning system.



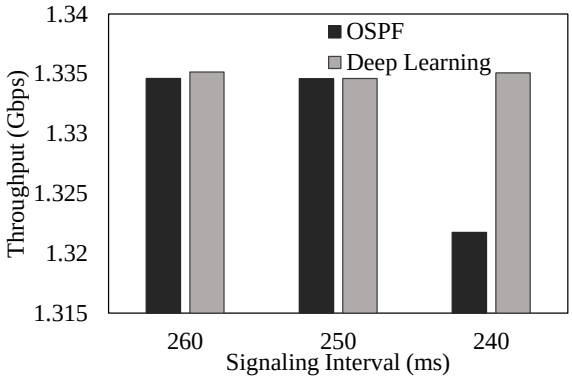
(c) Comparison of average delay per hop for the conventional OSPF and the proposed deep learning system.

Figure 3.7: Comparison of network performance under different network loads in our proposal and the bencmark method (OSPF) in terms of signaling overhead, throughput, and average delay per hop.

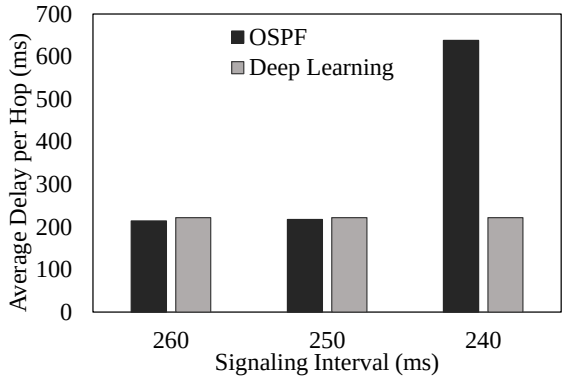
In the running phase, we choose OSPF as a benchmark method to compare the pro-



(a) Comparison of signaling overhead for the conventional OSPF and the proposed deep learning system.



(b) Comparison of aggregate throughput for the conventional OSPF and the proposed deep learning system.



(c) Comparison of average delay per hop for the conventional OSPF and the proposed deep learning system.

Figure 3.8: Comparison of network performance under different signaling intervals in our proposal and the bencmark method (OSPF) in terms of signaling overhead, throughput, and average delay per hop.

posed deep learning based routing strategy. To compare the performance under various network loads, we change the data generating rate and record the values of network sig-

naling overhead, throughput, and average delay per hop. The signaling interval is fixed at 0.25 second. Fig. 3.7a and Fig. 3.7b compare the numbers of successfully transferred signaling packets and the network throughput with two routing strategies when the data generating rate changes from 1.44 Gbps to 2.208 Gbps. Fig. 3.7c compares the variation of average delay per hop under two scenarios when the data generating rate increases from 1.5168 Gbps to 1.5744 Gbps. In Fig. 3.7a, we can find the number of successfully transferred signaling packets in our proposal remains nearly unchanged, which is normal since the signaling interval and the simulation time are both fixed. However, in the network using the conventional OSPF protocol, the number of successfully transferred signaling packets gradually decreases when the data generating rate is more than 1.536 Gbps, which can be explained by the traffic congestion and the following increasing loss of some signaling packets. It can be noticed that the number of signaling packets in the conventional case is much higher than the number in our proposal. This happens because in our proposal, every router only needs to send the signaling packets to the edge routers for computing the routing paths while in OSPF every router needs to flood the signaling packets to all the other routers in the network. The difference in the quantities of signaling packets affects the network throughput and the average delay per hop. Fig. 3.7b demonstrates that the throughput of our proposal linearly increases with the data generating rate. However, in the network using OSPF, the throughput increases linearly before the data generating rate reaches 1.536 Gbps, and after that, the throughput increases rather slowly. The difference of performance in the two routing strategies is more clearly shown in Fig. 3.7c which demonstrates the changes of the average delay per hop with the increasing network overhead. It can be observed that the average delay per hop under the two scenarios is nearly the same when the data generating rate is below 1.5456 Gbps due to the fact that the DBAs in our proposal are trained with the data from OSPF. Therefore, it can be concluded that the training of our DBAs is successful since it can give the same output as OSPF. However, the average delay per hop in OSPF increases after the data generating rate exceeds 1.5456 Gbps, while that of our proposal still remains unaffected. This can be explained by the occurrence of traffic congestion, when the data generating rate is above 1.5456 Gbps in the network with OSPF, leads to the decreasing throughput and increasing average delay per hop. On the contrary, for the shown data generating rates, the proposed routing strategy based on deep learning achieves much lower signaling overhead and avoids the traffic congestion issue.

After the analysis of network performance with various data generating rates, we further analyze and compare the effects of different signaling overheads on network performance using the two different routing strategies. Here, we fix the data generating rate at 1.536 Gbps and change the signaling interval from 260 ms to 240 ms. Figs. 3.8 show the result consisting of the signaling overheads, throughput, and average delay per hop for

the two cases when the signaling intervals are 260 ms, 250 ms, and 240 ms, respectively. In Fig. 3.8a, we can find that the signaling overheads in our proposal are much lower than those in the case with OSPF. In Figs. 3.8b and 3.8c, we can clearly see the effects of signaling overheads on the performance of the two cases. In Fig. 3.8b, the throughput of our proposal remains nearly unchanged when the signaling interval is different. On the other hand, the throughput of OSPF, when the signaling interval is 240 ms, is much lower than that when the signaling interval is 260 ms or 250 ms. Thus, it may be inferred that the traffic congestion happens for the network using OSPF when the signaling interval is 240 ms. This is further demonstrated by the result in Fig. 3.8c which shows that the average delay per hop of OSPF, when the signaling interval is 240 ms, is nearly twice longer than that when the signaling interval is 260 ms or 250 ms. Moreover, we can find that when the signaling interval is 260 ms or 250 ms, the average delay per hop of OSPF is nearly the same as that of our proposal.

Through comparing the performance in the network using OSPF and our proposed routing strategy based on deep learning, we can find that our proposed deep learning based routing strategy has much lower signaling overhead, leading to better traffic control. The reason for the lower signaling overhead in our proposal is that only the edge routers instead of all routers require signaling packets since the edge routers can use the trained DBAs to build the whole paths and the inner routers do not need the signaling packets to compute the next nodes. However, in the network with OSPF, the edge routers cannot utilize current weights' values of all links to build the practical whole paths as the paths computed through OSPF are only suitable for current network states. But during the packets' transmission, the network traffic is changing and then the decided paths become unsuitable. On the other hand, for the routing strategy based on deep learning, the DBAs can find the complex relationship between the current traffic patterns and the real paths if we utilize the traffic patterns and real paths to train them. Therefore, the edge routers can utilize the trained DBAs to build the whole paths with only current network information.

3.6 Summary

In this chapter, we explored current SDR architectures and envisioned that deep learning, which has recently emerged as a promising machine learning technique, can be used to compute the routing paths instead of the conventional routing protocol. This can substantially improve the backbone network traffic control. Considering current GPU-accelerated SDRs enable the massively parallel computing, we proposed a supervised deep learning system to utilize the traffic patterns to compute the paths directly, different from the conventional rule-based routing. The simulation result shows that the proposed deep learning

based routing strategy outperforms the conventional OSPF in terms of the network packet transmission throughput and average delay per hop since our proposal has much lower signaling overhead. This demonstrated that the shift of routing computation from the traditional rule-based strategy to deep learning can improve the backbone network control substantially. In addition, the complexity of our proposed routing strategy was analyzed to evaluate that the GPU-accelerated SDR is much more efficient to run the proposed algorithms than the CPU-based SDR.

Chapter 4

Online Learning Based Routing Strategy for Software Defined Communication Systems

4.1 Introduction

In last chapter, the supervised learning based routing strategy is proposed to tackle the increasing traffic overhead in the backbone networks. Since the labeled data impact on the performance of the considered deep learning architectures, it is a critical step to collect the training data. However, in many scenarios, it is very difficult to collect enough satisfying labeled data, for which the heterogeneous network is a good example.

As we know, for some heterogeneous networks, various kinds of communication technologies, e.g. FiWi, D2D, and 5G, are utilized to meet users' requirements in different scenarios [78, 79, 80]. Since these networks have various infrastructures and topology, to solve the difficulty in managing all these networks, researchers considered the SDN technology [33, 81]. As we mentioned in Chapter 1, the structures of routers and switches in the SDN scenarios get significantly simplified and unified due to the separation of complicated network logic. Similar to the cloud-based computing applications, the controllers conduct all the computation tasks for the switches [13]. Therefore, the utilized controller in SDN is usually composed by various computation platforms. To fit for the new network scenario as well as the improved computation capacity in SDN, the network algorithms should have been updated or redesigned. However, the packet forwarding algorithms in current Software Defined Communication Systems (SDCSs) [82] still follow the conventional manner [28]. Since the paths are computed according to fixed rules, when similar traffic patterns happen, the controller chooses the same paths even the decision has been previously proved wrong, which leads to unnecessary network performance deterioration.

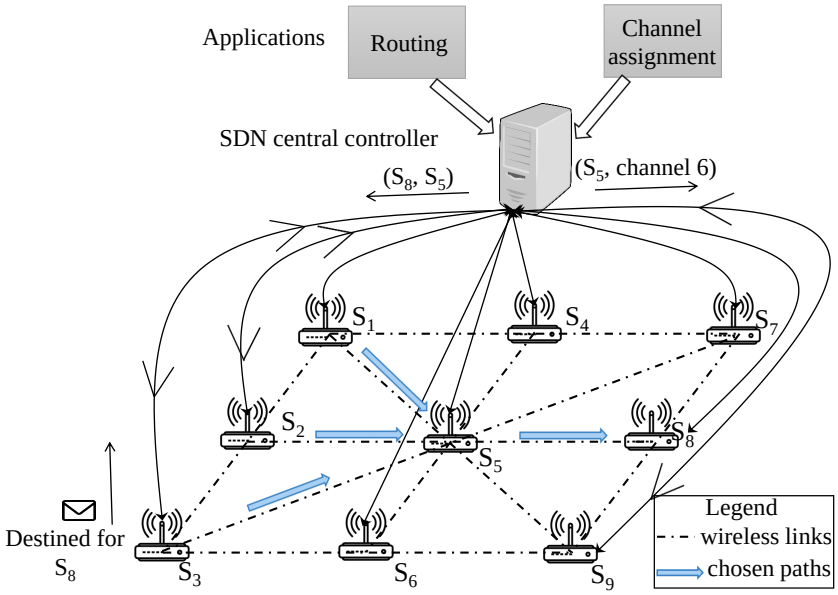
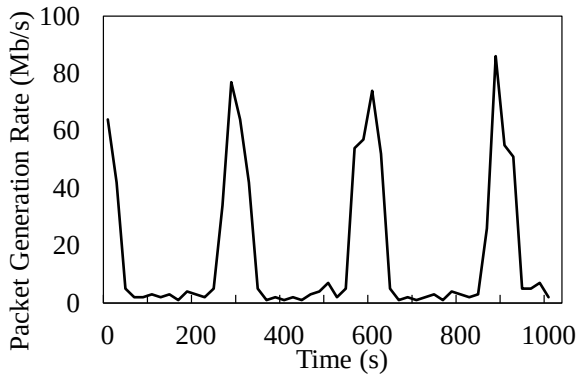


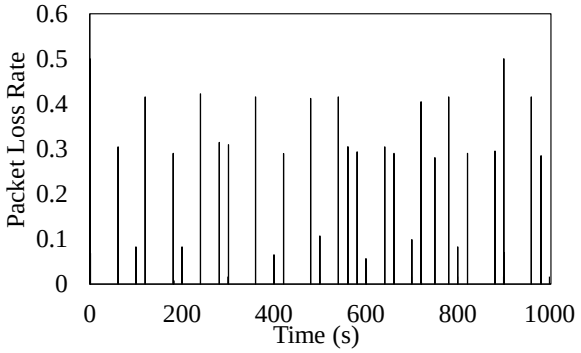
Figure 4.1: The considered structure of SDCS.

This means that the current routing protocols lack the intelligence to learn from previous experiences. On the other hand, if we consider the deep learning based routing strategy similar to that in Chapter 3, it is still difficult to obtain the satisfied performance. This is because the data planes in the SDCSs consist of various communication technologies, different from the backbone networks. Therefore, the traffic patterns in the data plane are more complex and varying fast. Also, the bursty traffic is very common in the data plane. Therefore, even the considered architecture is trained with massive data, it cannot predict the paths accurately since the network surrounding may have changed. To solve this problem, in this chapter, we propose an online learning based routing strategy which periodically trains the considered architecture with real-time traffic patterns. The proposal consists of two steps: the initial phase and running phase. In the initial phase, the controller runs the conventional routing protocol while the switches record the traffic trace, which is utilized by the controller to initialize the utilized CNNs. Then, in the running phase, the CNNs are adopted in the controller to choose paths. Furthermore, to adapt the trained CNNs to the changing traffic patterns as well as reduce the training computation overhead, in the running phase, the switches keep recording the traffic trace for periodically retraining the CNNs in the controller.

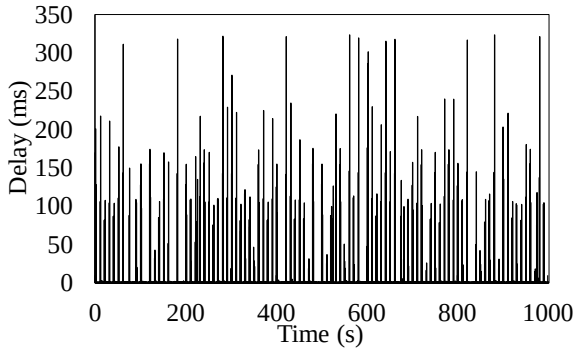
The remainder of the chapter is organized as follows. Sec. 4.2 describes the problems in the routing strategies for current SDCSs, and then discusses our proposal to overcome the problems. The detailed procedures of our proposal are presented in Sec. 4.3. Sec. 4.4 analyzes the time and space complexity for the controller and switches in our proposal. We evaluate the network performance of our proposal in Sec. 4.5. Finally, Sec. 4.6 summarizes this chapter.



(a) The traffic patterns of switches S_1, S_2, and S_3.



(b) The network packet loss rate.



(c) The network delay.

Figure 4.2: An illustrative example: when switches S_1, S_2, and S_3 choose S_5 as the next node to destination S_8, S_5 will be the bottleneck, which means that traffic congestion will easily happen to S_5.

4.2 Problem Statement and Model Design

With the increasing number of users and rapidly changing network environment, global networks are confronted by many challenges. To meet the future network requirement, SDN has been regarded as the next generation network paradigm since the separation of complex control logic and data forwarding significantly simplifies and unifies the structures of the switches. Moreover, the well-defined programmable interface increases the network flexibility. However, current packet forwarding algorithms still follow conventional fixed-

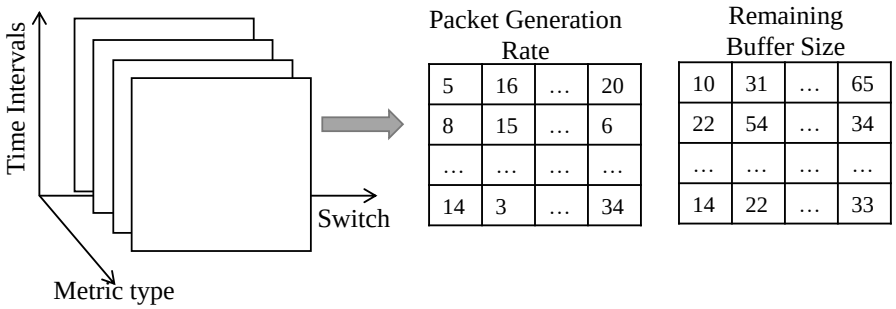


Figure 4.3: The input of the CNN in our proposal.

rule-based routing protocols, e.g., minimum or maximum metric values, resulting in the same decision when similar traffic patterns occur, even though the decision has been proved wrong. To describe this more clearly, we can take the data plane as shown in Fig. 4.1 as an example. Here, it is reasonable to assume that some switches, S_1, S_2, and S_3, in the data plane, generate packets destined for the switch S_8. And the central controller chooses the best paths which have the minimum hop numbers. According to the network topology, it is easily understood that the central controller chooses S_5 as the next node for source switches, S_1, S_2, and S_3, to send packets to S_8. Therefore, when three source switches send packets to S_8 along with their shortest paths concurrently, the joint router of these paths, S_5, easily becomes the network bottleneck, leading to the degradation of network performance [83]. Even though the congestion can be alleviated after the switches inform the controller about the congestion and the controller updates the paths for S_1, S_2, and S_3, this situation can happen again while similar traffic patterns appear. Seriously, when source switches S_1, S_2, and S_3 have burst traffic patterns as shown in Fig. 4.2a and all these packets are destined for S_8, the joint switch S_5 becomes congested and the network performance in terms of the packet loss rate and average packet delay deteriorates repeatedly as shown in Figs. 4.2b and 4.2c.

The reason behind this phenomenon is because of the fixed rule based routing strategy which lacks the ability of self-reconfiguration. According to the conventional routing algorithm, the controller always chooses S_5 as the next node for S_1, S_2, and S_3 since these paths have the minimum hop numbers, despite of the fact that this decision has been proved wrong many times. If the router can adjust its configuration according to the previous experience, the joint node can be easily avoided. However, conventional proprietary hardware based router architecture does not allow the reconfiguration of the routing rules without redesigning the hardware architecture. Furthermore, to increase the adaptation of the routing strategies is concerned with more complex algorithms. On the other hand, the central controller equipped with a large computation capacity in the SDCSs can act as the platform to run the improved routing algorithms.

In this chapter, we consider utilizing deep learning to improve the routing strategy as

the deep learning technique is promising in the future communication network management. To design a deep learning based routing protocol, the first step is to choose the suitable deep learning structure and define its input and output. Since our goal is to manage multiple paths to avoid the traffic congestion, we can choose the most direct sign of the traffic situation, the traffic pattern, as the input. We can merge the values of different features of the traffic pattern into a three dimensional matrix as shown in Fig. 4.3. And the three dimensions represent the router ID, the time interval, and different features of traffic patterns, respectively. If we use TP to denote the input matrix, then the value of unit, tp_{ijk} denotes the value of switch S_i's feature k at the j^{th} time interval. The size of the router ID dimension depends on the number of considered switches, while that of the time dimension is decided by many factors, such as the network size, the required accuracy rate, and so on. The number of chosen time intervals should be reasonable since traffic patterns of too few time intervals leads to low prediction accuracy and too many time intervals cause high computation burden. The number of traffic pattern features is concerned with our purpose. As the deep learning architectures in our proposal are utilized for routing, we choose the packet generation rates and the switches' remaining buffer sizes as our considered features. Considering the input design, suppose the whole network as an image and different traffic pattern features as the different color channels of the image. Therefore, it is reasonable to choose the CNN shown in Fig. 2.2c as the deep learning structure due to its wide applications to process the images. For the output design, since there are several paths for each OD pair in our proposal, the chosen paths in every round can be regarded as a paths combination, which consists of one path for each OD pair. Then, each CNN can represent one paths combination and the output of the CNN should indicate whether the paths combination will be congested or not. In our proposal, we can use a two dimensional vector to denote the output. We can set $(1, 0)$ as the notation of congestion and use $(0, 1)$ to denote that the paths combination will not be congested in the next round. Moreover, if the trained CNN outputs results of (0, 0) or (1, 1), it means that the training is not effective. Therefore, we can make some adjustment of the architecture or accumulate more training data. Since the output layer consists of the binary units, we consider the softmax regression method in the final output layer. The cost function is the same as Equation 2.36. To adapt to the changing traffic patterns, we consider the online training manner which means the adopted CNNs will be periodically trained with the real-time traffic patterns. Therefore, the CNNs can adjust its parameters through the training with new traffic trace.

4.3 Procedures of Our Proposal

After introducing our strategy to the problems, in this part, we describe the detailed procedures. Our proposal can be divided into two phases: initial phase and running phase. In the initial phase, we construct a CNN for each paths combination and run the conventional routing protocols to obtain some data to train these CNNs before utilizing the CNNs to choose the paths combination in the running phase. As mentioned previously, the CNNs will be periodically retrained for learning the new network experiences. Therefore, besides the routine path update with CNNs, the running phase consists of two periodically conducted process: data collection and CNN retraining. Here, it should be noted that the cycle time of the three process is decided according to the performance. To describe the three process more clearly, we utilize δ, t_u and t_r to denote the cycle time of recording traffic patterns, updating paths, and retraining CNNs, respectively. And they should satisfy some relationships which can be assumed that $t_r = n_1 t_u$ and $t_u = n_2 \delta$ (n_1 and n_2 are both integers). The detailed procedures are shown in Fig. 4.4, and Algs. 5, 6 and 7 present the pseudocodes of our proposal. In the following, we will introduce the two phases according to the figure and algorithms.

4.3.1 Initial Phase

Since the purpose of initial phase is to obtain some data to train the CNNs for the considered paths combinations, the initial phase consists of two process: utilizing the conventional routing protocols to forward packets and recording the traffic trace, and training the CNNs. We explain the two process in details one by one.

Firstly, the switches execute a neighbor discovery process and send the neighbor information to the central controller. Then the central controller builds a global view of the whole network, including the positions of the switches and their interfaces. Therefore, as shown in Step 6 in Algorithm 5, the controller can run the conventional routing protocols according to the network topology to choose the best path for every OD pair in each path update cycle. Then, the controller generates the packet forwarding rules and installs the rules on corresponding switches. Consequently, the switches in the data plane can forward the packets according to the installed rules. Once some link is congested and some switch becomes inaccessible, the neighbor switch can upload the information to the controller for updating the path.

Apart from the packet forwarding process, the controller also needs to construct and train CNNs for choosing the network paths combinations. The procedures consist of two steps. The first step happens at the very beginning when the controller first gets the global view of the whole network and the weight values of all links as shown in Step 1 in Algorithm 5. With the information, the controller can compute multiple paths for

every OD pair according to some predefined requirements, for example, the maximum weight value should not exceed two times of the minimum weight value. Here, we can use a vector consisting of n units to save a path in a network made up of n switches, and utilize a $m \times n$ matrix to save the m paths for an OD pair in a descending priority order defined by the paths' metric values. It should be noted that m is the maximum path number among all OD pairs and zeros should be padded in the matrix if the path number is smaller than m for some OD pairs. We can use $P_{i,j}$ and $p_{i,j}^{k}$ to denote the paths matrix for source switch i and destination switch j and the k^{th} path, respectively. The controller can choose one path from each path matrix to construct a paths combination. And all the paths combinations can be saved in a three-dimensional matrix represented by C in a descending priority order. In this matrix, the value of its unit $c_{i,j,k}$ denotes the path order for OD_{ij} in the k^{th} paths combination.

Algorithm 5 Initial Phase
Input: network topology.
Output: CNNs.

1: The controller generates the global view of the whole network according to the network topology, compute the paths combination matrix C with graph theory;
2: **for** each paths combination $c_{..k}$ in C **do**
3: Controller creates a CNN, $CNN_{..k}$
4: **end for**
5: **for** each path update interval t_u **do**
6: The controller computes the best paths and generates the forwarding rules, and then installs the rules on corresponding switches
7: **for** each traffic patterns recording interval δ **do**
8: Every switch forwards packets according to the installed rules, records the traffic patterns, and calculates the delay for each received packet.
9: Every switch calculates the delay of the paths destined for itself, sends the traffic pattern and path delay values to the controller.
10: **end for**
11: The controller constructs the input traffic patterns of the CNN, TP.
12: **if** the delay for any path of OD_{ij}, $d_{i,j} > threshold$ **then**
13: $y = (1, 0)$
14: **else**
15: $y = (0, 1)$
16: **end if**
17: The controller can generate a set of data (TP, Y) for current paths combination $c_{..k}$
18: Every switch conducts a signaling process, sends the link weight to the controller;
19: **end for**
20: Controller trains all the CNNs with the obtained training data

After obtaining all the paths combinations, the controller constructs the CNNs as shown from Steps 2 to 4 in Algorithm 5. Here, we can use $CNN_{..k}$ to denote the CNN for

paths combination $c_{..k}$. Since the CNNs will be utilized for routing in the running phase, we need to get some data to train our CNNs in the initial phase. As mentioned previously, in every δ, each switch records its traffic patterns including the traffic generation rate and the remaining buffer size as shown in Step 8 in Algorithm 5. To judge whether the path is congested or not, every switch also needs to calculate and record the delay when receiving packets destined for itself. Therefore, in each t_u, every switch can calculate the delay values for the paths destined for itself. Then, all the switches upload the information including the traffic patterns and delay values to the controller according to Step 9. With these data, the controller can form a matrix TP representing the traffic patterns of all switches, which will be used as the input of the deep CNNs. Also, after numerous cycles running the conventional routing protocols, the controller can obtain multiple sets of delay values for each paths combination in C with different traffic patterns. Therefore, for each paths combination, the central controller can judge whether it is congested or not according to some pre-defined standard, for example, the threshold of the congestion can be two times of the minimum delay value. As shown in Steps 12 to 16, if the delay of any chosen path p_{ij} exceeds the threshold, it means the chosen paths combination $c_{..k}$ is congested, we can get one set of training data for $CNN_{..k}$: the input is the traffic patterns in previous update interval and the output is $(1, 0)$, as we can only use the traffic patterns in last t_u to decide the paths in next t_u.

4.3.2 Running Phase

After getting initialized, the CNNs will be applied for routing in the running phase to replace the traditional routing protocols. Moreover, since we utilize a real-time learning strategy as mentioned in Sec. 4.2, the CNNs in our proposal will be periodically retrained with real-time data. Therefore, this phase can consist of three parts as shown in Fig. 4.4: data collection, routing judgement, retraining CNNs, which will be discussed next.

4.3.2.1 Data Collection

Besides forwarding packets all the time, as shown in Steps 10 to 13 in Algorithm 6, every switch in the data plane keeps collecting the data of traffic patterns in each δ as the input of CNNs. The switches also calculate and record the delay values when receiving packets. And during each update interval t_u, every switch uploads these data to the central controller, and the central controller addresses the data and utilizes for two purposes. First, the traffic patterns in the previous path update interval are adopted as the input of CNNs to choose the path for next t_u. Second, the controller utilizes the traffic patterns and delay values for retraining the CNNs in next t_r. For example, if the delay of paths combination $c_{..k}$ exceeds the threshold when the traffic pattern is TP, then the

Algorithm 6 Using CNNs to Choose Paths Combination during Each t_u
Input: CNNs
Output: (x, y) (x represents the traffic patterns, TP, y denotes the labels of each paths combination)

1: **for** $p = 1, ..., n_1$ **do**
2: **for** each paths combination $c_{..k}$ **do**
3: Controller conducts a forward propagation process by inputting TP to $CNN_{..k}$ and output Y.
4: **if** Y is $(0, 1)$ **then**
5: The paths combination $c_{..k}$ is chosen.
6: **end if**
7: break
8: **end for**
9: Controller uses the chosen paths combination to generate the rules and installs the rules on the corresponding switches
10: **for** each traffic patterns recording interval δ **do**
11: Every switch forwards packets according to the installed rules, records the traffic patterns, and calculates the delay for each received packet.
12: Every switch calculates the delay of the paths destined for itself, sends the traffic pattern and path delay values to the controller.
13: **end for**
14: The controller constructs the input traffic patterns of the CNN, TP.
15: **if** the delay for any path of OD_{ij}, $d_{i,j} > threshold$ **then**
16: $Y = (1, 0)$
17: **else**
18: $Y = (0, 1)$
19: **end if**
20: The controller can generate a set of data (TP, Y) for current paths combination $c_{..k}$
21: **end for**

controller gets one set of data for retraining $CNN_{..k}$, and the input and output are TP and $(1, 0)$, respectively.

4.3.2.2 Routing Judgement

Since it receives the traffic patterns from all the switches during the whole packet forwarding process, the controller can organize the traffic patterns in the form of CNN's input as explained in Sec. 4.2. Therefore, at the beginning of the k^{th} update interval, t_u, the traffic pattern data of $(k-1)^{th}$ update interval are utilized as the input to CNNs to determine whether the paths combination will lead to congestion or not. As the paths combinations are saved in the descending priority order, the controller will judge these paths combinations one by one. As shown from Step 2 to Step 8 in Algorithm 6, the

Algorithm 7 Retraining the CNNs
Input: (X, Y) (X represents the traffic patterns, TP, Y denotes the labels of each paths combination)
Output: Updated CNNs

1: **for** each paths combination $c_{..k}$ **do**
2: Controller trains $CNN_{..k}$ with its training data (X, Y)
3: **end for**

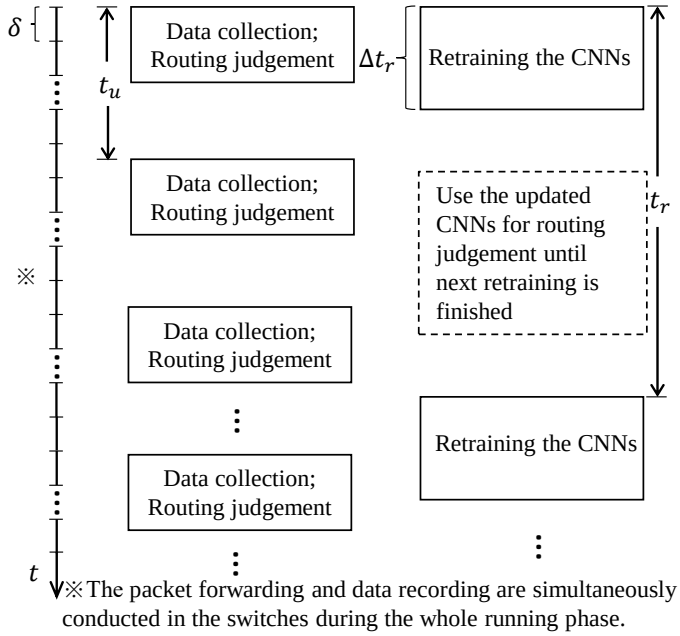


Figure 4.4: The process in the running phase.

judgement process for each paths combination can be fulfilled by conducting a forward propagation of the corresponding CNN with the traffic patterns as the input. The detailed computation process has been introduced in Sec.2.2.2.2. As shown in Steps 4 to 7, if the result of $CNN_{..k}$ is $(0, 1)$, which means the paths combination $c_{..k}$ will not be congested, then the controller chooses the paths combination for routing in the next t_u and the remaining paths combinations will not be considered since they have lower priorities. It should be noted that the computation for judging each paths combination is simple and the time cost is negligible compared to δ. In this chapter, we do not consider the delay caused by the judgement process.

4.3.2.3 Retraining Phase

As mentioned in the previous section, in our proposal, the routing strategy keeps learning from the experiences, which is fulfilled by periodically updating the weight matrices with the newly generated traffic trace during the packet forwarding process shown in Algorithm 7. The retraining of the CNNs in the initial phase is nearly the same as that in the

Table 4.1: The parameters of the considered CNN structure

input layer		*conv*1		*conv*2		*fc*1		*fc*2		output layer	
width	3	width	3	width	3	#node	100	#node	15	#node	2
		height	3	height	3						
		channel	2	channel	2						
height	10	stride	1	stride	1	active	relu	active	relu	active	softmax
		padding width	1	padding width	1						
		padding height	1	padding height	1						
channel	2	#filter	20	#filter	30	initialize	xavier	initialize	xavier	initialize	xavier
		active	relu	active	relu						
		initialize	xavier	initialize	xavier						

initial phase. And compared with the training in this phase, the retraining is based on the previous training, which means that the weights of every CNN have reasonable values and the training has less iterations. To more clearly explain the two training process in these two phases, we can think that in the initial phase, the CNNs are trained to get the basic knowledge about how to choose the paths combination, while in the running phase, the CNNs are trained to update and strengthen their knowledge. As the retraining is a time-consuming process, here, we can assume that the time cost for the retraining process is Δt_r. Then, as shown in Fig. 4.4, before the retraining process is finished, the controller still utilizes the CNNs before retraining to judge the paths combinations while the updated CNNs can be adopted once the retraining process is finished.

4.4 Complexity Analysis

In this section, we make some analysis about the time and space complexity of our proposal. As we mentioned earlier, in our proposal, the switches record their traffic patterns and the delay values of different paths, which are sent to the controller for the training and running of the considered CNNs in the controller. Since all the training and running tasks are conducted by the controller, most of the computation and storage costs happen in the controller. We first focus on the controller part and then the switches.

In the controller, the deep learning related computations cost most of the resource. The values of computation and storage costs depend on the architectures of the CNN, which can be only decided by trial and error. Therefore, we focus on the time and space complexity analysis. According to the analysis in [9], the computation overhead of training and running one CNN at a time is dependent on the number of nodes in its input layer. Specifically, if n denotes the number of switches in the network and each path update interval equals k traffic pattern recording intervals, the input layer of each CNN consists of kn units. Then the time complexity can be denoted as $O(k^2n^2)$. As

k is usually negligible compared with n in our considered SDCS, the time complexity can be simplified as $O(n^2)$. The traditional shortest path strategies, such as the Dijkstra algorithm, also have a time complexity not less than $O(n^2)$. Therefore, the computation cost of training one CNN with one set of data is comparable to conventional algorithms. The most computation-consuming part of training a deep learning architecture is that massive data need to be adopted to repeat and iterate the training. However, in our considered proposal, we consider the real-time training manner, meaning that each time, we just use a few sets of accumulated data to retrain several CNNs. Compared with the traditional method which trains all the CNNs one time, the increase of computation cost is still limited. This analysis is also applicable to the storage cost. Therefore, the space complexity to save the recorded traffic patterns and delay values is $O(kn) \approx O(n)$.

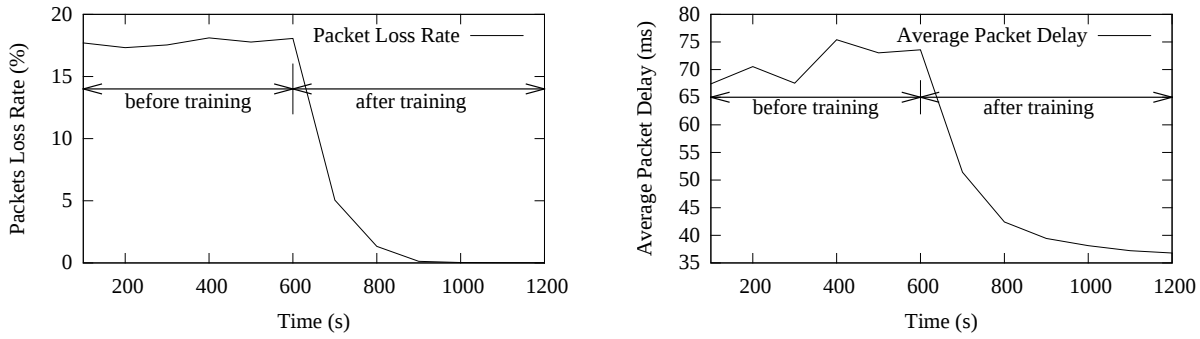
Compared with the training process, to run the deep learning based proposal, it costs much less computation and storage resource since we just need to utilize one set of traffic pattern to conduct the forward propagation of several CNNs. And this process has no iteration or repetition. Therefore, the time and space complexity are just $O(n^2)$ and $O(n)$, respectively.

In our proposal, the switches have the same operations during the training and running periods, meaning the same computation and storage costs. Moreover, the switches do not need to conduct any deep learning related computations and are just responsible for recording their own traffic patterns and delay values of the paths from the source nodes to themselves. Therefore, we just need to analyze the storage complexity, which is $O(k + n) \approx O(n)$. Thus, it can be found that the storage cost for the switches is reasonable.

According to the above analysis, we can find that the deep learning technique is related to more computation and storage costs compared with traditional strategy. However, our considered online training manner can not only increase the self-adaptation of the CNNs to the traffic changes, but also significantly alleviate the costs for the controller. Therefore, in the chapter, we consider the controller consisting of the CPU and GPU pools, which can accelerate the computation process.

4.5 Performance Evaluation

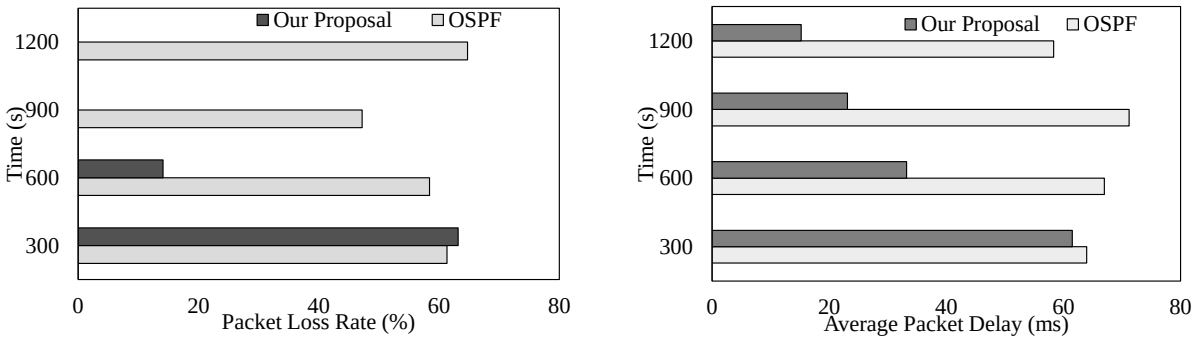
This section evaluates our proposal in terms of network performance through the simulation based on C++ [9]. Since all the computation is conducted on a workstation with Intel Core i7-6900K CPU, 64GB RAM, and Nvidia Geforce TitanX GPU, it is reasonable to restrict the simulation to a small size network. Therefore, we consider a scenario of 3×3 wireless heterogeneous network as the data plane and a PC as the central controller which has been shown in Fig. 4.1. We consider that the controller manages the switches



(a) The packet loss in the considered SDCS

(b) The average packet delay in the considered SDCS

Figure 4.5: The network performance before and after training.



(a) The packet loss in the considered SDCS

(b) The average packet delay in the considered SDCS

Figure 4.6: The network performance comparison between the conventional routing protocol and our proposal in terms of packet loss rate and average packet delay.

in the form of out of the band. Therefore, independent connections between the central controller and the switches should be established for the transmission of control messages. And the congestion in the data plane does not affect the transmission of control messages. It is worthwhile to note that this scale of simulation is sufficient enough to demonstrate that our proposal outperforms conventional routing protocols such as IS-IS, OSPF, and RIP. In this network, the switches S_1, S_2 and S_3 generate packets destined for S_8. In order to increase the spectral efficiency, we consider a WLAN system that simultaneously uses multiple bands such as 2.4GHz and 5GHz [84, 85]. The link bandwidth and the buffer size of each switch are set to 480Mbps and 10MB, respectively. In our simulation, the sizes of each data packet and signaling packet are 1kb and 512b, respectively. The time slot (δ) in the simulation is 1s and the path updating interval (t_u) consists of only 1 time slot while the retraining time interval (t_r) consists of 100 time slots.

In our simulation, the structure of CNN after training and the parameters have been shown in Table 4.1. We can find that each CNN consists of 2 convolutional layers (denoted

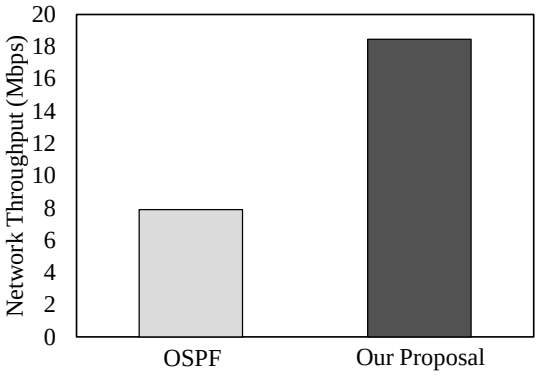
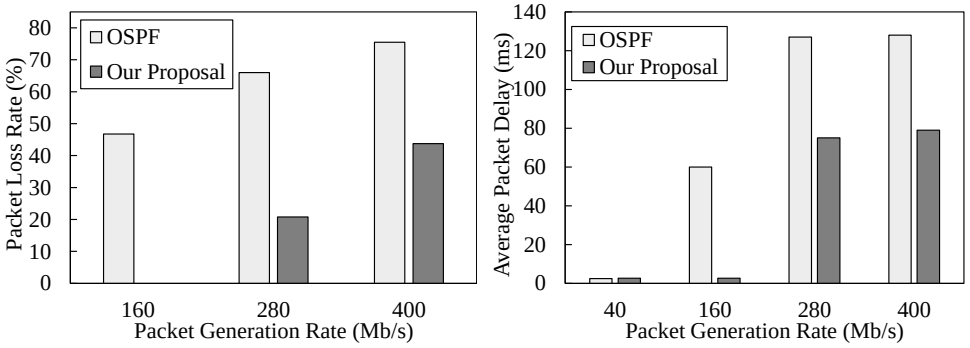


Figure 4.7: The throughput comparison in the considered SDCS.

as $conv1$ and $conv2$, respectively) and 2 fully connected layers (represented as *fc1* and *fc2*, respectively) as well as the input and output layers. Since the size of the input layer is limited because of the considered network size, the pooling layer is not necessary in our CNNs. In the input layer, we adopt the packet generation rate and remaining buffer size of each switch as two channels of the CNN. In each channel, every switch records the data in last 10 updating intervals. In $conv1$, we have 20 filters while $conv2$ has 30 filters, and the size of each filter is 3×3. The padding parameter and the step size are both 1. The two fully connected layers consist of 100 nodes and 15 nodes, respectively. We consider Xavier initialization [86] to set the initial values of all weights and biases. The accuracy rate of this CNN structure after training reaches 98.7%, which is sufficient for our proposal.

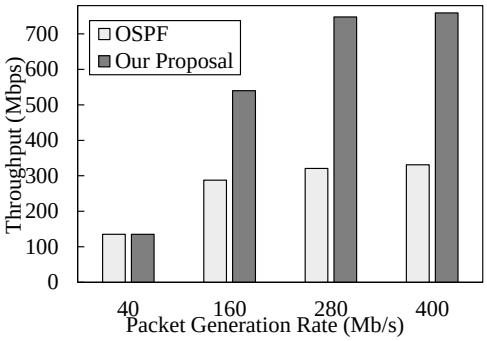
In the first simulation, we compare the network performance before and after utilizing our deep learning-based proposal. The packet generation process in three source switches satisfies the Poisson distribution. And the whole simulation lasts about 1,200s while the initial phase and running phase both share half of the simulation. The average packet generation rate is 180Mbps. In the initial phase, the central controller runs the conventional routing protocols to generate data for training the CNNs. Then, the CNNs are adopted in the controller to choose the paths combinations in the running phase. Figs. 4.5a and 4.5b show the network performance in terms of packet loss rate and average packet delay. In the two figures, we can find a significant decrease after the application of trained CNNs into routing, meaning that our proposed CNNs learn to avoid the congested paths from previous experiences. Moreover, the values of packet loss rate and average packet delay are still decreasing until reaching the lower bound. This indicates that our proposed CNNs are retrained periodically to learn the new experience, which helps to increase its knowledge on routing and improve the SDCS performance.

To compare our proposal with conventional routing protocols, we consider the network traffic patterns generated by the switches S_1, S_2 and S_3 are similar to Fig. 4.2a. We consider the OSPF algorithm as a benchmark method. And in the simulation utilizing



(a) Comparison of packet loss rate for OSPF and the proposed deep learning system.

(b) Comparison of average packet delay for OSPF and the proposed deep learning system.



(c) Comparison of aggregate throughput for OSPF and the proposed deep learning system.

Figure 4.8: Comparison of SDCS performance under different packet generation rates in our proposal and the bencmark methods (OSPF) in terms of packet loss rate, average packet delay, and throughput.

our proposal, the initial phase lasts a short time to get a few data for training the CNNs. To increase the complexity, the start time, cycle time, and the amplitudes of the traffic pattern curves are randomly set for the three switches. Figs. 4.6a and 4.6b compare our proposal and conventional routing protocols in terms of the packet loss rate and the average packet delay. From both results, we can find that the performance of our proposal and OSPF are nearly the same at $t = 300s$, which means that the CNNs have acquired the knowledge on routing after a few times of training. After that, the accuracy of CNNs in our proposal gets continuously improved through the periodical retraining. Therefore, the performance of our protocol outperforms the conventional routing protocol in terms of both the packet loss rate and average packet delay after $t = 300s$. Moreover, our proposal keeps improving the network performance while the performance of OSPF remains nearly unchanged. This happens because the periodical retraining increases the CNNs' knowledge for better routing while the conventional routing protocol is based on fixed rules. To further compare our proposal with the conventional routing protocol,

Fig. 4.7 demonstrates the network throughput of our proposal and OSPF. It can be noticed that the network throughput of our proposal is nearly twice than that of OSPF, which can further demonstrate the advantages of our proposal over the conventional routing protocol.

In order to further evaluate the performance of our proposal under varying network environments, we conduct the simulations with the increasing packet generation rate of every source switch from 40Mbps to 400Mbps and compare the packet loss rate, average packet delay, and network throughput of our proposal and conventional routing protocols (OSPF) as shown in Fig. 4.8. It should be noted that the packet loss rates with the two routing strategies are both 0 when the packet generation rate is 40Mbps. In Fig. 4.8a, it can be clearly found that the network running the conventional routing algorithm gets congested when the packet generation rate is just above 160Mbps while our proposal can still successfully transfer all the packets when the packet generation rate is 160Mbps. When the packet generation rate is 280Mbps and 400Mbps, the SDCS using our proposal also gets congested which can be explained by the switches' limited buffer size and link bandwidth. On the other hand, the result can still demonstrate that compared with the conventional routing protocol, the proposed CNNs can make the better routing decision for alleviating the traffic congestion.

4.6 Summary

SDN has been viewed as the paradigm of next generation network due to its flexibility and conciseness. However, current SDN structure mainly utilizes conventional routing protocols which are based on fixed rules and lacks the intelligence to learn from previous experiences. This can lead to the repetition of wrong decisions when similar traffic patterns happen. The inaccurate path decision results in the network congestion, which leads to further performance deterioration. In this chapter, we propose a deep learning based routing strategy which utilizes CNNs to choose the paths combinations according to the network traffic trace in an online fashion. This strategy can not only better choose the paths combinations according to previous network trace, but also keeps improving its performance through continually learning from previous experience. Analysis shows that our proposal can avoid the congested paths and balance the network traffic, resulting in the significant improvement of packet loss rate and average packet delay in the SDCS. Thus, it can be concluded that our proposal outperforms conventional routing protocols in SDCSs.

Chapter 5

Value Iteration based Deep Learning Architecture for Routing in Dynamic Networks

5.1 Introduction

The above two chapters propose two deep learning based routing strategy for the backbone network and heterogeneous network. And it can be clearly found that the deep learning can significantly improve the traffic control performance for these static networks. However, there exist various dynamic scenarios in practical networks, which have changing topology. Therefore, the proposed deep learning architectures in Chapters 3 and 4 cannot be utilized to predict the paths for the dynamic networks, since they only consider the node information instead of the whole topology. In this chapter, we propose a deep reinforcement learning based strategy which utilizes the Value Iteration Architectures (VIAs) to make the routing decisions. Different from the proposals in Chapters 3 and 4, the input of the adopted VIA in this chapter contains the information of the whole network graph including the nodes and links instead of only the nodes' traffic patterns. Through training with data from various networks, the considered VIAs can learn the routing policy for networks of the same sizes. Even when the network links change, the trained VIA can still predict the paths with a high accuracy rate, which is different from our previous proposal focusing on static networks [9]. Moreover, we consider the deep reinforcement learning manner to train the VIA, overcoming the difficulties in obtaining the labeled data. To adopt the proposal, we analyze the time complexity and consider the HCP as the computation platform. We also analyze the computation consumption and the running time cost in different deployment manners. To illustrate the performance of our VIA based Deep Learning (VIADL) proposal in networks with changing adjacency

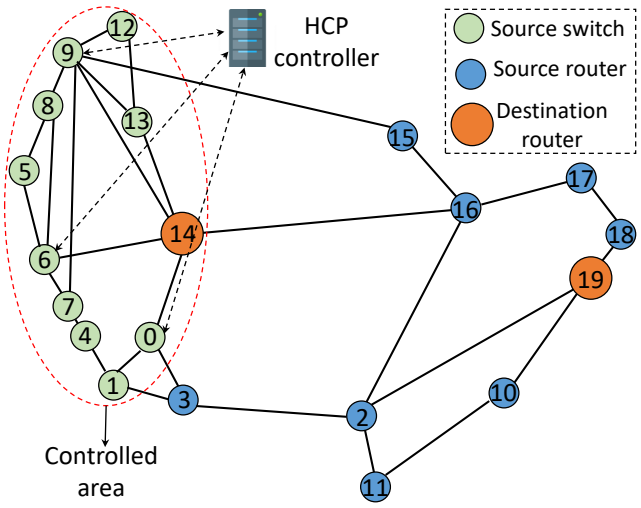


Figure 5.1: The considered network topology.

matrix, we choose the existing supervised learning based method as the benchmark, and make a comparison considering the link failures.

The remainder of this chapter is structured as follows. Sec. 5.2 explains the problem studied in this chapter. Sec. 5.3 introduces some basic knowledge about the Markov Decision Process (MDP), the value iteration method, and the CNNs. Sec. 5.4 discusses our proposed deep reinforcement learning based strategy as well as the VIAs. The complexity analysis of our proposal is conducted in Sec. 5.5. Sec. 5.6 evaluates the performance of our proposal. In this section, we firstly analyze how to deploy our proposal. According to the characteristics of our proposed intelligent routing strategy, we compare the training computation consumption and the running time cost for networks with different percentages of centralized controlled switches. Then, we analyze the network performance of our proposal as well as the supervised learning based method in the network scenarios when some links fail. Sec. 5.7 summarizes the whole chapter.

5.2 Problem Formulation

As we mentioned above, the deep learning based traffic control strategies in Chapters 3 and 4 are based on the node information to predict the paths. Therefore, the characterized input of most utilized deep learning architectures is just the information of network nodes. For instance, the DBAs utilized in Chapter 3 adopt the traffic patterns of all nodes and next node as the input and output, respectively. After being supervised trained with massive labeled data, the parameters of the DBAs can represent the relationship between the traffic and next nodes. Then, the trained DBAs can efficiently predict the traffic pattern for static networks. However, once the connections among network nodes change, the prediction accuracy of the trained DBA deteriorates sharply since the network links

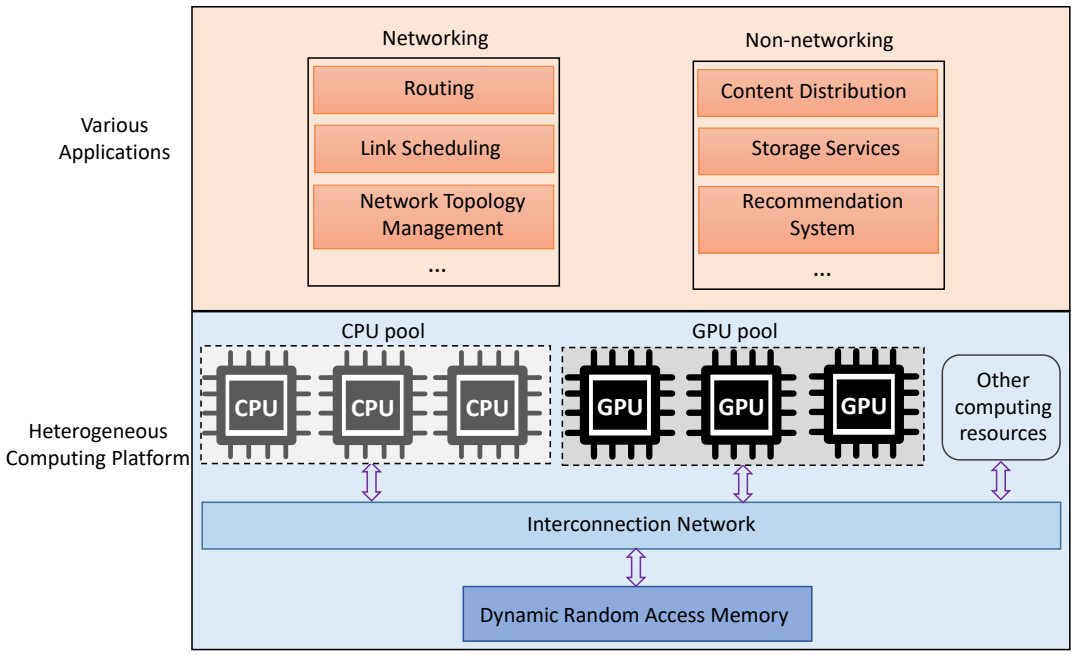


Figure 5.2: The Heterogeneous Computing Platform (HCP) and the applications built on it.

also affect the traffic pattern [87]. One way to recovery the prediction accuracy is to retrain the DBA with labeled data from the new network scenarios, which leads to significant delay and computation overhead. However, the training data cannot cover all network scenarios in various shapes, which means that the DBA needs to be retrained once the network adjacency matrix changes. Furthermore, since the input layer of the adopted deep learning architectures in most proposals depends on the network size, once the network size changes, these architectures need to be reconstructed to fit the new scenarios [88].

If we analyze the problem mentioned above, we can find several limitations in the strategies described in Chapters 3 and 4. Firstly, the network dynamics consist of too many cases to be covered by labeled data. Therefore, other training manners should be considered such as the deep reinforcement learning which spares some probabilities to explore the unvisited cases [20, 89, 90]. Secondly, the common usage to analyze the network problem is to consider a network as a graph, $\mathcal{G}$, which consists of vertex $\mathcal{V}$ and edges $\mathcal{E}$, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ [91]. And the values of different metrics of $\mathcal{V}$ and $\mathcal{E}$ are deeply related to each other. Therefore, it is inaccurate to just consider the node information or the link information to make predictions and decisions in networks with changing topologies. For example, the existing SP algorithms such as the Dijkstra Algorithm all consider the adjacency matrix (representing how the nodes are connected) as well as the link weights to decide the paths [9]. Another limitation is that the input layer of the adopted deep learning architecture is dependent on the network size, meaning that new deep learning architectures are necessary for the networks in different sizes.

Considering current deep learning architecture cannot have dynamic configurations, in this chapter, we just focus on the dynamic networks with fixed sizes. To predict the paths in networks with changing links, we propose a deep reinforcement learning based strategy which considers the node information as well as the adjacency matrix. In the strategy, we assume there exists an agent in responsible of finding the best path between every source node v_s and every destination node v_d in the network. Fig. 5.1 shows the considered network topology which is from Internet Topology Zoo [92]. It should be noted that we make some slight revisions on the topology named "Electric Lightwave" to make it more clear. As the MDP model is usually adopted to solve the reinforcement learning problem, we regard the process of the agent moving from the source node to destination node as an MDP [89]. The considered MDP model consists of several elements: the environment e denoting the network, a set of states $s \in S$ representing the positions of the agent, a set of actions $a \in A$ meaning moving to one neighbor node, the reward r resulting from the action [89], and the transition probabilities $P_{s',s,a}$ representing the probability of moving to next state s' from current state s when taking action a. As we know, the network routing for graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an NP-hard problem which needs to combine multiple vertexes in a weighted graph to a path p^* with the minimum weights as shown in Equation 5.1 [87, 93].

$$p^* = \{v_0 v_1 ... v_n | v_k \in \mathcal{V}, k = 1...n, min \sum_{k=0}^{k=n-1} w_{v_k v_{k+1}}\}, \tag{5.1}$$

where $w_{v_k v_{k+1}}$ is the weight value of the link between vertexes v_k and v_{k+1}. $n+1$ is the assumed total number of nodes in the chosen path. Since every step, after taking action meaning moving to one of the neighbor nodes, the agent can receive an instantaneous reward. In the deep reinforcement learning model, the routing is regarded as the process of the agent moving from the source node to the destination node for maximizing the total rewards. The total reward can be shown as in Equation 5.2 [74].

$$total\_reward = r_1 + r_2 + \cdots + r_n, \tag{5.2}$$

where we assume that r_k $(k = 1...n)$ is the reward at the k^{th} step. Since the agent takes actions according to the expected reward from the environment which is still affected by many factors, it is common to use a discount factor γ $(0 \leq \gamma \leq 1)$ to weight the rewards at different steps. The total discounted reward, R, is as follows [74],

$$R = \sum_{k=1}^{n} \gamma^{k-1} r_k. \tag{5.3}$$

Then, if we assume the reward is linearly proportional to the distance between the

current node and the destination, then r_k can be calculated according to Equation 5.4.

$$r_k = \begin{cases} -\rho\|n_k - n_d\|, & n_k! = n_d \\ r_d, & n_k = n_d \end{cases} \tag{5.4}$$

where n_k is the node at the k^{th} step and n_d is the destination node. $\|n_k - n_d\|$ is the distance between the nodes n_k and n_d. ρ is a positive weight. r_d is the reward when the destination is arrived at and its value is a positive constant.

The routing design is finally modeled as the MDP which could be solved with the reinforcement learning method. Another problem worthy of note is how to deploy these deep learning based methods in networks [9, 73]. As the deep learning consumes more computing resources than conventional methods, if we want to apply the intelligent routing strategies, it is necessary to redesign the network hardware architectures or adopt the existing high computing platforms in the network. Moreover, the time complexity of the intelligent strategies should be minimized to speedy the algorithm convergence and reduce the computation delay. Besides the intelligent routing, an increasing number of network services and other network management algorithms utilize the technique of deep learning. It is impossible to deploy separate computing platforms for every application, which also means a waste of resource. Since the deployment manner affects the computation performance as well as the network performance, it is necessary to analyze the best way to deploy the deep learning based models. To address this problem, we adopt the SDN technology introduced in Chapter 4. Then, the deep learning based proposal is conducted by the controller consisting of various computing resources including the CPU, GPU, and some other hardware. For the controller part, we consider that the HCPs as shown in Fig. 5.2 conduct the computation of routing, link scheduling, and other network management works. Thus, the flexibility of network management can get significantly improved since the upgrade of some management methods can be fulfilled by updating the corresponding applications [29, 94]. Moreover, to further improve the usage of the computing resources, many non-networking based services can be also conducted by the HCPs, such as the content distribution, storage services, as well as the recommendation systems [95].

5.3 Preliminaries

5.3.1 Markov Decision Process (MDP)

MDP is usually utilized to model the reinforcement learning. In our considered model, the agent needs to choose the next node for the destination at the current position. As

we know, at each step, the number of actions the agent can take is equal to the number of nodes connected to the current position. And choosing different neighbors, the agent can obtain various rewards from the environment. The goal of the MDP is to find the best way to maximize the accumulated rewards R as in Equation 5.3 [96]. The way by which the agent acts is termed as a policy π, which maps the state to action. And the value of $\pi(s, a)$ denotes the possibility of taking action a in the state s. In the deep reinforcement learning, we usually adopt the value function $V^{\pi}(s)$ to describe expected reward value of the state s and it is equal to the expected accumulated reward for the agent starting from s and following policy π as shown in Equation 5.5.

$$V^{\pi}(s) = E^{\pi}[\sum_{k=1}^{n} \gamma^{k} r(s_k, a_k) | s_0 = s], \tag{5.5}$$

where s_k and a_k represent the state and action at the kth step, respectively.

For the convenience to deduce the value function, it is common to define a state-action value function $Q^{\pi}(s, a)$ to describe the expected return when starting from the state s, taking action a, and following the policy π as shown in Equation 5.6. The relationships between the value function $V^{\pi}(s)$ and the state-action value function $Q^{\pi}(s, a)$ are described in Equations 5.7 and 5.8.

$$Q^{\pi}(s) = E[R_k | s_k = s, a_k = a], \tag{5.6}$$

$$V^{\pi}(s) = \sum_{a} \pi(s, a) Q^{\pi}(s, a), \tag{5.7}$$

$$Q^{\pi}(s, a) = \sum_{s' \in S} P_{s', s, a} (R^{a}_{ss'} + \gamma V^{\pi}(s')), \tag{5.8}$$

where $R^{a}_{ss'}$ is the immediate reward after taking action a, transmitting from the state s to s'. Since there is usually one path which is better than or equal to other paths, therefore, the routing path construction process is to find the optimal policy $\pi*$, which returns the maximum rewards $V^{*}(s)$.

$$\pi^{*} = argmax_{\pi} V^{\pi}(s). \tag{5.9}$$

According to Equation 5.7, if we know the optimal $Q^{*}(s, a)$, the optimal policy can also be extracted by choosing action a that maximizes $Q^{*}(s, a)$ in the state s.

$$\pi^{*}(s) = argmax_{a} Q^{*}(s, a). \tag{5.10}$$

Algorithm 8 Value Iteration

Input: K is the number of maximum iterations, θ is a threshold, $\theta > 0$
Output: $\pi^*(s)$: the optimal policy

1: Initialize $V_0(s)$ to arbitrary values
2: **repeat**
3: $k \leftarrow k + 1$
4: **for** each state s **do**
5: **for** all $a \in A$ **do**
6: $Q(s,a) \leftarrow \sum_{s' \in S} P_{s',s,a}(R^a_{ss'} + \gamma V^{\pi}(s'))$
7: **end for**
8: $V_k(s) \leftarrow max_a Q(s,a)$
9: **end for**
10: **until** $k \geqslant K$ or $\forall s \; |V_k(s) - V_{k-1}(s)| < \theta$
11: **for** each state s **do**
12: $\pi^*(s) = argmax_a Q(s,a)$
13: **end for**

5.3.2 Value Iteration

The methods to search the optimal policy in the reinforcement learning consist of the random policy search, genetic algorithms, policy iteration, value iteration, and Q-learning. Since the action space for network routing problem is very large, the random search method does not work well. And genetic algorithms cannot guarantee an optimal policy [97]. In this chapter, we consider the value iteration method which computes the optimal state value function by iteratively improving the estimate of $V^{\pi}(s)$ [96].

According to the relationship between $V^{\pi}(s)$ and $Q^{\pi}(s,a)$ as shown in Equations 5.7 and 5.8, if the transition probabilities $P_{s',s,a}$ and the immediate reward of every step are known, the optimal value can be obtained through repeatedly updating the values of $Q(s,a)$ and $V(s)$ until convergence, which is named "value iteration". As shown in Algorithm 8, we firstly initialize $V_0(s)$ to an arbitrary value as in Step 1. Then, in every state, we calculate the value of $Q(s,a)$ for all actions and assign the maximum $Q(s,a)$ to $V(s)$. We repeat Steps 4-9 until the maximum number of iterations have finished or the difference of $V_k(s)$ in two adjacent iterations is less than a given threshold. Finally, we record the actions which result in maximum values of $Q(s,a)$ at all states to construct the optimal policy π^* [97, 98].

5.4 Design of the Deep Reinforcement Learning Based Routing Strategy

In this part, we discuss the proposed deep reinforcement learning based routing algorithm and the deployment of the proposed routing strategy. The utilized deep learning archi-

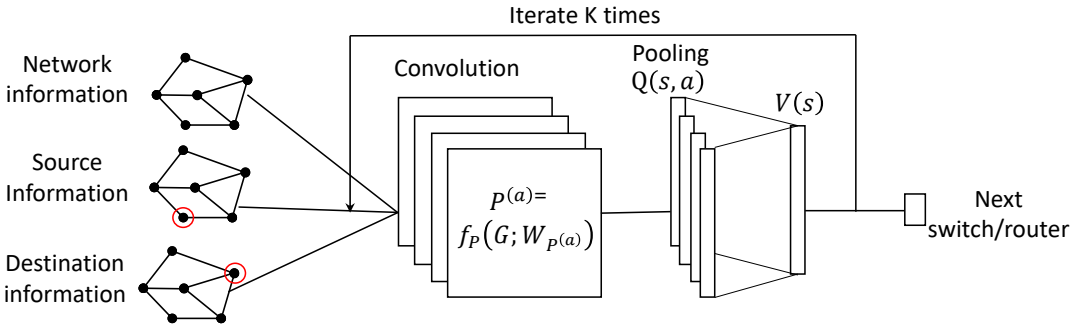


Figure 5.3: The proposed Value Iteration Architecture (VIA).

tecture is the VIA which utilizes the CNN to model the value iteration process. And we adopt the episodic Q-learning algorithm to train the VIA [96].

As mentioned in Sec. 5.3.1, the key component of an MDP is the transition matrix P, which can be modeled as a graph convolution operator and parameterized by the graph based kernel function [96]. Then, the goal of training the VIA is to learn the policy of routing for networks in different shapes. The main part of the VIA is the value iteration module shown in Fig. 5.3 which iteratively operates the graph convolution and max-pooling, imitating the value iteration process. We consider the network topology as an undirected, weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. And the number of nodes in the network is N_r. The input of the VIA consists of the graph information, the source, and destination. The graph information contains the coordinates of the nodes which is depicted as $X \in R^{N_r \times 2}$ and the link weights represented by a $N_r \times N_r$ dimensional matrix A_w. The source and destination nodes are encoded as a one-sparse vector consisting of N_r binary elements, respectively. As shown in Fig. 5.3, the transition matrix can be obtained through the graph convolution operation which is depicted in Equation 5.11. The channel number of the convolution operation is equal to the number of actions.

$$P^{(a)} = f_{P^{(a)}}(G; W_{P^{(a)}}), \tag{5.11}$$

where $P^{(a)}$ is the convolution operator in the ath channel. And $W_{P^{(a)}}$ is the weight matrix of the graph convolution operation. G denotes the graph information. After the convolution steps, the values of all actions can be obtained according to Equation 5.8, after which the max pooling step is operated and the action which maximizes the value $V(s)$ is chosen. The value iteration process is modeled by iteratively conducting the convolution and maxpooling steps.

The training of the VIA is based on the episodic Q-learning algorithm which can be referred to [99]. When we input a start node v_0, the agent takes the ϵ-greedy strategy, which means that the probability to choose the neighbor which returns the maximum reward is $(1 - \epsilon)$ while ϵ probability to randomly choose one neighbor node. If the

Algorithm 9 VIA-based Packet Forwarding Algorithm Implemented in the HCP

1: The HCP trains the VIA with the training data using the Episodic Q-learning method; ▷ Training phase
2: **for** each path update round **do** ▷ Running phase
3: Switches and routers send the hello messages to neighbors. And routers upload the neighbor and link information to the HCP;
4: **for** every source-destination pair **do**
5: The HCP utilizes the information from switches to construct the input of the VIA;
6: The HCP adopts the VIA to predict the path;
7: The HCP generates the packet forwarding rules for the source switch;
8: **end for**
9: The HCP installs the generated packet forwarding rules in corresponding switches;
10: The switches forward the packets according to the installed rules.
11: **end for**

destination node is reached or the predefined threshold is reached, the episodes terminate. The training is based on trial and error experience and the backpropagation process updates all the training parameters. Since the goal of VIA is to predict the paths for any network, the training data consist of the information of networks in different topologies with N_r nodes. Different from existing supervised learning or other deep learning methods, the proposed VIA is to learn the path construction policy which can be applied to any source-destination pair. Therefore, the training of the VIA can be conducted only one time and utilized for all nodes in the network. Moreover, once the network links change, the trained VIA can still be adopted for path prediction without retraining. As the training data consists of the information of various networks, the training process is conducted offline through the CPU. On the other hand, the running process is fulfilled through the forward propagation of the VIA, which needs the network real-time information as the input. Therefore, the VIA is periodically utilized to compute the paths for the switches.

As the training and running process of VIA consumes more computing resources including the CPU, GPU, and Dynamic Random Access Memory (DRAM), compared with conventional routing protocols, in this chapter, we attempt to utilize the HCP shown in Fig. 5.2 to conduct the training and running of VIA in our proposal. More specifically, we consider a centralized control manner and the HCP acts as the controller. The routers governed by the HCP are replaced by switches. The HCP keeps a global view of the network and generates the forwarding rules for the switches, while the switches are just responsible for the packet forwarding [29]. At first, the HCP allocates the CPU, GPU, and DRAM to train the VIA before the VIA is utilized for routing as shown in Step 1 in Algorithm 9. Since this process does not need the real-time network information, it is only concerned with the HCP controller. After that, the running phase begins. Similar to the SDN, the switches need to periodically find their neighbors and send the information to

Algorithm 10 VIA-based Packet Forwarding Algorithm Implemented in Every Router

1: The router trains the VIA with the training data using the Episodic Q-learning method; ▷ Training phase
2: **for** each path update round **do** ▷ Running phase
3: Switches and routers send the hello messages to neighbors. The NIC sends the headers of signaling packets to the CPUs;
4: **for** every destination node **do**
5: The CPUs utilize the information contained in the headers of the signaling packets to construct the input of the VIA;
6: The GPUs adopt the VIA to predict the next node;
7: **end for**
8: The CPUs construct the routing table according the prediction results;
9: The router transfers the packets according to the routing table.
10: **end for**

the HCP controller as shown in Step 3. Then, the HCP allocates some CPU resources to address the information uploaded by the switches and construct the input of the VIA as in Step 5. After that, the HCP adopts the trained VIA to predict the paths for every pair of source and destination nodes as shown in Fig. 5.1, which costs much less computing resources compared with the training process. The computation in this step is fulfilled by the GPUs in the HCP. After computing the paths, the HCP generates the packet forwarding rules for switches and installs the rules in corresponding switches shown in Steps 7 and 9. And the switches can forward the packets according to the rules. It can be found that the functions of switches get significantly simplified since the computation work is transferred to the HCP. Moreover, the network management algorithms as well as other network services are installed in the HCP as applications and share the same computing resources. Any upgrade can be fulfilled by updating the corresponding applications, which greatly improve the network flexibility. It can be found that the considered network is similar to the SDN which is the next network paradigm [29]. However, the controller in the conventional SDN is just responsible of computing the packet forwarding strategy. Thus, the considered network in our proposal can be regarded as an extended SDN where the HCP acts as the computing platform for more applications. On the other hand, the switches in our considered network are the same as those in the practical SDN switches.

As mentioned above, the VIA can be trained only once and utilized for predicting the paths for any node. The centralized control manner utilizing the HCP to conduct all the routing computation can minimize the consumption. However, since the cost to replace all the routers with switches is too high, in this chapter, we consider the network as shown in Fig. 5.1 where part of the routers are replaced with switches and controlled by the HCP. For the routers, we consider the GPU accelerated SDRs proposed in our previous work [9] to train and run the VIA for routing. It should be noticed that as the

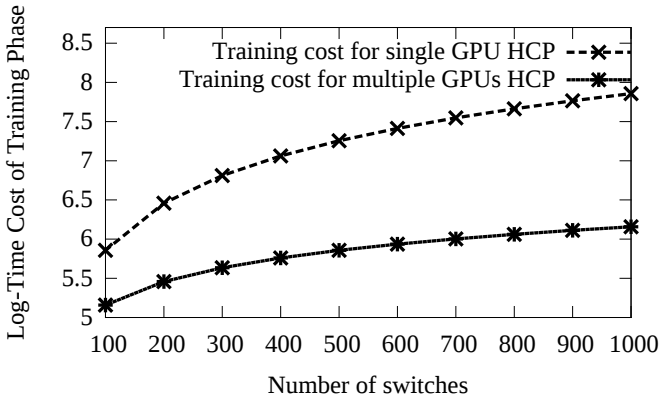


Figure 5.4: The log time cost of training VIA with the single GPU HCP and the multiple GPUs HCP.

proposal in this chapter is different from the strategy in [9], the practical implementation is slightly different. As shown in Step 1 in Algorithm 10, as the same as the centralized control, the training process can be still conducted in an offline manner, which is fulfilled by every GPU accelerated SDR, independently. In the running phase, the NIC in every router sends the headers of the signaling packets to the CPUs as shown in Step 3. Then, the CPUs construct the input matrix of the VIA (Step 5) and the GPUs predict the next node for every destination (Step 6). After that, the routing table can be constructed and the router can transfer the packets according to the routing table.

5.5 Complexity Analysis From the HCP Perspective

After introducing the VIA based routing strategy in Sec. 5.4, in this section, we make some analysis about the time complexity of our proposal. The analysis is based on the number of arithmetical operands as well as the computing resources of HCPs in the network. We first study the time complexity of the training phase and running phase, then the time cost to train and run the VIA with the HCP is discussed.

First, we consider the time cost of the training phase. The training phase can be divided into two parts: the forward propagation and backward propagation which has been explained in Sec. 2.2.1. Since these two parts have the same time complexities which are both quadratic to the number of units in the input layer and following layers, we just analyze the forward propagation process in this chapter. As in deep reinforcement learning, the number of epochs is related to many factors, such as the parameters of the optimizers, the choice of the loss function, as well as the training data, the values of these parameters are usually obtained by trial and error. Therefore, we just assume that the values of these parameters have been already set. And the time complexity considered here is mainly concerned with the MDP as well as the value iteration process, which is still reasonable since the proposed VIA is constructed to model the value iteration process.

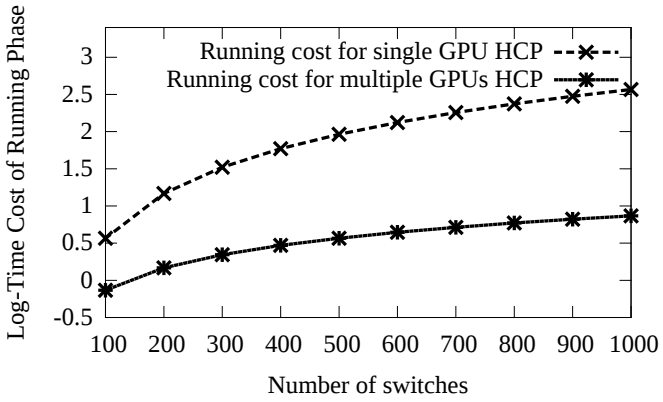


Figure 5.5: The log time cost of running VIA with the single GPU HCP and the multiple GPUs HCP.

Here, we use m and n to denote $\|A\|$ and $\|S\|$, respectively. As Step 6 in Algorithm 8 needs to consider all states, the time complexity is $O(n)$. Thus, the time complexity of one iteration consisting of Steps 4 to 9 is $O(mn^2)$. Considering the number of iterations, the time complexity of Algorithm 8 is $O(Kmn^2)$ where K is the number of iterations, of which the maximum value satisfies the following relationship [100, 101],

$$K^* \leq \frac{B + \log 1/\epsilon + \log 1/(1-\gamma) + 1}{1-\gamma}, \tag{5.12}$$

where K^* is the minimum number of iterations. B is the maximum number of bits required to denote any component of the transition matrix P or the cost of every step. ϵ is the probability to choose the best action in Step 1 in Algorithm 9. In our proposal, all these parameters are constants and assumed to have ideal values. Therefore, we can conclude that the time complexity of the training phase is $O(mn^2)$.

After the training process, the trained VIA is utilized to predict the paths for every source node in the network. As in the running phase, the HCP just conducts the forward propagation process to predict the next node for all source nodes, the HCP can calculate the state-value function as shown in Steps 5 to 7 in Algorithm 8 for all states and then chooses the actions which maximize the values of $Q(s, a)$, which means the value iteration can be conducted only once. Therefore, the time complexity of the running phase is $O(mn^2)$.

Then we consider the time cost to train and run the VIA on the HCPs. As both the training and running process of VIA are concerned with massive matrix computations, it is reasonable to consider the GPUs as the main computation hardware of our proposal. The CPUs are still utilized for some work such as the task scheduling. We assume the number of GPUs adopted to train the VIA is n_g, then the time cost follows the following equation,

$$t_{comp} \propto \frac{O(mn^2)}{n_g} = O(\frac{mn^2}{n_g}), \tag{5.13}$$

where t_{comp} represents the time cost to undertake the computation of training and running of VIAs. In above equation, the value of m is related to the maximum node degree of the considered network and n is related to the number of nodes in the considered network. Therefore, $1 \leq m \leq n-1$, where $m = n-1$ for the network with the star topology [102] which usually exists in access networks. In our considered network, we consider that the maximum number of interfaces of every switch/router is a constant. Thus, the training/running time satisfies $t_{comp} \propto \frac{O(n^2)}{n_g}$. Moreover, the value of n_g is equal to the number of adopted GPUs for computation. If only one GPU is available for the computation task of training and running, then the time cost is $O(n^2)$. On the other hand, if we assume that the number of assigned GPUs is linearly proportional to the number of switches governed by the HCP ($n_g \propto n$), the time complexity can be reduced to $O(n)$ and the time cost satisfies $t_{comp} \propto O(n)$.

According to the above analysis, we conduct the simulation to record the time cost of training and running the VIA with single GPU (the Nvidia Titan X Pascal) for a network with 20 switches. Then, we can theoretically calculate the time cost for networks with 100 to 1000 switches as shown in Figs. 5.4 and 5.5. In Fig. 5.4, we can find that using multiple GPUs to train the VIA can significantly reduce the time cost. And the advantage of multiple GPUs over single GPU can be enlarged with the increase of network topology. Even though the training of the VIA is a time-consuming process according to Fig. 5.4, it can be conducted offline and does not affect the network performance. In Fig. 5.5, we can find the similar enlarging advantage of running the VIA with multiple GPUs over that with single GPU for the increasing network topology. On the other hand, since the running phase is conducted periodically during the packet forwarding process, too long time cost leads to the delay of path update. Therefore, the practical number of switches governed by the HCP should be decided with the network parameters and performance requirement considered. It should be also noted that the unit for the time cost before the logarithm arithmetic is second and the base of the logarithm arithmetic is 10.

5.6 Performance Evaluation

After introducing the proposed VIA, in this section, we analyze the performance of our proposal through simulation based on python and tensorflow [103]. The network topology is shown in Fig. 5.1. Since the simulation platform is a workstation with Intel Core i7-6900K CPU, 64GB RAM, and Nvidia Geforce Titan X GPU, it is reasonable to choose only two nodes, 14 and 19, as the destinations in the simulation, which can still demonstrate the advantages of our proposed proof-of-concept. The link bandwidth is 100Mb/s while the buffer size of all the nodes is set to 12.5MB. The sizes of the data packet and signaling

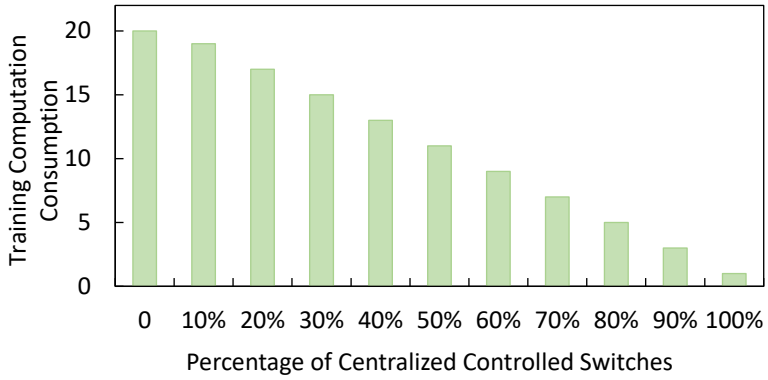


Figure 5.6: Training computation consumption of networks with different percentages of centralized controlled switches.

packet are both set to 1kb. Different from our previous work [9], the considered network in this chapter has no inner nodes or edge nodes and all nodes except the destination nodes act as the data packet sources. And the data packet generation process follows the poisson distribution with an average value of 1.

To clearly illustrate the performance, we choose the supervised learning method as the benchmark and the DBA as the deep learning architecture. The input and output of the DBAs are the traffic pattern and next node, respectively. As all the nodes except the destination nodes generate the data packets, the input of DBAs is the traffic patterns of all nodes instead of just edge nodes in our previous work [41]. Since every DBA just predicts the next node, to construct the whole paths, we need multiple DBAs in the supervised learning method. More specifically, in our considered network as shown in Fig. 5.1, every destination node needs to train and run one DBA for predicting the next node for the other destination node, while all the other nodes need to train and run 2 DBAs for the two destinations. The labeled training data for the supervised learning method come from the networks running the Open Shortest Path First (OSPF) protocol in the considered network. After training, each DBA consists of 5 layers and 20 units in every layer. The activation function for Layers 2 to 4 is the ReLU function, while that of the output layer is the softmax function [103]. The loss function is the cross entropy function [103]. To train the DBAs to minimize the loss, we choose the Adam optimizer [103]. The training data size is 10,000, while every training batch consists of 20 sets of data.

For the VIA, the input consists of the coordinates of all nodes, the adjacency matrix, the link weights, as well as the source and destination nodes. The iteration number K of the VIA is 30. The value of the reward discount γ is 0.99. The value of ϵ in Step 1 in Algorithm 9 is 0.95. The training consists of 200 epochs. The training data consist of 1000 different network topologies with 20 nodes. In this section, we first study the best way to deploy the proposal. Since it is not realistic to utilize switches to replace all the routers in the network with extremely high cost, we consider the deployment of switches

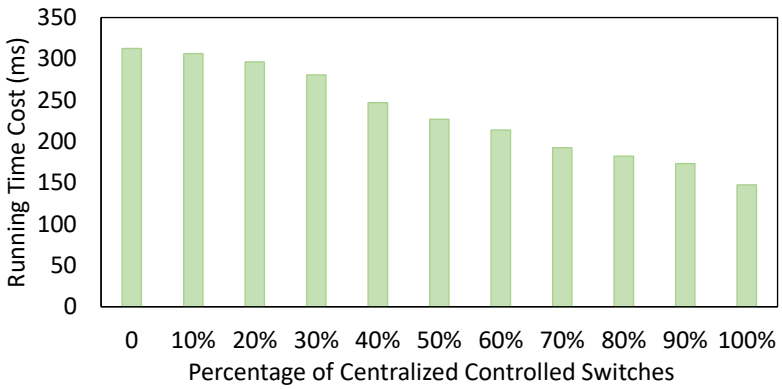
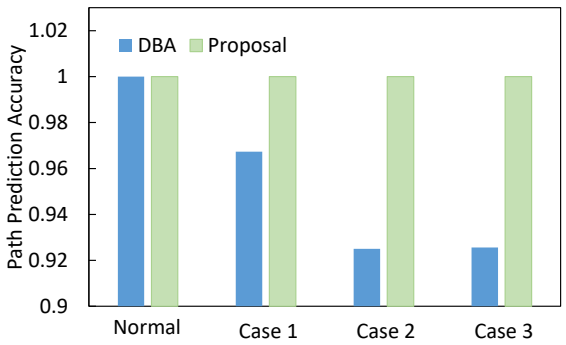


Figure 5.7: Running time cost for networks with different percentages of centralized controlled switches.

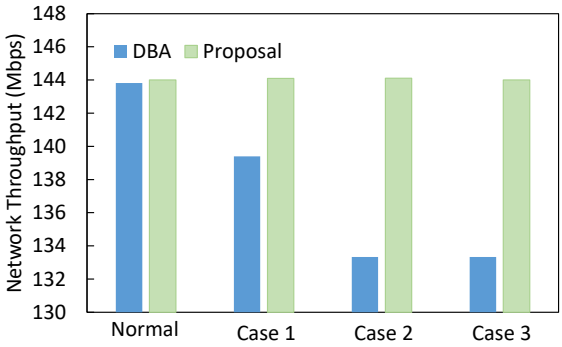
step by step. Then, we compare the training computation consumption and the time cost of our proposal in networks with different percentages of switches. To demonstrate the advantage of our proposal in dynamic networks, we also assume some links failures in the considered network and analyze the performance of our proposal and supervised learning method in terms of the network throughput, the packet successful transfer rate, and the average delay per hop. We also compare the path prediction accuracy rates of two deep learning architectures.

5.6.1 Deployment Analysis

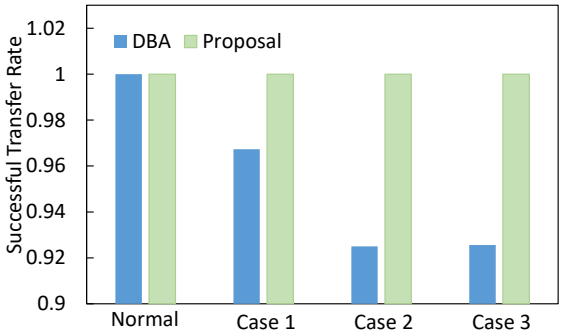
In Sec. 5.4, we consider utilizing the HCP to control the switches in a centralized manner. For the routers not governed by the HCP, they compute the next nodes all by themselves. Therefore, with different percentages of switches, the computation consumption of the VIA based routing varies. In this chapter, we analyze the number of switches counts from 0 to 100% with an interval of 10% among all nodes in the network. In the example shown in Fig. 5.1, the controller trains and runs the VIAs to predict the paths for the network area controlled by the HCP. For the routers in the network, they train and run the same VIAs all by themselves. Therefore, for the networks with different percentages of switches, in the training period, the total number of the conducted training process is various. As each training process trains the same structured VIAs, it needs nearly the same number of training data. Therefore, the total computing resource consumption is linearly proportional to the number of trained VIAs. If we use CR to denote the total training computation consumption of the network, then $CR = f(x)$ where x represents the percentage of switches in the network. We can assume that $CR = 1$ in the network where all routers are replaced with switches ($x = 100\%$), then the training computation



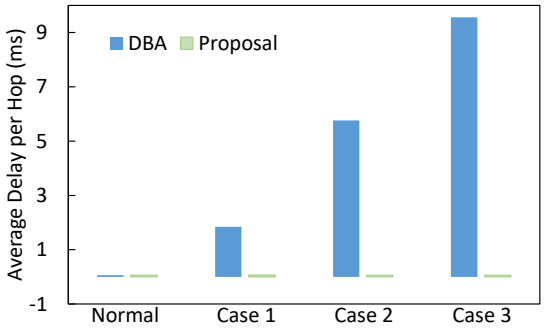
(a) Path Prediction Accuracy Rate.



(b) Considered Network Throughput.



(c) Packet Successful Transfer Rate.



(d) Average Delay per Hop.

Figure 5.8: Considered network performance for different cases.

consumption of networks with different percentages of switches can be described as follows,

$$CR(x) = \frac{f(x)}{f(100\%)}. \tag{5.14}$$

Fig. 5.6 gives the computation consumption for networks with different percentages of centralized controlled switches. It can be found that the computation consumption keeps a nearly linear downward tendency with the increase of centralized controlled switches. Fig. 5.7 shows the time cost for running the networks with different numbers of switches. The time cost also keeps a downward tendency. This can be explained that the running time cost consists of not only the computation cost which is linear to the number of path prediction tasks, but also the preprocessing of the input data which happens only once for one VIA. Therefore, even though the prediction tasks for networks with different numbers of switches are the same, the network with more switches has less data preprocessing tasks. According to the analysis and results, we can conclude that despite of the high cost, the centralized control manner of our VIADL based routing method can minimize the computation overhead.

5.6.2 Performance with Link Failures

In this part, we utilize the trained DBA and VIA to predict the paths for the considered network when some links fail. We assume the link failures happen before the packet transmission process begins. The failed links are randomly chosen, while the failed links for the network running DBA and VIA are kept the same for fairness. Here, we restrict that the failed links are distributed in the whole network instead of being concentrated on one single router/switch which can easily lead to congestion for the node. Also, the failed links do not cause any isolated island in the network. In the simulation, we consider four cases: the normal case with no failed links, Case 1: one failed link between Switch 14 and Router 16, Case 2: another failed link between Switches 7 and 9 on the basis of Case 1, Case 3: on the basis of Case 2, the link between Switches 6 and 8 fails. The packet generation rate of the considered network is 144Mbps. The performance is shown in Fig. 5.8.

In Fig. 5.8a, it can be easily found that the accuracy rates for both strategies in the normal case are 1, which means that the trained DBA and VIA can predict the shortest paths with no error. However, when links fail, the accuracy rate of DBA drops significantly while our proposal can still predict the paths accurately. The reasons for the difference are multi-fold. Firstly, the input of our deep learning structure VIA contains the network link information which is denoted as the adjacency matrix, while that of the DBA is just the traffic information of the network nodes. Therefore, when predicting the paths, the VIA chooses the next nodes just from the neighbor nodes. On the other hand, the potential output of the DBA still contains all the nodes. Secondly, the training data of our proposal are from various networks, whilst those of the DBA can be only from the fixed network. Therefore, we can conclude that our proposed VIA are suitable for different networks with a fixed number of nodes. However, the trained DBA can be only utilized for the networks where the training data is generated. It should be noted that the accuracy rates of Case 2 and Case 3 are nearly the same, which can be explained that the link between Switches 6 and 8 is not utilized even in normal case.

Due to the decreased path prediction accuracy of DBA, the network performance significantly deteriorates as shown in Figs. 5.8b, 5.8c, 5.8d. More specifically, the network throughput drops from about 144Mbps to 133Mbps and the packet successful transfer rate decreases to about 92%. On the other hand, our proposal can transfer all the packets to the destinations in time, which can also demonstrate that the network is not congested. In the simulation, we consider that the packets are saved in the buffer if the router cannot find the accurate next nodes for the packets. Therefore, the average delay per hop for the DBA increases dramatically when some links fail in the network. It can be considered that the network running DBA is congested in Cases 1, 2, and 3. Consequently, we can conclude that compared with the DBA, our proposal can tolerate the network link failures

and predict the paths with high accuracy rate.

5.7 Summary

Since most of current networks are not static and the topology changes are very common due to the node joining or leaving, link or node failures, and so on. Existing deep learning based routing strategies build the intelligent architectures based on the node or link information due to its simplicity to characterize. However, since most of the proposed deep learning architectures are deeply related to the considered network topology, existing node or link information based deep learning strategies can be only adopted in static networks. In this chapter, we propose a deep reinforcement learning based routing strategy which considers the network node information as well as the adjacency matrix. The proposed VIA can repeatedly choose the next node until the destination is reached. Simulation results evaluate the stable performance of VIA when links fail in the network. Compared with the existing supervised learning method, our proposal can build the paths with high accuracy even the link failures exist, leading to stable network throughput, packet successful transfer rate, and packet delay per hop. To adopt the proposed VIADL based routing algorithm, we consider the HCP as the computation platform and utilize the GPUs to train and run the VIA. Analysis of the time complexity shows that the time cost can be significantly reduced with the multiple GPUs. In this chapter, we also study the deployment of the proposal. We mainly consider the HCP to govern part of the network nodes and conduct the computation tasks for the governed switches. The analysis demonstrates the decreasing computation consumption and running time cost when the percentage of switches increases. Since the proposed VIA can still be adopted for networks with the same sizes, our future research will study the intelligent routing strategy for networks with dynamic sizes. As the input layer of the deep learning architecture is dependent on the network size and there is no dynamic deep learning architectures, the proposed VIADL method can only be adopted for networks with fixed number of nodes.

Chapter 6

Conclusion

With the development of deep learning and the computation hardware, the AI technique has been regarded as one of most important technologies to improve users' experience. Inspired by the flexibility and accuracy of deep learning, researchers have made many attempts to adopt this technology to optimize the network performance. To alleviate the increasing traffic overhead, this dissertation considers the deep learning technique to predict the routing paths. Since the deep learning technique consists of so many architectures and three training manners, this dissertation discusses the architecture design for different network scenarios, especially about the characterization of input and output. Moreover, we also analyze the computation overhead of the deep learning based packet transmission strategies and propose novel computation platforms to conduct the algorithms. Even though the deep learning technique concerns more computation overhead compared with conventional routing algorithms, the considered platforms and proposed deployment manners can significantly reduce the computation time. Furthermore, in this dissertation, we focus on not only the static core networks, but also the dynamic networks considering link failures. The performance evaluation demonstrates that proposed deep learning architectures can address the challenges caused by potential link failures. Precisely, our contributions are listed as follows:

1. In Chapter 2, we introduce preliminary knowledge about the training of deep learning, several commonly utilized deep learning architectures and the three training manners. The existing research about the network performance optimization with deep learning is also surveyed in this chapter. After that, we also study the traditional traffic control strategies. It can be clearly found that the traffic control can be cooperatively conducted by different layers. To begin our research, this dissertation focuses on the routing design with deep learning to improve the traffic control performance.

2. In Chapter 3, a deep learning based packet transmission strategy is proposed to improve the traffic control performance of static core networks. In this proposal, the DBA architecture is utilized to predict the next node with the traffic pattern of every

Table 6.1: Comparison of the three deep learning based strategies.

Chapters	Chapter 3	Chapter 4	Chapter 5
Network scenario	Static core network	Static heterogeneous network	Dynamic network
Control manner	Distributed control	Centralized control	Centralized control
Platform	SDR	Computing server	HCP
Architecture	DBA	CNN	VIA
Input	#inbound packets	#inbound packets buffer size	Node coordinates adjacency matrix
Output	Next node	Path combination	Next node
Learning manner	Supervised learning	Online learning	Reinforcement learning

node as the input. To expedite the computation of the intelligent protocol, the GPU accelerated SDR is considered. And the numerical analysis illustrates that the GPU resource can significantly reduce the time consumption. Moreover, the simulation results demonstrate that the proposed deep learning based routing can achieve much better network performance compared with conventional routing protocols.

3. In Chapter 4, an online learning based routing strategy is proposed for the SDCS. In the proposal, the switches in the data plane keep recording the traffic trace and send it to the controller. Then, the controller periodically updates the trained CNNs. It can be clearly found that the accuracy of the deep learning architectures is continuously improved after repeated training process. Moreover, the CNNs get adaptive to the changing traffic patterns due to the periodical training with newly collected traffic trace. Additionally, the simulation analysis demonstrates the advantages of the online learning method and the intelligent routing method is illustrated to outperform the conventional strategy in terms of network performance.

4. In Chapter 5, we discuss the deep learning based packet forwarding for dynamic networks. According to our descriptions in above chapters, it can be easily found that the deep learning architecture design is related to the network topology. To fit the topology changes, we propose a value iteration based deep learning architecture to compute the paths. The considered reinforcement training manner enables the VIA to learn the routing policy independent on the network topology. Therefore, once given the network scenario, the trained VIA can be utilized to predict the paths directly. Moreover, this chapter discusses the time complexity of the proposal. Besides the network performance improvement, the proposed HCP can accelerate the execution of the proposal and the considered deployment manner can further reduce the computation overhead.

Table 6.1 further gives a comparison of the proposed three strategies in Chapters 3, 4, and 5. It can be clearly found that the three proposals are adopted by different network scenarios. The supervised based routing strategy is efficient for the considered static

backbone network, while the online learning based method can effectively address the challenges of the changing traffic pattern. Moreover, for the dynamic networks, the deep learning architectures should learn the routing policy beyond the definite the network topology. Then, the reinforcement learning should be selected to meet this goal. Since the three proposals have different complexity, we consider three different platforms to conduct the related computations. Moreover, the control manner is not only dependent on the network scenario, but also related to the deep learning computations. For example, in Chapter 3, since each trained DBA can be only adopted for one source-destination pair, the distributed control is chosen. On the other hand, the considered VIA in Chapter 5 can predict the path for any source-destination pair. Then, the centralized control manner is adopted to alleviate the computation consumption and reduce the cost of the network. Additionally, as the CNNs in Chapter 4 are utilized to predict the path combination, the centralized control is the only method for this strategy. In conclusion, it can be clearly found that the deep learning technique can be utilized to efficiently tackle the challenges of globally increasing traffic. The various deep learning architectures and different training manners significantly increase the flexibility, leading to great potential to be applied in practical network deployment. To further improve the network performance, more meaningful research can be conducted in the future.

Appendix

Method to Adjust the Weights and Biases of RBMs

As introduced in Chapter 2.3a, the initialization of a DBA is fulfilled by training each RBM. And for each RBM, the values of its weights and biases can be updated according to Equations 2.14 and 2.15. In this part, we will give more details about how to calculate the values of $\frac{\partial l(\Theta,A)}{\partial \theta}$ and $\frac{\partial l(\Theta,A)}{\partial a_i}$. According to Equations 2.13 and 2.17, the following equations can be obtained.

$$\begin{aligned}\frac{\partial l(\Theta,A)}{\partial \theta} &= \frac{\partial \sum_V \log e^{-E(V,H)}}{\partial \theta} - \frac{\partial \sum_V \sum_H \log e^{-E(V,H)}}{\partial \theta} \\ &= \frac{1}{\sum_H e^{-E(V,H)}} \frac{\partial \sum_H e^{-E(V,H)}}{\partial \theta} - \frac{1}{\sum_V \sum_H e^{-E(V,H)}} \frac{\partial \sum_V \sum_H e^{-E(V,H)}}{\partial \theta} \\ &= \frac{\sum_H e^{-E(V,H)}(-\frac{\partial E(V,H)}{\partial \theta})}{\sum_H e^{-E(V,H)}} - \frac{\sum_{V,H} e^{-E(V,H)}(-\frac{\partial E(V,H)}{\partial \theta})}{\sum_{V,H} e^{-E(V,H)}}.\end{aligned} \tag{6.1}$$

As $p(V,H) = \frac{e^{-E(V,H)}}{\sum_{V,H} e^{-E(V,H)}}$ and $p(H|V) = \frac{e^{-E(V,H)}}{\sum_H e^{-E(V,H)}}$, Equation 6.1 can be further written as below:

$$\begin{aligned}\frac{\partial l(\Theta,a)}{\partial \theta} &= \sum_H (p(H|V)(-\frac{\partial E(V,H)}{\partial \theta})) - \sum_{V,H} (p(V,H)(-\frac{\partial E(V,H)}{\partial \theta})) \\ &= E_{p(H|V)}(-\frac{\partial E(V,H)}{\partial \theta}) - E_{p(V,H)}(-\frac{\partial E(V,H)}{\partial \theta}).\end{aligned} \tag{6.2}$$

Then, we can use w_{ij} and b_j to substitute θ, we can get Equations 6.3 and 6.4.

$$\begin{aligned}
\frac{\partial l(\Theta, A)}{\partial w_{ij}} &= \frac{\sum_H e^{-E(V,H)}(-\frac{\partial E(V,H)}{\partial w_{ij}})}{\sum_H e^{-E(V,H)}} - \frac{\sum_{V,H} e^{-E(V,H)}(-\frac{\partial E(V,H)}{\partial w_{ij}})}{\sum_{V,H} e^{-E(V,H)}} \\
&= \sum_H (p(H|V)(-\frac{\partial E(V,H)}{\partial w_{ij}})) - \sum_{V,H} (p(V,H)(-\frac{\partial E(V,H)}{\partial w_{ij}})) \\
&= \sum_H p(H|V) h_j v_i - \sum_{V,H} p(V,H) h_j v_i \\
&= \sum_H p(H|V) h_j v_i - \sum_V \sum_H p(V,H) h_j v_i \\
&= \sum_H p(H|V) h_j v_i - \sum_V \sum_H p(V) p(H|V) h_j v_i \\
&= \sum_H p(H|V) h_j v_i - \sum_V p(V) \sum_H p(H|V) h_j v_i \\
&= \sum_H p(H|V) h_j v_i - \sum_V p(V) v_i \sum_H p(H|V) h_j.
\end{aligned} \tag{6.3}$$

$$\begin{aligned}
\frac{\partial l(\Theta, A)}{\partial b_j} &= \sum_H (p(H|V)(-\frac{\partial E(V,H)}{\partial b_j})) - \sum_{V,H} (p(V,H)(-\frac{\partial E(V,H)}{\partial b_j})) \\
&= \sum_H p(H|V) h_j - \sum_{V,H} p(V,H) h_j \\
&= \sum_H p(H|V) h_j - \sum_V \sum_H p(H|V) p(V) h_j \\
&= \sum_H p(H|V) h_j - \sum_V p(V) \sum_H p(H|V) h_j.
\end{aligned} \tag{6.4}$$

As the units in the hidden layer are binary, we can get the following equation,

$$\begin{aligned}
\sum_H p(H|V) h_j &= \sum_{h_j=0} p(H|V) h_j + \sum_{h_j=1} p(H|V) h_j \\
&= \sum_{h_j=1} p(H|V) \\
&= p(h_j = 1|V).
\end{aligned} \tag{6.5}$$

Therefore, Equations 6.3 and 6.4 can be transferred as following:

$$\frac{\partial l(\Theta, A)}{\partial w_{ij}} = v_i p(h_j = 1|V) - \sum_V p(V) p(h_j = 1|V) v_i, \tag{6.6}$$

$$\frac{\partial l(\Theta, A)}{\partial b_j} = p(h_j = 1|V) - \sum_V p(V) p(h_j = 1|V). \tag{6.7}$$

We can use the same method to calculate $\frac{\partial l(\Theta,A)}{a_i}$ according to the following equation.

$$\begin{aligned}
\frac{\partial l(\Theta,A)}{\partial a_i} &= \sum_H (p(H|V)(-\frac{\partial E(V,H)}{\partial a_i})) - \sum_{V,H}(p(V,H)(-\frac{\partial E(V,H)}{\partial a_i})) \\
&= \sum_H p(H|V)v_i - \sum_{V,H} p(V,H)v_i \\
&= \sum_H p(H|V)v_i - \sum_V \sum_H p(H|V)p(V)v_i \\
&= \sum_H p(H|V)v_i - \sum_V p(V) \sum_H p(H|V)v_i \\
&= v_i - \sum_V p(V)v_i.
\end{aligned} \tag{6.8}$$

As it is impossible to know all the values of V in Equations 6.6, 6.7, and 6.8, it is a practical way to utilize the Markov sampling method to get a set of samples from the training data. We assume the set has l samples, then we utilize the sample set to calculate the values of $\frac{\partial l(\Theta,A)}{w_{ij}}$, $\frac{\partial l(\Theta,A)}{a_i}$, and $\frac{\partial l(\Theta,A)}{b_j}$ as follows:

$$\frac{\partial l(\Theta,A)}{\partial w_{ij}} = v_i p(h_j = 1|V) - \frac{1}{l}\sum_{k=1}^{l} p(h_j = 1|V_k)v_{ki}, \tag{6.9}$$

$$\frac{\partial l(\Theta,A)}{\partial a_i} = v_i - \frac{1}{l}\sum_{k=1}^{l} v_{ki}, \tag{6.10}$$

$$\frac{\partial l(\Theta,A)}{\partial b_j} = v_i p(h_j = 1|V) - \frac{1}{l}\sum_{k=1}^{l} p(h_j = 1|V_k). \tag{6.11}$$

Therefore, we can utilize Equations 6.9, 6.10, and 6.11 to update the values of w_{ij}, a_i, and b_j according to the Equations 2.6, 2.15, and 2.7 in Sec. 2.3a.

Bibliography

[1] J. Liu, H. Guo, Z. M. Fadlullah, and N. Kato, "Energy Consumption Minimization for FiWi Enhanced LTE-A HetNets with UE Connection Constraint," *IEEE Communications Magazine*, vol. 54, no. 11, pp. 56–62, Nov. 2016.

[2] J. Liu, H. Nishiyama, N. Kato, and J. Guo, "On the Outage Probability of Device-to-Device-Communication-Enabled Multichannel Cellular Networks: An RSS-Threshold-Based Perspective," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 1, pp. 163–175, Jan. 2016.

[3] X. Ge, S. Tu, G. Mao, C. Wang, and T. Han, "5G Ultra-Dense Cellular Networks," *IEEE Wireless Communications*, vol. 23, no. 1, pp. 72–79, Feb. 2016.

[4] "5G." [Online]. Available: https://en.wikipedia.org/wiki/5G

[5] "3G." [Online]. Available: https://en.wikipedia.org/wiki/3G

[6] G. H. Sim, S. Klos, A. Asadi, A. Klein, and M. Hollick, "An Online Context-Aware Machine Learning Algorithm for 5G mmWave Vehicular Communications," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2487–2500, Dec. 2018.

[7] J. Ni, X. Lin, and X. S. Shen, "Efficient and Secure Service-Oriented Authentication Supporting Network Slicing for 5G-Enabled IoT," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 644–657, Mar. 2018.

[8] F. Tang, B. Mao, Z. M. Fadlullah, and N. Kato, "On a Novel Deep-Learning-Based Intelligent Partially Overlapping Channel Assignment in SDN-IoT," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 80–86, Sep. 2018.

[9] B. Mao, Z. M. Fadlullah, F. Tang, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "Routing or Computing? The Paradigm Shift Towards Intelligent Computer Network Packet Transmission Based on Deep Learning," *IEEE Transactions on Computers*, vol. 66, no. 11, pp. 1946–1960, Nov. 2017.

[10] "Cisco Visual Networking Index: Forecast and Trends, 20172022." [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html

[11] S. Das, G. Parulkar, and N. McKeown, "Rethinking IP Core Networks," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 5, no. 12, pp. 1431–1442, Dec. 2013.

[12] H. Huang, S. Guo, P. Li, B. Ye, and I. Stojmenovic, "Joint Optimization of Rule Placement and Traffic Engineering for QoS Provisioning in Software Defined Network," *IEEE Transactions on Computers*, vol. 64, no. 12, pp. 3488–3499, Dec. 2015.

[13] C. Chuang, Y. Yu, A. Pang, H. Tseng, and H. Lin, "Efficient Multicast Delivery for Data Redundancy Minimization Over Wireless Data Centers," *IEEE Transactions on Emerging Topics in Computing*, vol. 4, no. 2, pp. 225–241, Apr. 2016.

[14] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: A GPU-accelerated Software Router," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 195–206, Aug. 2010.

[15] "Shortest Path Problem." [Online]. Available: https://en.wikipedia.org/wiki/Shortest_path_problem

[16] C. Mercer and T. Macaulay, "How tech giants are investing in artificial intelligence." [Online]. Available: https://www.techworld.com/picture-gallery/data/tech-giants-investing-in-artificial-intelligence-3629737/

[17] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[18] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, Jul. 2006.

[19] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, vol. 529, pp. 484–489, Jan 2016.

[20] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Driessche, T. Graepel, and D. Hassabis, "Mastering the Game of Go without Human Knowledge," *Nature*, vol. 550, pp. 354–359, Oct. 2017.

[21] M. Bakator and D. Radosav, "Deep Learning and Medical Diagnosis: A Review of Literature," *Multimodal Technologies and Interaction*, vol. 2, no. 3, 2018. [Online]. Available: http://www.mdpi.com/2414-4088/2/3/47

[22] L. Fridman, D. E. Brown, M. Glazer, W. Angell, S. Dodd, B. Jenik, J. Terwilliger, J. Kindelsberger, L. Ding, S. Seaman, H. Abraham, A. Mehler, A. Sipperley, A. Pettinato, B. Seppelt, L. Angell, B. Mehler, and B. Reimer, "MIT Autonomous Vehicle Technology Study: Large-Scale Deep Learning Based Analysis of Driver Behavior and Interaction with Automation," *CoRR*, Aug. 2017. [Online]. Available: http://arxiv.org/abs/1711.06976

[23] H. Nguyen, L. Kieu, T. Wen, and C. Cai, "Deep Learning Methods in Transportation Domain: A Review," *IET Intelligent Transport Systems*, vol. 12, no. 9, pp. 998–1004, Oct. 2018.

[24] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-Art Deep Learning: Evolving Machine Intelligence Toward Tomorrows Intelligent Network Traffic Control Systems," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2432–2455, Fourthquarter 2017.

[25] Y. LeCun and M. Ranzato, "Deep Learning Tutorial," in *Tutorials in International Conference on Machine Learning ICML*, Atlanta, Georgia, USA, 2013.

[26] J. Aweya, "IP Router Architectures: An Overview," *International Journal of Communication Systems*, vol. 14, no. 5, pp. 447–475, 2001.

[27] M. Mukerjee, D. Naylor, and B. Vavala, "Packet Processing on the GPU," Computer Science Department, Carnegie Mellon University, Tech. Rep. [Online]. Available: https://pdfs.semanticscholar.org/28ae/4b5d2aabe691c5b801525e4f2b187771cf19.pdf

[28] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[29] H. Huang, P. Li, S. Guo, and W. Zhuang, "Software-Defined Wireless Mesh Networks: Architecture and Traffic Orchestration," *IEEE Network*, vol. 29, no. 4, pp. 24–30, Jul. 2015.

[30] J. Chen, Y. Ma, H. Kuo, C. Yang, and W. Hung, "Software-Defined Network Virtualization Platform for Enterprise Network Resource Management," *IEEE Transactions on Emerging Topics in Computing*, vol. 4, no. 2, pp. 179–186, Apr. 2016.

[31] D. Kreutz, F. M. V. Ramos, P. E. Verssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[32] K. Wang, Y. Wang, D. Zeng, and S. Guo, "An SDN-based Architecture for Next-Generation Wireless Networks," *IEEE Wireless Communications*, vol. 24, no. 1, pp. 25–31, Feb. 2017.

[33] I. T. Haque and N. Abu-Ghazaleh, "Wireless Software Defined Networking: A Survey and Taxonomy," *IEEE Communications Surveys Tutorials*, vol. 18, no. 4, pp. 2713–2737, Fourthquarter 2016.

[34] B. Mao, F. Tang, Z. M. Fadlullah, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "A Novel Non-Supervised Deep-Learning-Based Network Traffic Control Method for Software Defined Wireless Networks," *IEEE Wireless Communications*, vol. 25, no. 4, pp. 74–81, Aug. 2018.

[35] X. Meng and V. Chaudhary, "A High-Performance Heterogeneous Computing Platform for Biological Sequence Analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 9, pp. 1267–1280, Sep. 2010.

[36] Z. Baker, T. Bhattacharya, M. Dunham, P. Graham, R. Gupta, J. Inman, A. Klein, G. Kunde, A. McPherson, M. Stettler *et al.*, "The PetaFlops Router: Harnessing FPGAs and Accelerators for High Performance Computing," *Links*, vol. 12, no. 4, pp. 16–18, Jan. 2009.

[37] K. Krewell, "Whats the Difference Between a CPU and a GPU?" [Online]. Available: https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/

[38] "Machine Learning." [Online]. Available: https://en.wikipedia.org/wiki/Machine_learning

[39] L. Wang, *Support Vector Machines: Theory and Applications*. Berlin, Heidelberg: Springer-Verlag, 2005.

[40] "A Beginner's Guide to Neural Networks and Deep Learning." [Online]. Available: https://skymind.ai/wiki/neural-network

[41] N. Kato, Z. M. Fadlullah, B. Mao, F. Tang, O. Akashi, T. Inoue, and K. Mizutani, "The Deep Learning Vision for Heterogeneous Network Traffic Control: Proposal, Challenges, and Future Perspective," *IEEE Wireless Communications*, vol. 24, no. 3, pp. 146–153, Jun. 2017.

[42] "Machine Learning Concept: Learning Approaches." [Online]. Available: http://numahub.com/articles/machine-learning-concept-learning-approaches

[43] "Fundamentals of Deep Learning Activation Functions and When to Use Them?" [Online]. Available: https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/

[44] S. K. Kumar, "On Weight Initialization in Deep Neural Networks," *arXiv*, Aug. 2017. [Online]. Available: http://arxiv.org/abs/1704.08863

[45] "5 Regression Loss Functions All Machine Learners Should Know." [Online]. Available: https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0

[46] "How the backpropagation algorithm works." [Online]. Available: http://neuralnetworksanddeeplearning.com/chap2.html

[47] G. E. Hinton, "Training Products of Experts by Minimizing Contrastive Divergence," *Neural Computation*, vol. 14, no. 8, pp. 1771–1800, Aug. 2002.

[48] G. Hinton, "A Practical Guide to Training Restricted Boltzmann Machines," 2010. [Online]. Available: https://www.cs.toronto.edu/~hinton/absps/guideTR.pdf

[49] H. Goh, N. Thome, M. Cord, and J.-H. Lim, "Top-down Regularization of Deep Belief Networks," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'13, USA, 2013, pp. 1878–1886.

[50] Y. LeCun and Y. Bengio, "The Handbook of Brain Theory and Neural Networks." Cambridge, MA, USA: MIT Press, 1998, ch. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258.

[51] A. Deshpande, "A Beginner's Guide To Understanding Convolutional Neural Networks Part 2." [Online]. Available: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

[52] "Convolutional Neural Network." [Online]. Available: http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

[53] "Softmax Regression." [Online]. Available: http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/

[54] C. Zhang, H. Zhang, D. Yuan, and M. Zhang, "Citywide Cellular Traffic Prediction Based on Densely Connected Convolutional Neural Networks," *IEEE Communications Letters*, vol. 22, no. 8, pp. 1656–1659, Aug. 2018.

[55] J. Zhao, H. Qu, J. Zhao, and D. Jiang, "Towards Traffic Matrix Prediction with LSTM Recurrent Neural Networks," *Electronics Letters*, vol. 54, no. 9, pp. 566–568, May 2018.

[56] X. He, K. Wang, H. Huang, T. Miyazaki, Y. Wang, and S. Guo, "Green Resource Allocation based on Deep Reinforcement Learning in Content-Centric IoT," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2018.

[57] "An introduction to Deep Q-Learning: lets play Doom." [Online]. Available: https://medium.freecodecamp.org/an-introduction-to-deep-q-learning-lets-play-doom-54d02d8017d8

[58] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, "Deep Learning for Unsupervised Insider Threat Detection in Structured Cybersecurity Data Streams," *arXiv*, vol. abs/1710.00811, 2017. [Online]. Available: http://arxiv.org/abs/1710.00811

[59] I. Rubin, "Access-control Disciplines for Multi-access Communication Channels: Reservation and TDMA schemes," *IEEE Transactions on Information Theory*, vol. 25, no. 5, pp. 516–536, Sep. 1979.

[60] E. Soltanmohammadi, K. Ghavami, and M. Naraghi-Pour, "A Survey of Traffic Issues in Machine-to-Machine Communications over LTE," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 865–884, Dec. 2016.

[61] J. Huang, H. Wang, Y. Qian, and C. Wang, "Priority-Based Traffic Scheduling and Utility Optimization for Cognitive Radio Communication Infrastructure-Based Smart Grid," *IEEE Transactions on Smart Grid*, vol. 4, no. 1, pp. 78–86, Mar. 2013.

[62] "4G." [Online]. Available: https://en.wikipedia.org/wiki/4G

[63] A. Tanenbaum, *Computer Networks*, 4th ed. Prentice Hall Professional Technical Reference, 2002.

[64] F. Ren, T. He, S. K. Das, and C. Lin, "Traffic-Aware Dynamic Routing to Alleviate Congestion in Wireless Sensor Networks," *IEEE Transactions on Parallel & Distributed Systems*, vol. 22, no. 9, pp. 1585–1599, Jan. 2011.

[65] Y. Jia, I. Nikolaidis, and P. Gburzynski, "Multiple Path Routing in Networks with Inaccurate Link State Information," in *IEEE International Conference on Communications*, Finland, Jun. 2001, pp. 2583–2587.

[66] T. Liebig, N. Piatkowski, C. Bockermann, and K. Morik, "Dynamic Route Planning with Real-time Traffic Predictions," *Inf. Syst.*, vol. 64, no. C, pp. 258–265, Mar. 2017.

[67] X. H. Chen, "Adaptive Traffic-load Shedding and Its Capacity Gain in CDMA Cellular Systems," *IEE Proceedings - Communications*, vol. 142, no. 3, pp. 186–192, Jun. 1995.

[68] D. K. Y. Yau, J. C. S. Lui, F. Liang, and Y. Yam, "Defending Against Distributed Denial-of-Service Attacks with Max-min Fair Server-centric Router Throttles," *IEEE/ACM Transactions on Networking*, vol. 13, no. 1, pp. 29–42, Feb 2005.

[69] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: Motivation, Architecture, Algorithms, Performance," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1246–1259, Dec. 2006.

[70] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant." [Online]. Available: https://www.cs.princeton.edu/courses/archive/fall16/cos561/papers/Cubic08.pdf

[71] S. R. Pokhrel and C. Williamson, "Modeling Compound TCP Over WiFi for IoT," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 864–878, Apr. 2018.

[72] "Open Shortest Path First." [Online]. Available: https://en.wikipedia.org/wiki/Open_Shortest_Path_First

[73] H. Li, K. Ota, and M. Dong, "Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, Jan. 2018.

[74] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[75] "CAIDA." [Online]. Available: www.caida.org/home/

[76] A. Fog, *Instruction Tables*. Technological University of Denmark, 2018.

[77] "Comparison of Deep Learning Software." [Online]. Available: https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software

[78] J. Liu, S. Zhang, N. Kato, H. Ujikawa, and K. Suzuki, "Device-to-Device Communications for Enhancing Quality of Experience in Software Defined Multi-tier LTE-A Networks," *IEEE Network*, vol. 29, no. 4, pp. 46–52, Jul. 2015.

[79] H. Guo, J. Liu, Z. M. Fadlullah, and N. Kato, "On Minimizing Energy Consumption in FiWi Enhanced LTE-A HetNets," *IEEE Transactions on Emerging Topics in Computing*, vol. 6, no. 4, pp. 579–591, Oct. 2018.

[80] L. Qiang, J. Li, and C. Touati, "A User Centered Multi-Objective Handoff Scheme for Hybrid 5G Environments," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 3, pp. 380–390, Jul. 2017.

[81] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "Research Challenges for Traffic Engineering in Software Defined Networks," *IEEE Network*, vol. 30, no. 3, pp. 52–58, May 2016.

[82] H. Hu, H. Chen, P. Mueller, R. Q. Hu, and Y. Rui, "Software Defined Wireless Networks (SDWN): Part 1 [Guest Editorial]," *IEEE Communications Magazine*, vol. 53, no. 11, pp. 108–109, Nov. 2015.

[83] Y. Xiao, K. Thulasiraman, G. Xue, and M. Yadav, "QoS Routing Under Multiple Additive Constraints: A Generalization of the LARAC Algorithm," *IEEE Transactions on Emerging Topics in Computing*, vol. 4, no. 2, pp. 242–251, Apr. 2016.

[84] N. Egashira, K. Yano, S. Tsukamoto, J. Webber, and T. Kumagai, "Low Latency Relay Processing Scheme for WLAN Systems Employing Multiband Simultaneous Transmission," in *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, USA, Mar. 2017, pp. 1–6.

[85] Z. M. Fadlullah, Y. Kawamoto, H. Nishiyama, N. Kato, N. Egashira, K. Yano, and T. Kumagai, "Multi-Hop Wireless Transmission in Multi-Band WLAN Systems: Proposal and Future Perspective," *IEEE Wireless Communications*, pp. 1–6, 2018.

[86] "Understanding Xavier Initialization in Deep Neural Networks." [Online]. Available: https://prateekvjoshi.com/2016/03/29/understanding-xavier-initialization-in-deep-neural-networks/

[87] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning Combinatorial Optimization Algorithms over Graphs," *CoRR*, vol. abs/1704.01665, 2017. [Online]. Available: http://arxiv.org/abs/1704.01665

[88] L. Nie, D. Jiang, L. Guo, S. Yu, and H. Song, "Traffic Matrix Prediction and Estimation Based on Deep Learning for Data Center Networks," in *2016 IEEE Globecom Workshops (GC Wkshps)*, Dec. 2016, pp. 1–6.

[89] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1998.

[90] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-Driven Networking: A Deep Reinforcement Learning based Approach," *CoRR*, vol. abs/1801.05757, 2018.

[91] M. Hu, J. Luo, Y. Wang, and B. Veeravalli, "Adaptive Scheduling of Task Graphs with Dynamic Resilience," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 17–23, Jan. 2017.

[92] "Internet Topology Zoo." [Online]. Available: http://www.topology-zoo.org/dataset.html

[93] "NP-hardness." [Online]. Available: https://en.wikipedia.org/wiki/NP-hardness

[94] J. Liu, Y. Shi, L. Zhao, Y. Cao, W. Sun, and N. Kato, "Joint Placement of Controllers and Gateways in SDN-Enabled 5G-Satellite Integrated Network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 2, pp. 221–232, Feb. 2018.

[95] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[96] A. Tamar, Y. WU, G. Thomas, S. Levine, and P. Abbeel, "Value Iteration Networks," in *Advances in Neural Information Processing Systems 29*, Barcelona, Spain, 2016, pp. 2154–2162.

[97] M. Alzantot, "Deep Reinforcement Learning Demysitifed (Episode 2)-Policy Iteration, Value Iteration and Q-learning."

[98] "Value Iteration." [Online]. Available: http://artint.info/html/ArtInt_227.html

[99] A. Ito, "Application of Episodic Q-Learning to a Multi-agent Cooperative Task," in *PRICAI 2002: Trends in Artificial Intelligence*, Berlin, Heidelberg, 2002, pp. 188–197.

[100] M. L. Littman, T. L. Dean, and L. P. Kaelbling, "On the Complexity of Solving Markov Decision Problems," in *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, ser. UAI'95, Montral, Qu, Canada, 1995, pp. 394–402.

[101] C. Papadimitriou and J. N. Tsitsiklis, "The Complexity of Markov Decision Processes," *Math. Oper. Res.*, vol. 12, no. 3, pp. 441–450, Aug. 1987.

[102] "Star Network." [Online]. Available: https://en.wikipedia.org/wiki/Star_network

[103] "Tensorflow." [Online]. Available: https://www.tensorflow.org/

Publications

Journals

[1] Zubair Md. Fadlullah, **Bomin Mao**, Fengxiao Tang, and Nei Kato, "Value Iteration Architecture based Deep Learning for Intelligent Routing Exploiting Heterogeneous Computing Platforms," IEEE Transactions on Computers, Accepted, doi: 10.1109/TC.2018.287448.

[2] Nei Kato, Zubair Md. Fadlullah, Fengxiao Tang, **Bomin Mao**, Shigenori Tani, Atsushi Okamura, and Jiajia Liu, "Optimizing Space-Air-Ground Integrated Networks by Artificial Intelligence," IEEE Wireless Communications Magazine (WCM), Accepted, DOI: 10.1109/MWC.2018.1800365.

[3] Fengxiao Tang, Zubair Md. Fadlullah, **Bomin Mao**, Nei Kato, Fumie Ono, and Ryu Miura, "On A Novel Adaptive UAV-Mounted Cloudlet-Aided Recommendation System for LBSNs," IEEE Transactions on Emerging Topics in Computing, In press, DOI: 10.1109/TETC.2018.2792051.

[4] Fengxiao Tang, Zubair Md. Fadlullah, **Bomin Mao**, and Nei Kato, "An Intelligent Traffic Load Prediction Based Adaptive Channel Assignment Algorithm in SDN-IoT: A Deep Learning Approach," IEEE Internet of Things Journal, vol. 5, no. 6, pp. 5141 - 5154, Dec. 2018.

[5] Zubair Md. Fadlullah, Fengxiao Tang, **Bomin Mao**, Jiajia Liu, Nei kato, "On Intelligent Trafic Control For Large Scale Heterogeneous Networks: A Value Matrix Based Deep Learning Approach," IEEE Communication Letter, vol. 22, no. 12, pp. 2479-2482, Dec. 2018.

[6] Yibo Zhou, Zubair Md. Fadlullah, **Bomin Mao** and Nei Kato, "A Deep Learning Based Radio Resource Assignment Technique for 5G Ultra Dense Networks," IEEE Network Magazine, vol. 32, no. 6, pp. 28 - 34, Nov. 2018.

[7] Fengxiao Tang, **Bomin Mao**, Zubair Md. Fadlullah, and Nei Kato, "On a Novel Deep Learning Based Intelligent Partially Overlapping Channel Assignment in SDN-IoT," IEEE Communications Magazine, vol. 56, no. 9, pp. 80-86, Sep. 2018.

[8] **Bomin Mao**, Fengxiao Tang, Zubair Md. Fadlullah, Nei Kato, Osamu Akashi, Takeru Inoue, and Kimihiro Mizutani, "A Novel Non-supervised Deep Learning Based Network Traffic Control Method for Software Defined Wireless Networks," IEEE Wireless Communications Magazine, vol. 25, no. 4, pp. 74-81, Aug. 2018.

[9] Fengxiao Tang, **Bomin Mao**, Zubair Md. Fadlullah, Nei Kato, Osamu Akashi, Takeru Inoue, and Kimihiro Mizutani, "On Removing Routing Protocol from Future Wireless Networks: A Real-time Deep Learning Approach for Intelligent Traffic Control, " IEEE Wirelesss Magazine (WCM), vol. 25, no. 1, pp. 154-160, Feb. 2018.

[10] **Bomin Mao**, Zubair Md. Fadlullah, Fengxiao Tang, Nei Kato, Osamu Akashi, Takeru Inoue, and Kimihiro Mizutani, "Routing or Computing? The Paradigm Shift Towards Intelligent Computer Network Packet Transmission Based on Deep Learning," IEEE Transactions on Computers, vol. 66, no. 11, pp. 1946-1960, Nov. 2017.

[11] Zubair Md. Fadlullah, Fengxiao Tang, **Bomin Mao**, Nei Kato, Osamu Akashi, Takeru Inoue, and Kimihiro Mizutani, "State-of-the-Art Deep Learning: Evolving Machine Intelligence Toward Tomorrows Intelligent Network Traffic Control Systems," IEEE Communications Surveys and Tutorials, vol. 19, no. 4, pp. 2432-2455, May 2017.

[12] Nei Kato, Zubair Md. Fadlullah, **Bomin Mao**, Fengxiao Tang, Osamu Akashi, Takeru Inoue, and Kimihiro Mizutani, "The Deep Learning Vision for Heterogeneous Network Traffic Control Proposal, Challenges, and Future Perspective ," IEEE Wireless Communications, vol. 24, no. 3, pp. 146-153, Dec. 2016.

Refereed Conference Papers

[13] **Bomin Mao**, Zubair Md. Fadlullah, Fengxiao Tang, Nei Kato, Osamu Akashi, Takeru Inoue, and Kimihiro Mizutani, "A Tensor Based Deep Learning Technique for Intelligent Packet Routing," IEEE Global Communications Conference (GLOBECOM 2017), Singapore, Dec. 2017. Best Paper Award.

[14] Fengxiao Tang, **Bomin Mao**, Zubair Md. Fadlullah, and Nei Kato, "Deep Spatiotemporal Partially Overlapping Channel Allocation: Joint CNN and Activity Vector Approach," IEEE Global Communications Conference (GLOBECOM 2018), Abu Dhabi, UAE, Dec. 2018.

[15] Fengxiao Tang, **Bomin Mao**, Zubair Md. Fadlullah, and Nei Kato, "On Extracting the Spatial-Temporal Features of Network Traffic Patterns: A Tensor Based Deep Learning Model," to appear, IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC18), Guiyang, China, Aug. 2018.

Title: Routing or Computing? The Paradigm Shift Towards Intelligent Computer Network Packet Transmission Based on Deep Learning
Author: Bomin Mao
Publication: Computers, IEEE Transactions on
Publisher: IEEE
Date: 1 Nov. 2017

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Title: Value Iteration Architecture based Deep Learning for Intelligent Routing Exploiting Heterogeneous Computing Platforms

Author: Zubair Md. Fadlullah

Publication: Computers, IEEE Transactions on

Publisher: IEEE

Date: Dec 31, 1969

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Title: Optimizing Space-Air-Ground Integrated Networks by Artificial Intelligence
Author: Nei Kato
Publication: IEEE Wireless Communications Magazine
Publisher: IEEE
Date: Dec 31, 1969

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Title:	A Novel Non-Supervised Deep-Learning-Based Network Traffic Control Method for Software Defined Wireless Networks
Author:	Bomin Mao
Publication:	IEEE Wireless Communications Magazine
Publisher:	IEEE
Date:	AUGUST 2018

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK CLOSE WINDOW

Title: A Tensor Based Deep Learning Technique for Intelligent Packet Routing

Conference Proceedings: GLOBECOM 2017 - 2017 IEEE Global Communications Conference

Author: Bomin Mao

Publisher: IEEE

Date: Dec. 2017

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK CLOSE WINDOW