Routing or Computing? The Paradigm Shift Towards Intelligent Computer Network Packet Transmission Based on Deep Learning

Bomin Mao, *Student Member, IEEE*, Zubair Md. Fadlullah, *Senior Member, IEEE*, Fengxiao Tang, *Student Member, IEEE*, Nei Kato, *Fellow, IEEE*, Osamu Akashi, Takeru Inoue, and Kimihiro Mizutani

Abstract—Recent years, 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. Multi-core platforms significantly promote SDRs' parallel computing capacities, enabling them to adopt artificial intelligent techniques, i.e., deep learning, to manage routing paths. In this paper, 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. Even though deep learning techniques have been extensively exploited in various computing areas, researchers have, to date, not been able to effectively utilize deep learning based route computation for high-speed core networks. We envision a supervised deep learning system to construct the routing tables and show how the proposed method can be integrated with programmable routers using both Central Processing Units (CPUs) and Graphics Processing Units (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.

Index Terms—Software defined routers, network traffic control, deep learning, backbone networks, core networks, routing

1 Introduction

The routers in the Internet core have witnessed several generations of changes in hardware while the main idea behind the routing algorithms has traditionally been remarkably similar. This is because the manner, in which the Internet core and the wired/wireless heterogeneous backbone networks are constructed, has largely remained unchanged over the years [1]. On the other hand, 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. 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 confront the challenges of traffic management for ensuring the Quality of Service (QoS) as well as dealing with the declining profits [2]. Traditionally, operators rely on the hardware solution in order to improve the core network performance, i.e., by simply increasing the number and size of routers and/or links, which results in a massive investment splurge. On the other hand, 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 [3]. In other words, the advancement in the software driven routing strategies appears to always fall behind the prevalent routing policies due to the continuously varying network environments. 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. However, due to their proprietary hardware architectures, designing programmable routers is a challenging research area. Hence, researchers have considered the use of Software Defined Routers (SDRs) which deploy programmable routing strategies on the commodity hardware architecture to carry out packet processing and transmission. As a critical component of the Software Defined Networks (SDNs), SDRs are required not only to support the software-based packet transmission but also to flexibly execute other functions according to the network operators' requirements so as to optimize their networks. However, recent research works involving SDRs mainly focus on enhancing their hardware

- *B. Mao, Z. M. Fadlullah, F. Tang, and N. Kato are with the Graduate School of Information Sciences, Tohoku University, Sendai 980-8577, Japan. E-mail: {bomin.mao, zubair, fengxiao.tang, kato}@it.is.tohoku.ac.jp.*
- *O. Akashi, T. Inoue, and K. Mizutani are with Nippon Telegraph and Telephone Corporation (NTT) Network Innovation Laboratories, Yokosuka, Kanagawa 239-0847, Japan.*
  E-mail: {akashi.osamu, inoue.takeru, mizutani.kimihiro}@lab.ntt.co.jp.

Manuscript received 27 Jan. 2017; revised 9 May 2017; accepted 23 May 2017. Date of publication 29 May 2017; date of current version 16 Oct. 2017.
(Corresponding author: Nei Kato.)
Recommended for acceptance by J. D. Bruguera.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2017.2709742

aspects in order to improve the processing throughput performance comparable to that of the hardware-based routers. For instance, researchers as well as networking manufacturers have explored multi-core-based SDRs [4]. While the multi-core Central Processing Units (CPUs) dominate the relevant research domain [5], the Graphics Processing Unit (GPU) -accelerated SDR is emerging as an exciting research area. The reason behind this is the inherent capability of the GPU to concurrently run tens of thousands of threads to efficiently process packets [3], [6], [7]. In other words, the GPU can execute the same program to process different sets of data in a parallel fashion that can also incorporate with the multi-core CPU to undertake different instructions at the same time [8]. The cooperation of GPUs and CPUs can significantly promote the packet processing throughput of SDRs [7] that can be regarded as a competent candidate for modern backbone networks.

In order to further complement the research on the hardware architecture of the backbone routers, the network traffic control strategy also needs to be improved to meet the increasing traffic demand in recent decades [9]. This is because a good routing path management technique can directly alleviate (or even overcome) network congestion. In this paper, we focus on the design of packet routing strategies. Conventional rule-based routing strategies (e.g., the Shortest Path (SP) algorithm and so forth) which choose the paths according to the maximum or minimum metric values are usually attributed to the shortcomings of a significantly slow convergence. Furthermore, they may not be particularly suited to the multi-metric networks because the relationships among multiple metrics are difficult to be manually estimated [10]. In order to exploit the complex relationships among the various metrics to decide the best path, machine learning-based intelligent network traffic control systems have drawn much attention in a wide spectrum of network environments [11], [12]. Supervised, unsupervised, and reinforcement learning techniques have been exploited to manage the packet routing in many different network scenarios [13], [14], [15], such as Wireless Mesh Networks (WMNs) [14]. However, these intelligent strategies are still based on the traditional rule-based routing due to the inefficiency of conventional machine learning techniques in dealing with multiple network parameters as well as the difficulties of characterizations of the inputs and outputs [16], [17]. Since 2006, it has been witnessed that a step-by-step breakthrough has been achieved in the deep learning architectures (e.g., multi-layer neural networks) to enable them to be adopted in extremely complex activities where only humans have ever played the dominant role, such as Google's AlphaGo in complex board games [18]. Moreover, GPUs have emerged as the most viable hardware platform for running deep learning algorithms due to their highly parallel computing capability complemented by a plethora of programming toolkits offered by various GPU manufacturers [8]. Therefore, it is practical to exploit deep learning algorithms for smart network traffic control (e.g., intelligent route management) by harnessing the GPU-accelerated SDRs.

The contributions of our work in this paper are as follows. First, we explore the routing strategies from three aspects, namely network traffic control, deep learning, and CPU/GPU computing architectures as shown in Fig. 1. Second, we propose a deep learning based routing table construction method for a GPU-accelerated SDR. In our proposal, we adopt supervised Deep Belief Architectures (DBAs) [19] to compute the subsequent nodes (i.e., routers) with the traffic patterns of the edge routers as the input. Third, we present unique characterizations of inputs and outputs to our adopted DBAs based on the incoming traffic patterns at the edge routers. The proposed DBAs are trained according to the collected data comprising inbound traffic patterns and corresponding subsequent nodes (i.e., routers). Also, we demonstrate how the trained DBAs can predict the next nodes. Fourth, we demonstrate the benefits of the deep learning based routing strategy in terms of lower signaling overhead as well as fast convergence, which result in significantly better traffic control. Fifth, we demonstrate the effectiveness of our proposed deep learning based solution compared to a benchmark routing method through both analysis and extensive simulation results. In particular, we demonstrate how our proposal can work on a GPU-accelerated SDR and evaluate the time cost to run the routing strategy after analyzing its complexity. The results show that the proposed routing strategy runs more than 100 times faster on a GPU than on a CPU.

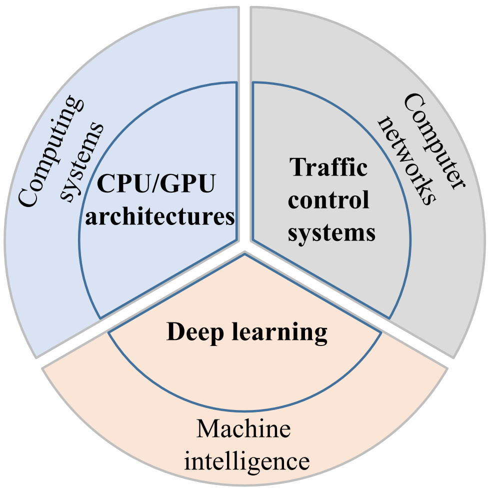


Fig. 1. Recent inter-disciplinary trends indicate an inter-disciplinary area involving computing systems, computer networks, and machine intelligence. Particularly, network traffic control systems are becoming robust and intelligent owing to the advancement in CPU/GPU technologies and deep learning, respectively.

The remainder of the paper is structured as follows. Section 2 includes relevant research works. And in Section 3, 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 Sections 4 and 5 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 in Section 6. Finally, Section 7 concludes the article.

2 Related Research Work

In this section, we introduce the relevant research work from hardware and software perspectives that take into account the state-of-the-art in SDRs and deep learning, respectively. In addition, while discussing the deep learning related research, we also describe the relevant machine intelligent routing strategies which exist in the literature.

2.1 Related Research Work on SDR

Because deep learning execution needs the support of high performance computational hardware, it is necessary to discuss the state-of-the-art SDR technology. Even though the approach of implementing the packet processing logic into hardware continues to improve the routers' line rates up to 100 Gbps, applying new network traffic control algorithms remains a challenging issue because of the shortcoming of the purpose-designed hardware in terms of feasibility and extensibility [6], [20]. On the other hand, the general hardware based SDR is not quite efficient in the packet processing although it is programmable and flexible. As the multi-core solution can significantly improve the computation power, researchers from academia and industries have shown great interests in utilizing the multiple cores and/or threads provided by a CPU or a GPU to parallelly operate the routing task to improve the processing throughput of current SDRs [6], [21]. The RouteBricks architecture presented in [21] explores a novel network architecture in which packets are processed in software running on a cluster of general-purpose PC hardware. The IPv4 forwarding throughput of RouteBricks has been shown to be as high as 8.7 Gbps with 64 B packets. However, its performance may not exceed 10 Gbps as the CPU becomes the bottleneck for more compute-intensive applications. As a remedy to this problem, the PacketShader architecture [3] shifts the computation needs for parallel packets processing from CPUs to GPUs as a GPU consists of many more cores compared to a CPU. The evaluation of this architecture with a single commodity PC demonstrated that the PacketShader's ability to forward 64 B IPv4 packets at 39 Gbps. Also, it was indicated in [3] that the line rate may be further improved to 100 Gbps by scaling the performance of I/O hubs [7]. Furthermore, network manufacturers including Intel and Cisco are adopting similar projects to exploit high performance computation equipment in SDR architectures and develop commercial products [4], [5]. It may be noticed from the aforementioned works that the current programmable SDRs can offer competitive line rates (i.e., a much lower expense compared to traditional purpose-designed routers) while keeping the strength of flexibility and extensibility.

2.2 Related Research Work on Deep Learning and Machine Intelligent Routing Strategies

As machine learning can be useful for predicting the network parameters, several researchers have attempted to exploit the Artificial Neural Networks (ANNs), a machine learning technique, to forecast the network traffic [15], the link bandwidth, or other metrics [22] over a few time-intervals. However, the efficiency of those strategies is restricted by the shortcomings of conventional machine learning techniques mainly because of the lack of appropriate learning algorithms for deep structures, availability of sufficiently large training data, and so forth. In fact, according to the work in [11], even though adding the layers can extract a much higher level of features in extremely complex applications, the conventional deep neural network architecture comprising many hidden layers exhibits poor performance comparable to that of the shallow ones. Recently, deep learning emerged as a promising computational model to effectively utilize multiple processing layers to extract features from data with multiple levels of abstraction [11]. This method uses the generic non-linear modules to transform the representations at one level into the representations at a much higher and more abstract level, and then compose these representations, by which it can automatically and accurately learn features [23]. The introduction of the Greedy Layer-Wise training in 2006 that utilizes unsupervised learning procedures to pre-train the deep neural network architectures revolutionized the deep learning technique [19], [24]. Through the Greedy Layer-Wise pre-training, followed by backpropagation algorithm based fine-tuning of the entire deep architecture, the deep learning achieved record-breaking results in many complex applications such as speech recognition [25] and so forth. A good example of applying deep learning into speech recognition is Apple's smart assistant called "Siri" [26]. In other fields which have been traditionally dominated by humans, researchers have explored the application of deep learning and achieved inspiring results, such as Google's AlphaGo in remarkably complex board games [18]. It is evident from the literature that the deep learning technique has become the state-of-the-art in applications which typically require a large volume of computation. On one hand, the deep learning applications have been limited to mainly image/character/pattern recognition and natural language processing fields [11]. On the other hand, deep learning for network traffic control systems has not yet been successfully attempted that also continues to experience a growing computation burden. This is because the number of devices connected to the Internet is increasingly heavy and the global network traffic has exploded in the recent decades [9]. As a consequence, network operators need to consider more parameters and more complex rules when designing routing strategies to meet the stringent and changing network requirements. According to [16], deep learning has a bright potential to be applied in network traffic control systems to estimate the best paths by taking into account various requirements. Moreover, since applying deep learning for routing is still a very new topic, no previous research work attempted to integrate deep learning into SDRs. On the other hand, our motivation comes from the fact that the SDRs can go hand-in-hand with deep learning due to the fact that many SDRs use multi-core platforms (e.g., GPUs). This paves the way for adopting deep learning based intelligent routing strategies in SDRs. From this point, we attempt to consider both the hardware and software perspectives by adopting GPU-powered SDRs to execute the deep learning technique to estimate the next nodes for better routing management.

3 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.

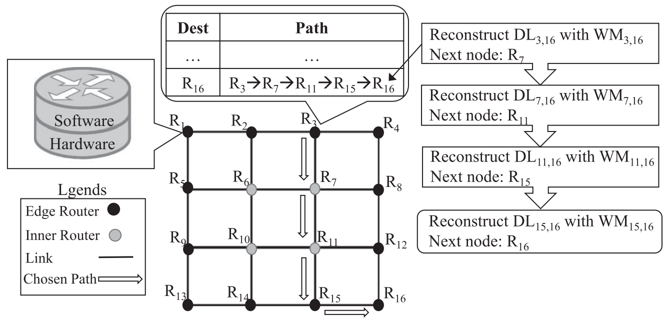
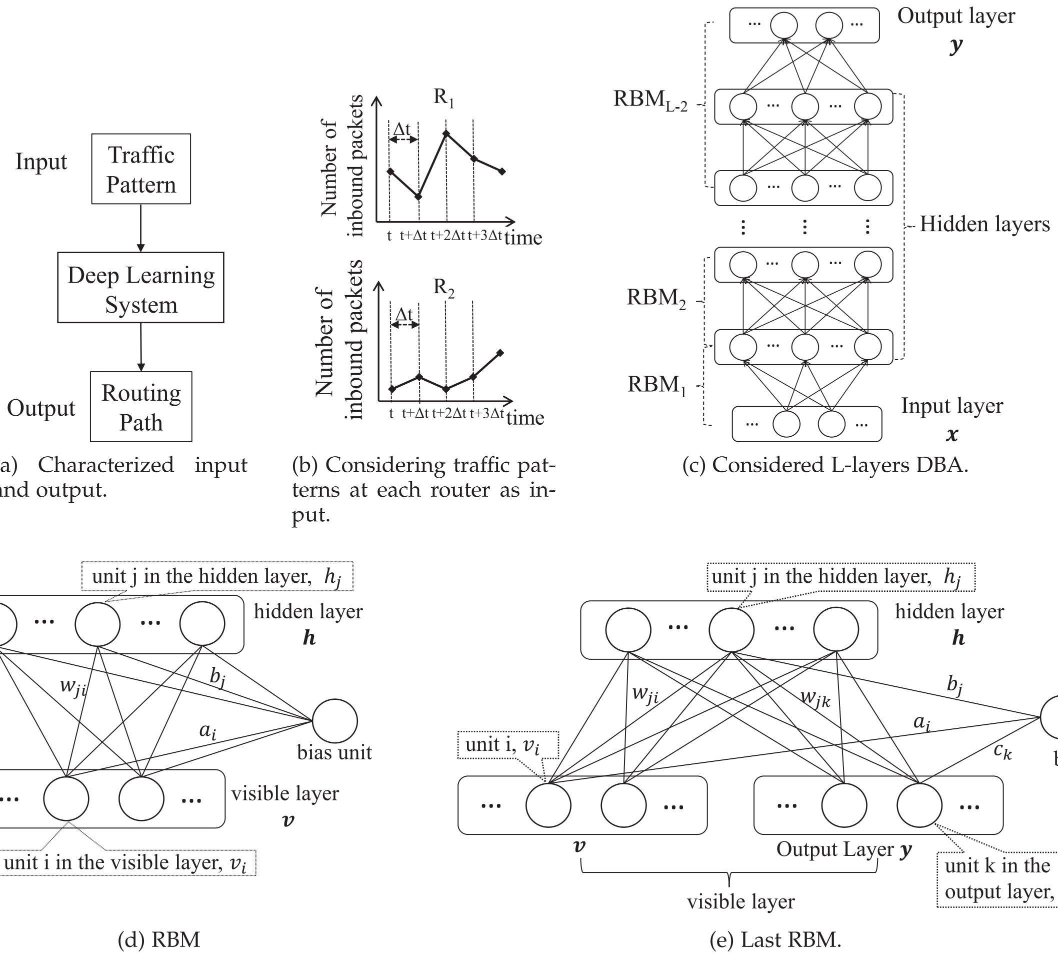


Fig. 2. Considered system model and problem statement.

3.1 Input and Output Design

Our considered core network system model is depicted in Fig. 2 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, 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. 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 Section 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. 3a 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



(a) Characterized input and output.

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

(c) Considered L-layers DBA.

(d) RBM

(e) Last RBM.

Fig. 3. Considered model of the proposed deep learning system.

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. 3b. 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 an N dimensional vector, whose ith element is the ith 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 ith element in the N dimensional vector is 1, then the ith router in the considered network is chosen as the next node. In summary, we can use two N dimensional vectors, $\boldsymbol{x}$ and $\boldsymbol{y}$, to represent the input and output of the deep learning structure and an example of $\boldsymbol{x}$ and $\boldsymbol{y}$ is given as follows:

$$\boldsymbol{x} = (tp_1, tp_2, \ldots, tp_{N-1}, tp_N), \tag{1}$$

$$\boldsymbol{y} = (0, 1, \ldots, 0, 0), \tag{2}$$

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 $\boldsymbol{y}$, 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 $\boldsymbol{y}$, the deep learning structure is a logistic regression model, which we need to design next.

3.2 Deep Learning Structure Design

In order to design the deep learning structure, labeled data to perform supervised training (i.e., many sets of $(\boldsymbol{x}, \boldsymbol{y})$) are required. For fulfilling the task to compute the next router with traffic patterns, we choose the DBA shown in Fig. 3c as our deep learning structure since it is most common and effective among all the deep learning models [27]. As shown in the figure, we assume the DBA consists of L layers, the input layer, $\boldsymbol{x}$, the output layer, $\boldsymbol{y}$, 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. The structure of each RBM is shown in Fig. 3d. It can be seen that each RBM consists of two layers, the visible layer, $\boldsymbol{v}$, and the hidden layer, $\boldsymbol{h}$. 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_{ji} denotes the weight of the link connecting the unit j in the hidden layer and the unit i in the visible 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 mentioned in Section 2.2, the deep learning training process consists of two steps: the Greedy Layer-Wise training to initialize the structure and the backpropagation 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. 3d [28]. 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

$$l(\boldsymbol{\theta}, \boldsymbol{a}) = \sum_{t=1}^{m} \log p(\boldsymbol{v}^{(t)}), \tag{3}$$

where $\boldsymbol{\theta}$ denotes the vector consisting of all the values of the weights and biases of the hidden layer. $\boldsymbol{\theta}$ can be written as $\boldsymbol{\theta} = (\boldsymbol{w}, \boldsymbol{b})$. $\boldsymbol{w}$ and $\boldsymbol{b}$ represent the vectors consisting of all the weights, w_{ji}, and biases of the hidden units, b_j, respectively. $\boldsymbol{a}$ consists of the biases of the visible units, a_i. m denotes the number of training data. $\boldsymbol{v}^{(t)}$ is the tth training data, the probability of which is $p(\boldsymbol{v}^{(t)})$.

To maximize $l(\boldsymbol{\theta}, \boldsymbol{a})$, we can use the gradient descent of $l(\boldsymbol{\theta}, \boldsymbol{a})$ to adjust $\boldsymbol{w}$, $\boldsymbol{a}$, and $\boldsymbol{b}$, which can be described as in Equations (4) and (5)

$$\theta := \theta + \eta \frac{\partial l(\boldsymbol{\theta}, \boldsymbol{a})}{\partial \theta}, \tag{4}$$

$$a_i := a_i + \eta \frac{\partial l(\boldsymbol{\theta}, \boldsymbol{a})}{\partial a_i}, \tag{5}$$

where η is the learning rate in deep learning. Here, θ represents any w_{ji} and any b_j.

To calculate the value of $p(\boldsymbol{v})$ (representing any $p(\boldsymbol{v}^{(t)})$), we need to model the RBM as an energy model since the RBM is a particular form of log-linear Markov Random Field (MRF) [29]. The energy function, $E(\boldsymbol{v}, \boldsymbol{h})$, and the joint probability function, $p(\boldsymbol{v}, \boldsymbol{h})$, are defined as follows:

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_i \sum_j h_j w_{ji} v_i \tag{6}$$

$$p(\boldsymbol{v},\boldsymbol{h}) = \frac{e^{-E(\boldsymbol{v},\boldsymbol{h})}}{Z} \tag{7}$$

$$Z = \sum_{\boldsymbol{v}}\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}, \tag{8}$$

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. 3d, respectively. Z represents the normalizing constant partition function. Also, the relationship between $p(v)$ and $p(\boldsymbol{v},\boldsymbol{h})$ can be expressed as follows:

$$p(\boldsymbol{v}) = \sum_{\boldsymbol{h}} p(\boldsymbol{v},\boldsymbol{h}). \tag{9}$$

We can use Equations (3), (4), (5), (6), (7), (8), and (9) to obtain the values of θ [28]. However, the complexity of the calculation of $\sum_{\boldsymbol{v}}\sum_{\boldsymbol{h}}$ in Equation (8) is $2^{n_{\boldsymbol{v}}+n_{\boldsymbol{h}}}$, which is extremely high ($n_{\boldsymbol{v}}$ and $n_{\boldsymbol{h}}$ represent the dimensions of vectors $\boldsymbol{v}$ and $\boldsymbol{h}$, respectively). Another problem is that to calculate Equation (8), it is necessary but impossible to consider all the possible values of $\boldsymbol{v}$ and $\boldsymbol{h}$ instead of only the obtained training data. To solve these problems, Hinton et al. proposed the Contrastive Divergence (CD) method [30]. The main idea of CD is to use the Gibbs Sampling method to sample the values of $\boldsymbol{v}$ and $\boldsymbol{h}$ to approximate the real values since the conditional distribution probability of one layer (while the value of the other layer is given), e.g., $p(\boldsymbol{v}|\boldsymbol{h};\boldsymbol{\theta},\boldsymbol{a})$, can be calculated. The detailed procedures of CD are omitted in the paper, and can be found in [30]. 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,

$$p(\boldsymbol{v}|\boldsymbol{h};\boldsymbol{\theta},\boldsymbol{a}) = \prod_i p(v_i|\boldsymbol{h};\boldsymbol{\theta},\boldsymbol{a}), \tag{10}$$

$$p(\boldsymbol{h}|v;\boldsymbol{\theta},\boldsymbol{a}) = \prod_j p(h_j|v;\boldsymbol{\theta},\boldsymbol{a}), \tag{11}$$

where $p(\boldsymbol{v}|\boldsymbol{h};\boldsymbol{\theta},\boldsymbol{a})$ and $p(\boldsymbol{h}|v;\boldsymbol{\theta},\boldsymbol{a})$ are the conditional probability of $\boldsymbol{v}$ given $\boldsymbol{h}$ and the conditional probability of $\boldsymbol{h}$ given $\boldsymbol{v}$, respectively. $p(v_i|\boldsymbol{h};\boldsymbol{\theta},\boldsymbol{a})$ is the conditional probability distribution of unit i in the visible layer when the hidden layer is fixed. Also, $p(h_j|v;\boldsymbol{\theta},\boldsymbol{a})$ 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 $p(v_i = 1|\boldsymbol{h};\boldsymbol{\theta},\boldsymbol{a})$ and $p(h_j = 1|v;\boldsymbol{\theta},\boldsymbol{a})$ are given as follows:

$$p(v_i = 1|\boldsymbol{h};\boldsymbol{\theta},\boldsymbol{a}) = sigm\left(\sum_j w_{ji}h_j + a_i\right) \tag{12}$$

$$p(h_j = 1|\boldsymbol{v};\boldsymbol{\theta},\boldsymbol{a}) = sigm\left(\sum_i w_{ji}v_i + b_j\right), \tag{13}$$

where $sigm$ represents the sigmoid activation function and $sigm(x) = \frac{1}{1+e^{-x}}$.

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 [31]. Therefore, for RBM$_1$ in our proposed DBA, Equations (6) and (12) should be revised as follows:

$$\begin{aligned} E(\boldsymbol{v},\boldsymbol{h}) = &-\sum_i \frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_j b_j h_j \\ &-\sum_i\sum_j \frac{v_i}{\sigma_i} h_j w_{ji}, \end{aligned} \tag{14}$$

$$p(v_i|\boldsymbol{h};\boldsymbol{\theta},\boldsymbol{a}) = N\left(a_i + \sigma_i \sum_j h_j w_{ji}, \sigma_i^2\right), \tag{15}$$

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

As shown in Fig. 3b, the last RBM, RBM$_{L-2}$, consists of three layers of the DBA for the reason that the DBA goes through supervised training in our proposal [32]. 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, $\boldsymbol{y}$. And its hidden layer is the top hidden layer of the DBA. The structure of RBM$_{L-2}$ is shown in Fig. 3e and its energy function is expressed as follows. To keep consistent with the other RBMs, we use $\boldsymbol{v}$ and $\boldsymbol{h}$ to denote RBM$_{L-3}$'s hidden layer and the top hidden layer, respectively

$$\begin{aligned} E(\boldsymbol{v},\boldsymbol{h},\boldsymbol{y}) = &-\sum_i a_i v_i - \sum_j b_j h_j - \sum_k c_k y_k \\ &-\sum_i\sum_j h_j w_{ji} v_i - \sum_j\sum_k h_j w_{jk} y_k, \end{aligned} \tag{16}$$

where $\boldsymbol{y}$ represents the vector in the output layer. c_k is the bias of the unit y_k. w_{jk} represents the weight of the link connecting the units h_j and y_k.

As the units in $\boldsymbol{v}$ and $\boldsymbol{y}$ are independent on each other, the conditional distribution of the concatenated vector consisting of $\boldsymbol{v}$ and $\boldsymbol{y}$ is,

$$\begin{aligned} p(\boldsymbol{v},\boldsymbol{y}|\boldsymbol{h};\boldsymbol{\theta},\boldsymbol{a}) &= p(\boldsymbol{v}|\boldsymbol{h};\boldsymbol{\theta},\boldsymbol{a})p(\boldsymbol{y}|\boldsymbol{h};\boldsymbol{\theta},\boldsymbol{a}) \\ &= \prod_i p(v_i|\boldsymbol{h};\boldsymbol{\theta},\boldsymbol{a}) \prod_k p(y_k|\boldsymbol{h};\boldsymbol{\theta},\boldsymbol{a}). \end{aligned} \tag{17}$$

We use the method mentioned above to train each RBM. The value of the visible layer of the first RBM is $\boldsymbol{x}$ 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 $\boldsymbol{\theta}$ is nearly optimum. Then the method of backpropagation is utilized to fine-tune the DBA. Our purpose of supervised training is to minimize the difference between the output of the DBA (denoted by $h_{\boldsymbol{\theta}}(\boldsymbol{x})$) and the labeled output $\boldsymbol{y}$. We use the cross-entropy cost function to measure their difference given in Equation (18) [28]

$$\begin{aligned} C(\boldsymbol{\theta}) = &-\frac{1}{m}\sum_{t=1}^{m}(\boldsymbol{y}^{(t)}\log(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(t)})) \\ &+ (1-\boldsymbol{y}^{(t)})\log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(t)}))) \\ &+\frac{\lambda}{2}\sum_{l=2}^{L}\sum_{j=1}^{n_l}\sum_{i=1}^{n_{l-1}}(w_{ji}^{(l)})^2. \end{aligned} \tag{18}$$

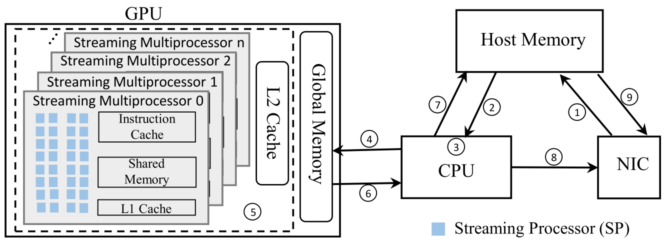


Fig. 4. The architecture of GPUs and steps of how packets are passed in the GPU-accelerated SDR.

Here, $(\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)})$ is the tth training data. $h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(t)})$ denotes the output of the DBA when the parameter of the DBA is $\boldsymbol{\theta}$ and the input is $\boldsymbol{x}^{(t)}$. On the right side of the equation, we can find $C(\boldsymbol{\theta})$ consists of two parts. The first part represents the difference between the output of DBA and the labeled output, and its value is 0 when $\boldsymbol{y}^{(t)} = h_\theta(\boldsymbol{x}^{(t)}) = 0 \text{ } or \text{ } 1$ for all t, otherwise, bigger than 0. The second part is used to keep the training process from overfitting. For minimizing the value of $C(\boldsymbol{\theta})$ in the backpropagation process, we use the gradient descent of $C(\boldsymbol{\theta})$ to update $\boldsymbol{\theta}$ as shown in Equation (19) [33]

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \eta_{bp}\frac{\partial C(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}, \tag{19}$$

where η_{bp} is the learning rate in the backpropagation process.

The fine-tuning algorithm works effectively since the value of $\boldsymbol{\theta}$ 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 $\boldsymbol{\theta}(\boldsymbol{w}, \boldsymbol{b})$. The value of $\boldsymbol{a}$ is not trained in the backpropagation 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.

3.3 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. 4, a GPU consists of the global memory, L2 cache, and several Streaming Multiprocessors (SMs), each of which is composed of many Streaming Processors (SPs). 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 Section 1, the reported line rate of the GPU accelerated SDR based on a common PC has reached as high as 40 Gbps, in this paper, we choose a general PC-based SDR to construct the routing tables and execute our deep learning based routing algorithm. Fig. 4 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. 4, 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 [6]. (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.

4 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. 2. The procedures can be divided into three steps, i.e., initialization, training, and running phases. The details of the three phases are provided below.

4.1 Initialization Phase

In the initialization phase, we need to obtain the data to train our proposed DBAs. As described in Section 3, 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) [34], 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.

4.2 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 $\boldsymbol{\theta}(\boldsymbol{w}, \boldsymbol{b})$ with the backpropagation method. After the training phase, we can obtain the values of $\boldsymbol{\theta}(\boldsymbol{w}, \boldsymbol{b})$.

As described in Section 3.1, 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 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 $(\boldsymbol{x}, \boldsymbol{y})$ 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 backpropagation 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 $\boldsymbol{\theta}(\boldsymbol{w}, \boldsymbol{b})$ nearly reaching the global optimum. Then the backpropagation 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 backpropagation is finished, the value of $\boldsymbol{\theta}(\boldsymbol{w}, \boldsymbol{b})$ of each DBA is recorded.

Algorithm 1. Supervised Train DBA

Input: $(\boldsymbol{x}, \boldsymbol{y}) = \{(\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}) | t = 1, \ldots, m\}$, η_{CD}, η_{bp}, L (number of layers), $\boldsymbol{n} = (n_1, \ldots, n_L)$ (the numbers of units in each layer)
Output: $\boldsymbol{\theta}$

```
1: for $i = 1, \ldots, L-2$ do
2:     TrainRBM($\boldsymbol{u}^{(i)}, \eta_{CD}, n_i, n_{i+1}$)
3: end for
4: Fine-tuneDBA($(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{\theta}, \eta_{bp}$)
5: return $\boldsymbol{\theta}$
```

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

4.3 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 $\boldsymbol{\theta}$ 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 $\boldsymbol{\theta}[N-I][N-1]$ to save the parameters of all the DBAs in the network. Another array $\mathcal{ER}[N-I]$ is used to save the

TABLE 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}	$R_3 \rightarrow R_7 \rightarrow R_{11} \rightarrow R_{15} \rightarrow R_{16}$

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 1 is the routing table built in router R_3, and Fig. 2 shows an example of the process of building the whole path from R_3 to R_{16}.

Algorithm 2. Running Phase

Input: $\mathcal{TP}[N]$, $\boldsymbol{\theta}[N-I][N-1]$, $\mathcal{ER}[N-I]$, sr (the source router).
Output: $\mathcal{NR}[N][N-I-1]$

```
1:  $\boldsymbol{D} \leftarrow \mathcal{ER}[N-I] - sr$
2:  while $\boldsymbol{D} \neq \emptyset$ do
3:      $d \in \boldsymbol{D}$
4:      $s \leftarrow sr$
5:      repeat
6:          $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}[s][d]$
7:          $nr \leftarrow$ run DBA with $\boldsymbol{\theta}'$ and $\mathcal{TP}[N]$
8:          $\mathcal{NR}[s][d] \leftarrow nr$
9:          $s \leftarrow nr$
10:     until $nr = d$
11:     $D \leftarrow D - d$
12: end while
13: return $\mathcal{NR}[N][N-I-1]$
```

5 Complexity 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.

5.1 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 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 $\boldsymbol{w}, \boldsymbol{a}, \boldsymbol{b}$ and $\Delta\boldsymbol{w}, \Delta\boldsymbol{a}, \Delta\boldsymbol{b}$. Then, we repeatedly utilize all training examples to update the values of $\Delta w_{ji}, \Delta a_i, \Delta b_j$ with the method named CD for adjusting $\boldsymbol{w}, \boldsymbol{a}, \boldsymbol{b}$ [30]. 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 $\Delta w_{ji}, \Delta a_i, \Delta b_j$ according to Step 15 to Step 19. Therefore, the values of $\Delta w_{ji}, \Delta a_i$, and Δb_j can be utilized to update w_{ji}, 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).

Algorithm 3. TrainRBM($\boldsymbol{v}, \eta_{CD}, n_v, n_h$)

Input: $\boldsymbol{v}$ ($\boldsymbol{v} = \{\boldsymbol{v}^{(t)} | t = 1, \ldots, m\}$. For RBM$_1$, $\boldsymbol{v} = \boldsymbol{x}$. For RBM$_{L-2}$, $\boldsymbol{v}$ is the activated output of RBM$_{L-3}$ and $\boldsymbol{y}$.
For other RBMs, $\boldsymbol{v}$ of RBM$_i$ is the activated output of RBM$_{i-1}$), n_v, n_h, η_{CD}.
Output: $\boldsymbol{w} = \{w_{ji} | i = 1, \ldots, n_v, j = 1, \ldots, n_h\}$, $\boldsymbol{a} = \{a_i | i = 1, \ldots, n_v\}$, $\boldsymbol{b} = \{b_j | j = 1, \ldots, n_h\}$.

```
1:  for $i = 1, \ldots, n_v, j = 1, \ldots, n_h$ do
2:      $\Delta a_i = 0, \Delta b_j = 0, \Delta w_{ji} = 0,$
3:      $a_i = log\frac{p_i}{1-p_i}, b_j = 0, w_{ji} \sim N(0, 0.01^2).$
4:  end for
5:  repeat
6:      for $t = 1, \ldots, m$ do
7:          for $j = 1, \ldots, n_h$ do
8:              compute $p(h_j = 1|\boldsymbol{v}) = 1/(1 + e^{-(b_j + \sum_{i=1}^{n_v} w_{ji} v_i)})$
9:              sample $h_j$ from $p(h_j = 1|\boldsymbol{v})$
10:         end for
11:         for $i = 1, \ldots, n_v$ do
12:             compute $p(v_i^{'} = 1|\boldsymbol{h}) = 1/(1 + e^{-(a_i + \sum_{j=1}^{n_h} w_{ji} h_j)})$
13:             sample $v_i^{'}$ from $p(v_i^{'} = 1|\boldsymbol{h})$
14:         end for
15:         for $i = 1, \ldots, n_v, j = 1, \ldots, n_h$ do
16:             $\Delta w_{ji} \leftarrow \Delta w_{ji} + p(h_j = 1|\boldsymbol{v}) v_i - p(h_j = 1|\boldsymbol{v}^{'}) v_i^{'}$
17:             $\Delta a_i \leftarrow \Delta a_i + v_i - v_i^{'}$
18:             $\Delta b_j \leftarrow \Delta b_j + p(h_j = 1|\boldsymbol{v}) - p(h_j = 1|\boldsymbol{v}^{'})$
19:         end for
20:     end for
21:     for $i = 1, \ldots, n_v, j = 1, \ldots, n_h$ do
22:         $w_{ji} \leftarrow w_{ji} + \eta_{CD} \Delta w_{ji} / m$
23:         $a_i \leftarrow a_i + \eta_{CD} \Delta a_i / m$
24:         $b_j \leftarrow b_j + \eta_{CD} \Delta b_j / m$
25:     end for
26: until $iter = r_1$
27: return $\boldsymbol{w}, \boldsymbol{a}, \boldsymbol{b}$
```

Since the visible layer of the first RBM satisfies the Gaussian Distribution as mentioned in Section 3.2, for training the first RBM, we need to calculate σ_i and $(a_i + \sigma_i \sum_j h_j w_{ji})$

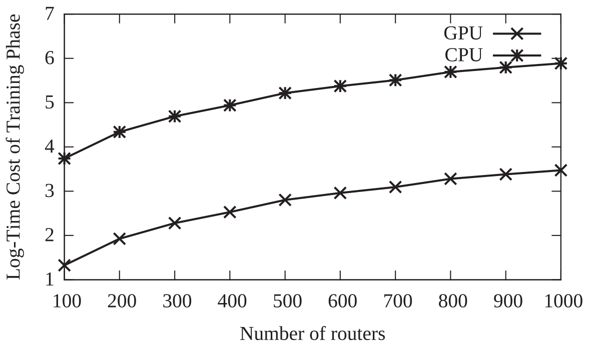


Fig. 5. The time cost of training phase on the chosen GPU and CPU-based SDRs.

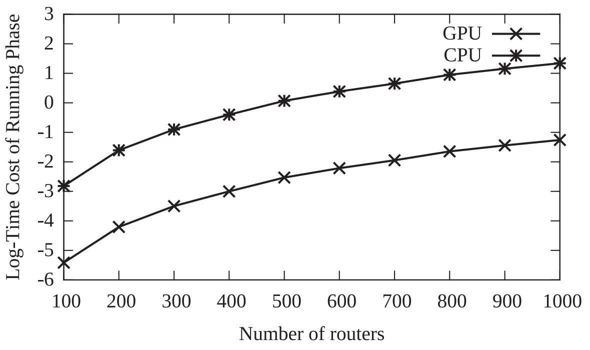


Fig. 6. The time cost of running phase on the chosen GPU and CPU-based SDRs.

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 (15). 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 (18) 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: feedforward pass (Step 3 to Step 11), backpropagation (Step 12 to Step 19), updating the values of $\boldsymbol{w}, \boldsymbol{b}$ (Step 20 to Step 27), and calculating the cost function (Step 29 to Step 43). As shown in the feedforward pass from Step 3 to Step 11, 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 13. 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 15 to Step 19 show how to utilize the error of the lth 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 $\boldsymbol{w}, \boldsymbol{b}$ according to Steps 20 to 27. After obtaining the updated values of $\boldsymbol{w}, \boldsymbol{b}$, we can re-calculate the value of the cost function, C, the procedures for which is shown from Step 29 to 43. Then, it can be confirmed whether a new iteration should be executed according to the value of C. If we assume that the algorithm iterates r_2 times, the total time cost is $r_2((4m+1)\sum_{l=2}^{L} n_l n_{l-1} - mn_1 n_2 + m(\sum_{l=2}^{L} n_l + 2n_L) + 1)$ADD $+\ r_2 m(2\sum_{l=2}^{L} n_l n_{l-1} - n_1 n_2 + \sum_{l=2}^{L} n_l + 4n_L)$ SUB $+\ r_2((8m+1)\sum_{l=2}^{L} n_l n_{l-1} - mn_1 n_2 + m\sum_{l=2}^{L} n_l + 2mn_L + 1)$ MUL $+\ r_2(2\sum_{l=2}^{L} n_l + 2)$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 [35]. 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. 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.

5.2 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 period 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 11 in Algorithm 4. As mentioned earlier in Section 4.3, 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. 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.

Algorithm 4. Fine-Tune the DBA$((\boldsymbol{x},\boldsymbol{y}),\boldsymbol{w},\boldsymbol{b},\eta_{bp})$

Input: $(\boldsymbol{x}^{(t)},\boldsymbol{y}^{(t)})$, $\boldsymbol{w}=(\boldsymbol{w}^{(2)},\ldots,\boldsymbol{w}^{(L-1)})$, $\boldsymbol{w}^{(l)}=\{w_{ji}^{(l)}|i=1,\ldots,n_{l-1}, j=1,\ldots,n_l\}$, $\boldsymbol{b}=(\boldsymbol{b}^{(2)},\ldots,\boldsymbol{b}^{(L)})$, $\boldsymbol{b}^{(l)}=(b_1^{(l)},\ldots,b_{n_l}^{(l)})$, η_{bp}.
Output: $\boldsymbol{w},\boldsymbol{b}$.

```
1: repeat
2:   for $t=1,\ldots,m$ do
3:     for $j=1,\ldots,n_1$ do
4:       initialize the units, $z_j^{(1)}$, of the input layer with $x_j^{(t)}$
5:     end for
6:     for $l=2,\ldots,L$ do
7:       for $j=1,\ldots,n_l$ do
8:         $z_j^{(l)}=\sum_{i=1}^{n_{l-1}} w_{ji}^{(l)}u_i^{(l-1)}+b_j^{(l)}$
9:         $u_j^{(l)}=1/(1+e^{-z_j^{(l)}})$
10:      end for
11:    end for
12:    for $i=1,\ldots,n_L$ do
13:      $\delta_i^{(L)}=u_i^{(L)}-y_i^{(t)}$
14:    end for
15:    for $l=L-1,\ldots,2$ do
16:      for $i=1,\ldots,n_l$ do
17:        $\delta_i^{(l)}=\sum_{j=1}^{n_{l+1}} w_{ji}^{(l+1)}\delta_j^{(l+1)}u_i^{(l)}(1-u_i^{(l)})$
18:      end for
19:    end for
20:    for $l=2,\ldots,L$ do
21:      for $j=1,\ldots,n_l$ do
22:        $b_j^{(l)}\leftarrow b_j^{(l)}-\eta_{bp}\delta_j^{(l)}$
23:        for $i=1,\ldots,n_{l-1}$ do
24:          $w_{ji}^{(l)}\leftarrow w_{ji}^{(l)}-\eta_{bp}(\delta_i^{(l)}u_i^{(l-1)}+\lambda w_{ji}^{(l)})$
25:        end for
26:      end for
27:    end for
28:  end for
29:  for $t=1,\ldots,m$ do
30:    for $l=2,\ldots,L$ do
31:      for $j=1,\ldots,n_l$ do
32:        $z_j^{(l)}=\sum_{i=1}^{n_{l-1}} w_{ji}^{(l)}u_i^{(l-1)}+b_j^{(l)}$
33:        $u_j^{(l)}=1/(1+e^{-z_j^{(l)}})$
34:      end for
35:    end for
36:    for $j=1,\ldots,n_L$ do
37:      $C_1\leftarrow C_1-(y_j^{(t)}\log u_j^{(L)}+(1-y_j^{(t)})\log(1-u_j^{(L)}))$
38:    end for
39:  end for
40:  for $l=2,\ldots,L, j=1,\ldots,n_l, i=1,\ldots,n_{l-1}$ do
41:    $C_2\leftarrow C_2+(w_{ji}^{(l)})^2$
42:  end for
43:  $C=\frac{1}{m}C_1+\frac{\lambda}{2}C_2$
44: until convergence
45: return $\boldsymbol{w},\boldsymbol{b}$
```

6 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 [16] is utilized since it provides the library of DBAs, which is not available in other simulators such as Caffe and Microsoft Cognitive Toolkit [36]. 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. 2 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 Section 3.1, 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 [34] consist of a mix of routing protocols, which are difficult to use for supervised training. Moreover, as the goal of this paper 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.

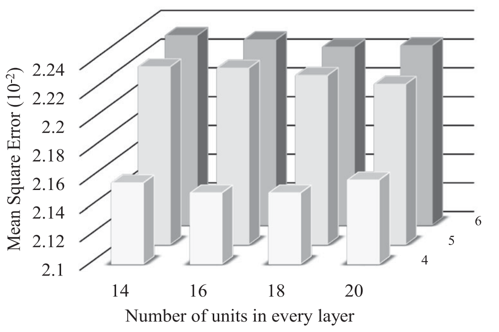


Fig. 7. Mean square errors (MSEs) of different DBAs.

TABLE 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	33,792	Path	-
16	256	Path	70%
16	16	Path	45%
16	**16**	**Next node**	**5%**
48	16	Next node	5%

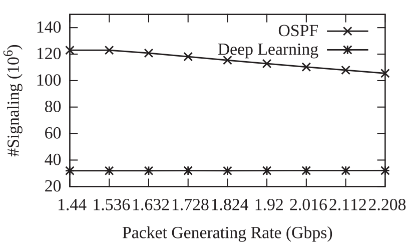
6.1 DBA Precision Analysis

Since the number of routers in the network shown in Fig. 2 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. 7. It can be noticed that the DBAs consisting of four 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.

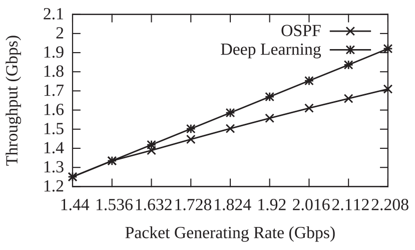
To elucidate the input and output of our proposed deep learning system model, please refer to Table 2. This table gives the error rates of five structures in a network with 16 routers. The first row represents the centralized routing control. The input is the traffic pattern in the last time interval 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 percent, 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 percent. 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.

6.2 Network Performance Analysis

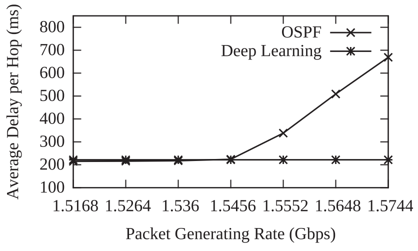
In the running phase, we choose OSPF as a benchmark method to compare the proposed deep learning based routing strategy. To compare the performance under various network loads, we change the data generating rate and record the values of network signaling overhead, throughput, and average delay per hop. The signaling interval is fixed at 0.25 second. Figs. 8a and 8b 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 to 2.208 Gbps. Fig. 8c compares the variation of average delay per hop under two scenarios when the data generating rate increases from 1.5168 to 1.5744 Gbps. In Fig. 8a, 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



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

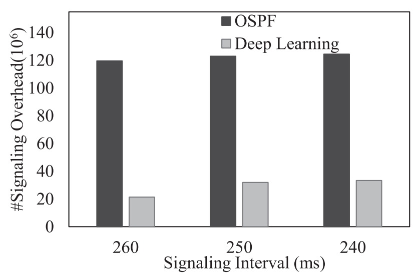


(b) Comparison of aggregate throughput for the conventional OSPF and the proposed deep learning system.

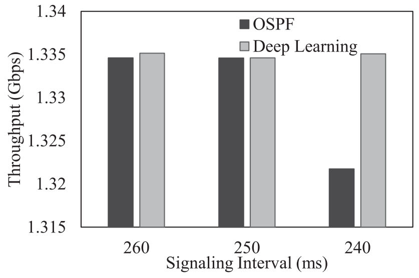


(c) Comparison of average delay per hop for the conventional OSPF and the proposed deep learning system.

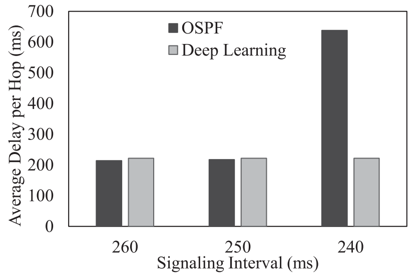
Fig. 8. Comparison of network performance under different network loads in our proposal and the bencmark method (OSPF) in terms of signaling overhead, throughput, and average delay per hop.



(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.

Fig. 9. 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.

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. 8b 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 in performance in the two routing strategies is more clearly shown in Fig. 8c 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 to 240 ms. Fig. 9 show the result consisting of the signaling overheads, throughput, and average delay per hop for the two cases when the signaling intervals are 260, 250, and 240 ms, respectively. In Fig. 9a, we can find that the signaling overheads in our proposal are much lower than those in the case with OSPF. In Figs. 9b and 9c, we can clearly see the effects of signaling overheads on the performance of the two cases. In Fig. 9b, 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 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. 9c 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 or 250 ms. Moreover, we can find that when the signaling interval is 260 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.

7 Conclusion

In this paper, we explained the importance to rethink the core router architectures and backbone network routing strategies to meet the changing network requirements and deal with the significant traffic growth in days to come. In this vein, 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. As various terminals access the network for different kinds of services, our future research will attempt to apply the deep learning techniques in modeling the complex relationships among multiple network metrics for better routing path management.

REFERENCES

[1] S. Das, G. Parulkar, and N. McKeown, "Rethinking IP core networks," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 5, no. 12, pp. 1431–1442, Dec. 2013.

[2] 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 Trans. Comput.*, vol. 64, no. 12, pp. 3488–3499, Dec. 2015.

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

[4] Cisco, "The Cisco flow processor: Cisco's next generation network processor solution overview," Jan. 2014. [Online]. Available: www.cisco.com/c/en/us/products/collateral/routers/asr-1000-series-aggregation-services-routers/solution_overview_c22-448936.html

[5] G. Vasilidadis, L. Koromilas, M. Polychronakis, and S. Ioannidis, "GASPP: A GPU-accelerated stateful packet processing framework," in *Proc. USENIX Annu. Tech. Conf.*, Philadelphia, PA, USA, 2014, pp. 321–332.

[6] M. Mukerjee, D. Naylor, and B. Vavala, "Packet processing on the GPU," Technical Report. Comput. Sci. Dept., Carnegie Mellon Univ., Pittsburgh, PA, USA.

[7] S. Han, K. Jang, K. Park, and S. Moon, "Building a single-box 100 Gbps software router," in *Proc. 17th IEEE Workshop Local Metropolitan Area Netw.*, 2010, pp. 1–4.

[8] S. Mu, X. Zhang, N. Zhang, J. Lu, Y. S. Deng, and S. Zhang, "IP routing processing with graphic processors," in *Proc. Conf. Des. Autom. Test Europe*, 2010, pp. 93–98.

[9] Cisco, "The Zettabyte Era Trends and analysis," Jun. 2016. [Online]. Available: www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html

[10] B. Zhang, J. Hao, and H. T. Mouftah, "Bidirectional multi-constrained routing algorithms," *IEEE Trans. Comput.*, vol. 63, no. 9, pp. 2174–2186, Sep. 2014.

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[12] N. Kato, M. Suzuki, S. Omachi, H. Aso, and Y. Nemoto, "A handwritten character recognition system using directional element feature and asymmetric Mahalanobis distance," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, no. 3, pp. 258–262, Mar. 1999.

[13] Y. C. Ouyang and A. A. Bhatti, "Neural network-based routing in an intelligent computer communication network," in *Proc. IEEE 22nd Southeastern Symp. Syst. Theory*, 1990, pp. 444–448.

[14] A. Raniwala and T. Chiueh, "Evaluation of a wireless enterprise backbone network architecture," in *Proc. 12th Annu. IEEE Symp. High Performance Interconnects*, 2004, pp. 98–104.

[15] M. Barabas, G. Boanea, R. A. Bogdan, and V. Dobrota, "Congestion control based on distributed statistical QoS-aware routing management," *Przeglad Elektrotechniczny*, vol. 89, no. 2b, pp. 251–256, 2013.

[16] N. Kato, et al., "The deep learning vision for heterogeneous network traffic control–Proposal, challenges, and future perspective," *IEEE Wireless Commun. Mag.*, 2016, doi: 10.1109/MWC.2016.1600317WC.

[17] Z. M. Fadlullah, et al., "State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems," *IEEE Commun. Surveys Tuts.*, 2017, doi: 10.1109/COMST.2017.2707140.

[18] Google DeepMind, "AlphaGo," (2017, May). [Online]. Available: https://deepmind.com/alpha-go

[19] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.

[20] A. Sivaraman, et al., "Programmable packet scheduling at line rate," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 44–57.

[21] D. Mihai, et al., "RouteBricks: exploiting parallelism to scale software routers," in *Proc. ACM SIGOPS 22nd Symp. Operating Syst. Principles*, USA, 2009, pp. 15–28.

[22] Y. Wang, M. Martonosi, and L. S. Peh, "Supervised learning in sensor networks: New approaches with routing, reliability optimizations," in *Proc. 3rd Annu. IEEE Commun. Soc. Sensor Ad Hoc Commun. Netw.*, 2006, pp. 256–265.

[23] B. Schölkopf and A. J. Smola, *Learning with Kernels*, MIT Press, USA, 2002.

[24] G. E. Hinton, "What kind of a graphical model is the brain?" in *Proc. 19th Int. Joint Conf. Artif. Intell.*, 2005, pp. 1765–1775.

[25] T. N. Sainath, A.-R. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for LVCSR," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2013, pp. 8614–8618.

[26] The iBrain is here and it's already inside your phone, Aug. 2016. [Online]. Available: https://backchannel.com/an-exclusive-look-at-how-ai-and-machine-learning-work-at-apple-8dbfb131932b#.43bf9cm00

[27] W. Huang, G. Song, H. Hong, and K. Xie, "Deep architecture for traffic flow prediction: Deep belief networks with multitask learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 15, no. 5, pp. 2191–2201, Oct. 2014.

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: www.deeplearningbook.org

[29] A. Fischer and I. Christian, "An introduction to restricted Boltzmann machines," in *Progress Pattern Recog., Image Anal., Comput. Vis., Appl.*, Springer, 2012, pp. 14–36.

[30] G. E. Hinton,"Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, no. 8, pp. 1771–1800, 2002.

[31] G. Hinton, "A practical guide to training restricted Boltzmann machines," in *Neural Networks: Tricks of the Trade*. Berlin, Germany: Springer, 2012, pp. 599–619.

[32] H. Goh, N. Thome, M. Cord, and J.-H. Lim, "Top-down regularization of deep belief networks," in *Proc. Advances Neural Inf. Process. Syst.*, 2013, pp. 1878–1886.

[33] Q. Zhang, L. T. Yang, and Z. Chen, "Privacy preserving deep computation model on cloud for big data feature learning," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1351–1362, May 2016.

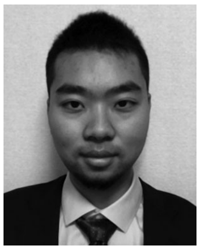
[34] CAIDA. [Online]. Available: www.caida.org/home/, Accessed on: May 2017.

[35] A. Fog, "Instruction Tables: Lists of Instruction Latencies, Throughputs and Micro-Operation Breakdowns for Intel, AMD and VIA CPUs," Technical University of Denmark, pp. 231–247, 2012.

[36] Comparison of deep learning software. [Online]. Available: https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software, Accessed on: May 2017.

Bomin Mao (S'15) received the BSc degree in telecommunications engineering and MS degree in electronics and telecommunications engineering from Xidian University, China, in 2012 and 2015, respectively. Currently, he is working toward the PhD degree in the Graduate School of Information Sciences, Tohoku University, Japan. His research interests include involving wireless networks, software defined networks, quality of service, particularly with applications of machine intelligence and deep learning. He is a student member of the IEEE.

Zubair Md. Fadlullah (M'11-SM'13) received the BSc (with Hons.) degree in computer science and information technology from the Islamic University of Technology (IUT), Bangladesh, in 2003, and the MSc and PhD degrees in applied information science from Tohoku University, in 2008 and 2011, respectively. He is currently an associate professor in the Graduate School of Information Sciences, Tohoku University, Japan. His research interests include the areas of 5G, smart grid, network security, intrusion detection, game theory, quality of security service provisioning mechanism, and deep learning. He received the Dean's Award and the President's Award from Tohoku University in 2011, the IEEE Asia Pacific Outstanding Researcher Award in 2015, and the NEC Foundation Prize for research contributions in 2016. He also received several best paper awards in the Globecom, IC-NIDC, and IWCMC conferences. He is a senior member of the IEEE.



Fengxiao Tang (S'15) received the BE degree in measurement and control technology and instrument from the Wuhan University of Technology, Wuhan, China, in 2012 and the MS degree in software engineering from Central South University, Changsha, China, in 2015. Currently, he is working toward the PhD degree at GSIS, Tohoku University, Japan. His research interests include unmanned aerial vehicles system, game theory optimization, and deep learning. He is a student member of the IEEE.

Nei Kato (F'13) is a full professor and the director of Research Organization of Electrical Communication (ROEC), Tohoku University, Japan. He has been engaged in research on computer networking, wireless mobile communications, satellite communications, ad hoc & sensor & mesh networks, smart grid, IoT, big data, and pattern recognition. He has published more than 350 papers in prestigious peer-reviewed journals and conferences. He is the editor-in-chief of the *IEEE Network Magazine* (2015.7-), the associate editor-in-chief of the *IEEE Internet of Things Journal* (2013-), an area editor of the *IEEE Transactions on Vehicular Technology* (2014-), and the chair of the *IEEE Communications Society Sendai Chapter*. He served as a Member-at-Large on the Board of Governors, IEEE Communications Society (2014-2016), a vice chair of Fellow Committee of IEEE Computer Society (2016), a member of IEEE Computer Society Award Committee (2015-2016) and IEEE Communications Society Award Committee (2015-2017). He has also served as the chair of Satellite and Space Communications Technical Committee (2010-2012) and Ad Hoc & Sensor Networks Technical Committee (2014-2015) of IEEE Communications Society. His awards include Minoru Ishida Foundation Research Encouragement Prize (2003), Distinguished Contributions to Satellite Communications Award from the IEEE Communications Society, Satellite and Space Communications Technical Committee (2005), the FUNAI information Science Award (2007), the TELCOM System Technology Award from Foundation for Electrical Communications Diffusion (2008), the IEICE Network System Research Award (2009), the IEICE Satellite Communications Research Award (2011), the KDDI Foundation Excellent Research Award (2012), IEICE Communications Society Distinguished Service Award (2012), IEICE Communications Society Best Paper Award (2012), Distinguished Contributions to Disaster-resilient Networks R&D Award from Ministry of Internal Affairs and Communications, Japan (2014), Outstanding Service and Leadership Recognition Award 2016 from IEEE Communications Society Ad Hoc & Sensor Networks Technical Committee, Radio Achievements Award from Ministry of Internal Affairs and Communications, Japan (2016), and Best Paper Awards from IEEE ICC/GLOBECOM/WCNC/VTC. He is a Distinguished Lecturer of IEEE Vehicular Technology Society. He is a fellow of the IEICE and the IEEE.

Osamu Akashi received BSc and MSc degrees in information science from the Tokyo Institute of Technology, in 1987 and 1989, respectively, and the PhD degree in mathematical and computing sciences from the Tokyo Institute of Technology, in 2001. He joined the Nippon Telegraph and Telephone Corporation (NTT) Software Laboratories, in 1989, and is a senior research scientist in the NTT Network Innovation Laboratories. His research interests include the areas of distributed systems, multi-agent systems, and network architectures. He is a member of the ACM, the IEICE, the IPSJ, and the JSSST.

Takeru Inoue received the BE, ME, and PhD degrees from Kyoto University, Japan, in 1998, 2000, and 2006, respectively. He is a distinguished researcher in the NTT Network Innovation Laboratories. He was an ERATO researcher at the Japan Science and Technology agency from 2011 through 2013. His research interests widely cover algorithmic approaches in computer networks.

Kimihiro Mizutani (M'11) received the MS degree in information system from the Nara Institute Science and Technology, in 2010. He is a researcher in the NTT Network Innovation Labs. His research interest is future internet architecture. He received the best student paper from International Conference on Communication Systems and Application (ICCSA) in 2010. He also received the research awards from IPSJ and IEICE in 2010 and 2013, respectively. He is a member of the IEICE and the IEEE Communication Society.