Toward Greater Intelligence in Route Planning: A Graph-Aware Deep Learning Approach

Zirui Zhuang, *Student Member, IEEE*, Jingyu Wang, *Member, IEEE*, Qi Qi, *Member, IEEE*, Haifeng Sun, *Member, IEEE*, and Jianxin Liao, *Member, IEEE*

***Abstract*—Software-defined networking decouples the control plane and data plane, which grants more computing power for routing computations. Traditional routing methods suffer from the complex dynamics in networking, and they are facing issues such as slow convergence and performance decline. Deep learning techniques have shown preliminary results on solving the routing problem, and bringing more accuracy, precision, and intelligence compared with traditional modeling techniques. However, the existing deep learning architectures are not built to learn from the crucial topological relations between forwarding nodes, which restricts the model's ability to handle different network conditions. In this paper, we propose a deep learning based intelligent routing strategy with revised graph-aware neural networks, which learns topological information efficiently. In addition, a set of features suitable for network routing are designed so that the networking state are well represented upon each routing decision. In experiments, the performance of the proposed work is demonstrated with a real-world topology and the production level software switches. The execution time is evaluated on various kinds of network topology and different network scales. Also, the simulation result shows that the proposed work is more accurate and efficient compared to the state-of-the-art routing strategy.**

***Index Terms*—Machine learning, routing, software-defined networking, traffic control.**

I. Introduction

With the emerging advancements in software-defined networking, the network controllers have more computing power nowadays, which builds the basis of machine learning applications in networking [1], [2]. With OpenFlow [3] and P4 [4], the network achieves better controllability [5]. Furthermore, more aspects of and more accuracy in network state are accessible to the network controllers by recent network measurement evolvements [6]. These state measurements improve the observability of the networks [7]. As controllability and observability are essential to complex networked systems [8], [9], the progress in networking leads to the opportunity for more optimal routing decisions. Recently, many machine learning applications in the networking area are proposed, such as attack detection [10], traffic classification [11], and resource management [12].

Network states and dynamics are sophisticated, and situations shift rapidly. There are some factors which are crucial to the strategy of network routing.

1) The complex dynamics of the networking process: Communication networks are complex systems, which consists of many networking devices connected by transmission links. The state of a network can be described by the metrics of each device, each link, and the topology itself. Each state has its effect on overall network performance. For instance, the latency affects networking efficiency by defeating network protocols [13], and the network load has a negative impact on performance [14]. Combining all these aspects, we can find that the network state space is a high-dimensional continuous space, which is of great complexity.
2) The performance persistence: Each routing decision has an active period, though the network state changes swiftly. As a result, a learning model with long-term stability is needed [15]. However, a mapping between the current network state and an optimal future outcome cannot be achieved by a simple linear approximation. Due to the complexity of state space, it is of great difficulty to deduce an explicitly defined model.
3) The convergence speeds: The diversification in incoming traffic patterns and situations such as link failure and congestion have significant influence on the underlying networking process. When such previously unseen situations are in appearance, the routing strategy should be able to converge quickly.

Traditional routing techniques, such as open shortest path first protocol (OSPF), ant colony optimization, particle swarm optimization, and Q-learning, leverage a little state information so that these methods are still not able to cope with the complexity in the network state and nonstationarity in networking. Deep learning techniques are ideal for dealing with these kinds of obstacles and thus, at this point, it is significant to introduce deep learning solutions into network routing.

There has been a recent work on network routing using deep learning techniques [16], which shows pretty good results on

Manuscript received December 13, 2018; revised April 15, 2019 and June 6, 2019; accepted June 7, 2019. Date of publication June 25, 2019; date of current version June 3, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61771068, Grant 61671079, and Grant 61471063, in part by the Beijing Municipal Natural Science Foundation under Grant 4182041, in part by the National Basic Research Program of China under Grant 2013CB329102, in part by the Fundamental Research Funds for the Central Universities under Grant 2018RC20, and in part by the Ph.D. Student Short-Term Exchange Program of Beijing University of Posts and Telecommunications. *(Corresponding authors: Jingyu Wang; Jianxin Liao.)*

The authors are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: zhuangzirui@bupt.edu.cn; wangjingyu@bupt.edu.cn; qiqi8266@bupt.edu.cn; hfsun@bupt.edu.cn; liaojx@bupt.edu.cn).

Digital Object Identifier 10.1109/JSYST.2019.2922217

simple topologies yet not on complex ones. Also, there have been works on bringing graph information into learning models in machine learning community. To be specific, graph-aware learning techniques have been successfully applied on tasks such as molecular feature extraction in chemistry [17], text categorization in natural language processing [18], and node classification in social networks [19]. Empirical results of these cases show that a learning model benefits extensively from graph-related information. As a result, we believe that the performance of deep learning based routing strategies can be further increased by making the deep learning architecture aware of a network's characteristics.

Therefore, we propose a novel graph-aware deep learning based intelligent routing strategy (GADL), to unravel the issues mentioned above. In the proposed strategy, a precise measure is maintained for the network state, as well as the current position and destination of forwarding tasks. The network state information is preprocessed into features in a representative form as the input of a revised deep learning architecture. The deep learning architecture consists of a graph kernel and a convolutional neural network (CNN). The learning model in this paper is subsequently trained from offline supervised learning and then tested online. In summary, the main contributions are as follows.

1) A centralized deep learning model is used for routing computation, which reduces overhead in networking and model training.
2) A feature processing procedure is designed, which transforms network state measurements into representative features suitable for a deep learning architecture to learn in a routing scenario.
3) A GADL architecture is proposed, which improves efficiency by introducing network topology awareness.

The organization of this paper is as follows. In Section II, we survey related work. In Section III, we define the target problem. In Section IV, we propose a GADL-based intelligent routing strategy. In Section V, experiments and results are presented. Finally, in Section VI, we conclude the paper and give a discussion.

II. Related Work

A. Routing Methods in Communication Networks

The communication networks are known to be complex networked systems [20], where the controllability [9] and observability [8] are vital to the effectiveness and efficiency. On the controllability, OpenFlow [3] brings a programmable network by decoupling the network's control-plane and data-plane, as P4 [4] additionally enables programming in data-plane. On the observability, recent works [6] are capable of achieving better network measurements.

Traditional routing protocols mainly use adjacency as the network state and hop count as metrics to solve the routing problem, and thus, the performance is limited. For instance, OSPF exchanges a link state between switches to elect a trajectory of designated routers with a minimum summed weight. Unfortunately, these protocols are known to have slow convergence issues [21]. Multiprotocol label switching [22] and segment routing [23] are proposed later as media for better traffic engineering, but still it is hard to find (near-)optimal solution with little expenses in time.

Optimization techniques, such as ant colony optimization [24] and particle swarm optimization [25], are seen targeting specific metrics, such as the end-to-end delay, link energy consumption, and packet loss. These methods try to utilize the observable network measurements for routing computation. However, these algorithms are computationally expensive and cannot be expedited by general hardware accelerators.

In our previous work, dimensions in regard to network resource management [26] and traffic engineering [27] are explored within the scope of service routing and software-defined networking. We also noticed that there is a trending interest in using machine learning to solve network route planning and management problems. Sharma *et al.* [28] use neural networks and decision trees to find next forwarding hop with minimum delay in opportunistic networks. With the computing power enabled by GPU, deep learning algorithms are increasingly powerful. Recently, there has been literature on using deep learning methods in network routing. Lee [29] combines deep learning with traditional methods. The work uses deep learning to estimate node degree, and then use the Viterbi algorithm for route selection. The performance would benefit from an end-to-end deep learning model without traditional methods. Valadarsky *et al.* explores various possible directions in machine learning based network routing [30], giving an inspiring discussion and preliminary results on this topic. Li *et al.* developed a deep reinforcement learning based flow scheduling system, which mimics the neural system in animals [31]. Xu *et al.* also used deep reinforcement learning in traffic engineering, which can be seen as a form of oblivious routing as it utilizes precomputed paths for selection [32].

The most state-of-the-art work on network routing is the deep belief architecture (DBA) from Mao *et al.* [16]. The DBA is a distributed deep learning based method, which takes the packet rates of each node as input and produces next forwarding hop as output. DBA works quite well when optimal path length is short, such as on a chessboard-like topology. When the topology grows more complex and path length increases as well, DBA's performance drops. Tang *et al.* also used convolutional neural networks for intelligent traffic control in wireless networks [33].

From an algorithmic perspective, it is known that, in general, it is impossible to find a constant ratio approximation for the routing tasks in a time-efficient way, although the approximation can be found in the end [34].

B. Techniques in Machine Learning

A deep belief network [35] is a generative model, which uses Gibbs sampling to adjust weights between neurons, and it is usually considered as a dimensionality reduction tool for sparse features. Deep neural network (DNN) with fully connected feed-forward layers using rectifying neurons [36] (also known as rectified linear unit, ReLU) achieves a better result for supervised learning tasks, eliminating the need of time-consuming unsupervised pretraining with Gibbs sampling. CNN uses a small convolution filter to scan different parts of each layer's input data, and since the neural network now only needs to fine-tune

the weights inside the filter, the cost of the training process is tremendously reduced. Convolutional neural networks are seen as successful applications in image classification, which deals with two-dimensional input features [37].

It is known in the machine learning community that the learning model can acquire performance boost if it can understand and utilize the input features in a way consistent with their physical meanings. There is a trending interest in bringing graph-awareness into learning models, and the works have shown promising results. Duvenaud *et al.* [17] use hashing and indexing functions to preprocess the input features so that the output feature of each node contains a portion of information from its neighbors. The design of these functions is application specific and cannot be applied elsewhere without effort. Defferrard *et al.* [18] use graph Fourier transform to bring the computation of convolution into the spectral domain. The limitation of this approach is that a graph has to be predefined and cannot be changed over time, as well as the incapability of dealing with graph edges when there is more than one attribute on each edge. Niepert *et al.* [19] use an intuitive method to find partitions of arbitrary graphs with the help of a labeling function. It can utilize both node and edge features no matter how many features there are. However the output order of features for each node is determined by the labeling function as well, and when the graph changes, the order of the nodes is changed at the same time.

III. Problem Formulation

A communication network consists of many endpoint devices, known as hosts, and multiple data forwarding devices, known as switches. Each host connects to an access switch. In addition, between a pair of connected switches, there is a communication link. Communication is accomplished by exchanging data between the source host and destination host over a subset of connected switches through a routing path made of a finite sequence of chosen links. A routing task is to find an optimal path between source host's access switch and destination host's access switch in a given network topology. For finding valid paths, a routing algorithm needs topology information, such as how the switches are connected. On the other hand, for finding optimal paths, it also requires extra information about how well the switches and links worked, such as switch load, link latency, link throughput, etc.

Now we formulate a mathematical model for the communication network, and the notations of variables in the model can be found in Table I. A communication network can be fully specified by a tuple $G(V, E, \{f^{\mathrm{v}}(u)\}, \{f^{\mathrm{e}}(u,v)\})$, where V is a set of vertices denoting switches, E is a set of edges denoting communication links, $\{f^{\mathrm{v}}(u)\}$ is a set of vertex features denoting the status metrics of each switch $u \in V$, and $\{f^{\mathrm{e}}(u,v)\}$ is a set of edge features denoting the status metrics of each link (u,v), where $u, v \in V$ and $(u,v) \in E$.

Routing from source access switch s to destination access switch t is a procedure of selecting a sequence of edges from E and their endpoints from V in consideration of $f^{\mathrm{v}}(u)$ and $f^{\mathrm{e}}(u,v)$, resulting in a subgraph G', where the degree of s and t is one and the degree of all other vertices is two. Let $\langle s,t\rangle$ denote a routing task from s to t. The direct result of a routing task is a routing path, which is a sequence of forwarding nodes as follows:

$$\langle s,t\rangle = (u_0, u_1, \ldots, u_n) \tag{1}$$

where $u_0 = s$ and $u_n = t$. The ith node in the path of routing task $\langle s,t\rangle$ is

$$\langle s,t\rangle_i = u_i. \tag{2}$$

The routing algorithm selects forwarding nodes one after another iteratively. Let P be the routing procedure. In each iteration, $P(u_n, t, G)$ selects a next hop vertex u_{n+1} connected to current vertex u_n, and then, it executes the next iteration $P(u_{n+1}, t, G)$. Now we have

$$u_{n+1} = P(u_n, t, G) \tag{3}$$

where $u_0 = s$. Combining (2) and (3), we have

$$\langle s,t\rangle_{n+1} = P(\langle s,t\rangle_n, t, G). \tag{4}$$

In general, a routing problem can be defined either using a direct form, as in (1), or using an iteration form, as in (4). For the simplicity, most routing strategies choose to use the latter one, so do we here. As a result, the main interest is to find a procedure P solving the routing path problem defined in (4).

TABLE I
Notations of Variables

Symbol	Description
$\langle s,t\rangle$	A routing task from source s to target t
$\langle s,t\rangle_i$	The i-th node in the resulting path of routing task $\langle s,t\rangle$
$a^{\mathrm{v}}(u)$	The attributes of vertex u
$a^{\mathrm{e}}(u,v)$	The attributes of edge (u,v)
$f^{\mathrm{v}}(u)$	The features of vertex u
$f^{\mathrm{e}}(u,v)$	The features of edge (u,v)
$c^{\mathrm{e}}(u,v)$	The capacity of each link (u,v)
$N_{\mathrm{f}}^{\mathrm{v}}$	The number of vertex features
$N_{\mathrm{f}}^{\mathrm{e}}$	The number of edge features
$\mathbf{M}_{\mathrm{input}}^{\mathrm{v}}$	The input matrix representing vertex features
$\mathbf{M}_{\mathrm{input}}^{\mathrm{e}}$	The input matrix representing edge features
$\mathbf{T}$	The graph-aware operator matrix
$\mathbf{M}_{\mathrm{NOI}}^{\mathrm{v}}$	The neighbors-of-interest vertex feature set matrix
$\mathbf{M}_{\mathrm{NOI}}^{\mathrm{e}}$	The neighbors-of-interest edge feature set matrix

IV. Solution

A. Overview

Network routing problem is hard to find an optimal solution using traditional methods due to the swift change of network states and complex dynamics in networking. Deep learning approaches using neural networks with extraordinary modeling power have shown successful applications in networking areas [10]–[12] and specifically network routing area [16]. We propose a GADL-based intelligent routing strategy.

The system of GADL consists of three phases, as shown in Fig. 1. First, the controller in software-defined networks keeps track of network state measurements from time to time. Second, the switch may request the controller for optimal path upon new traffic flows' arrival. Finally, an intelligent routing model in the controller will calculate an optimal path and then let the controller install it into data plane. The GADL is trained offline

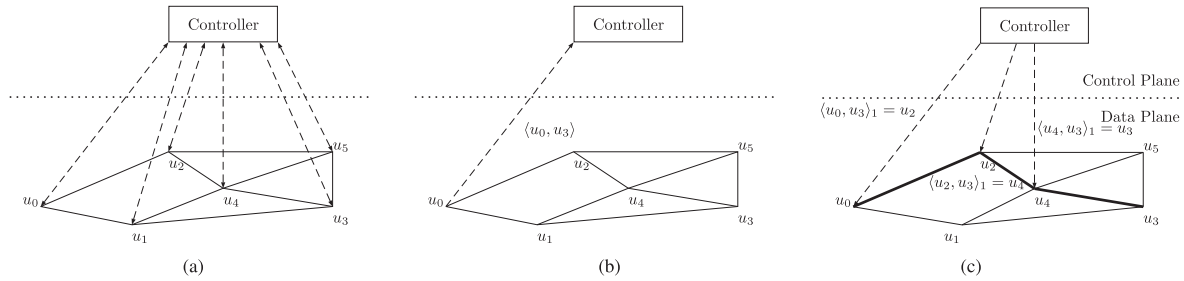

Fig. 1. Proposed routing architecture. (a) Measure network states. (b) Switch requests path. (c) Controller deploys optimal path.

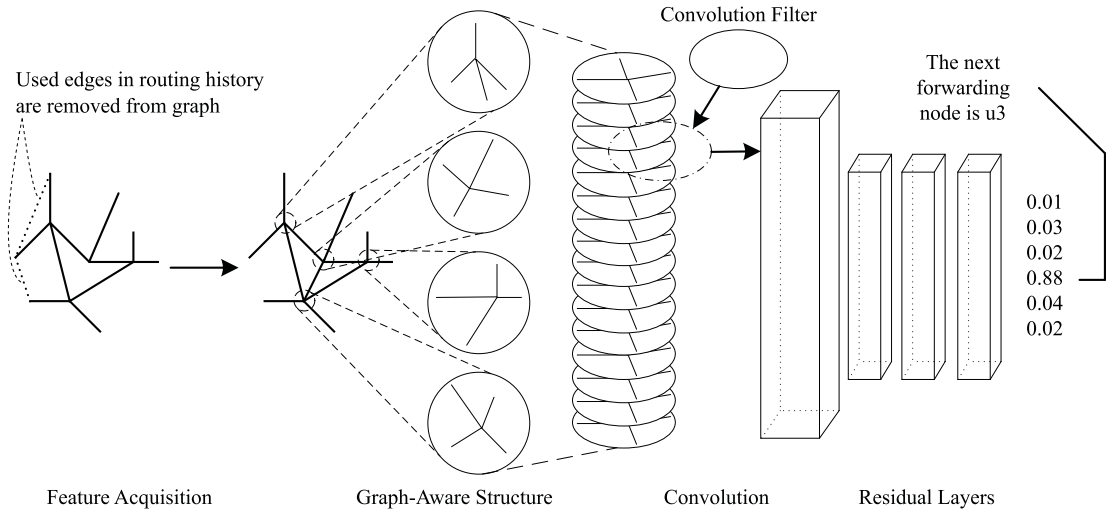

Fig. 2. Proposed graph-aware deep learning based routing model.

by engineering history data of the network and test it online when the training is finished.

Note that in terms of practical implementation, these phases do not need to happen one after another; in other words, these phases may run in parallel. The controller may query the switches and measure network states periodically. The network states are stored inside the controller. The new routing policy may be generated by the GADL model and stored whenever the network states are updated. In this way, the controller only needs to look up the corresponding route from the stored routing policy and deploy them upon receiving requests from switches. Therefore, the routing framework may exploit the power of the GADL model without introducing extra overhead to the system compared with existing software-defined networks.

Fig. 2 demonstrates that GADL takes well-defined features as input, utilizes a graph-aware structure specially customized for network routing, and then estimates the best next hop forwarding switch in one-hot encoding. Here, we use three sets of ResNet alike fully connected blocks in the deep neural structure. There are two key factors in GADL. One is feature acquisition and the other is the neural network setup. We are going to illustrate the proposed solution in these two aspects.

B. Feature Acquisition

In data-driven approaches, a model takes multiple features of a sample as input and produce the probability estimation of classes that the sample should belong to or action that the system should choose to take. Using more samples for training and better features as input can both improve the accuracy of the model. A set of high-quality features can relax the needs of large training samples and thus boost the model's efficiency.

On the one hand, a set of high-quality features should cover as many aspects of the subject as possible. Powered by the software-defined networking, now communication networks are aware of a wide range of network status, as shown in Fig. 1(a). In these networks, accurate measurements can be acquired, such as link connectivity, latency, throughput, and packet rate. As a result, more precise data are available for features to build upon.

On the other hand, high-quality features should be expressive. A straightforward representation of these measurements mentioned above is putting them into an adjacency matrix, which is sufficient to represent the topological information. However other measurements need more expressive signification for machine learning models to understand. In network routing tasks, it is a common knowledge that there is a negative correlation between network performance and loads. The features should be able to express this physical meaning in a monotonic and derivative monotonic way.

Let $a^{\mathrm{v}}(u)$ be the attributes of vertex u, $a^{\mathrm{e}}(u,v)$ be the attributes of edge (u,v), and $c^{\mathrm{e}}(u,v)$ be the full capacity of each link (u,v). Table II lists the raw attributes in use. An adjacency matrix is used to represent network links. For other attributes, the edge features $f^{\mathrm{e}}(u,v)$ are defined as a relative logarithm of

TABLE II
RAW ATTRIBUTES LIST

Attribute	Measurement
a_1^{e}	Link connectivity
a_2^{e}	Link latency
a_3^{e}	Receive bit rate
a_4^{e}	Receive packet rate
a_5^{e}	Transmit bit rate
a_6^{e}	Transmit packet rate

free capacity to the full capacity. The edge features are defined as

$$f_i^{\mathrm{e}}(u,v) = \begin{cases} a_1^{\mathrm{e}}(u,v), & i = 1 \\ \log\left(1 + \frac{c^{\mathrm{e}}(u,v) - a_i^{\mathrm{e}}(u,v)}{c_i^{\mathrm{e}}(u,v)}\right), & i \neq 1. \end{cases} \tag{5}$$

Since network measurements are made on links, the vertex attributes $a^{\mathrm{v}}(u)$ are calculated from the attributes of the edges as

$$a^{\mathrm{v}}(u) = \sum_{v \in \{v | a_1^{\mathrm{e}}(u,v)=1\}} a^{\mathrm{e}}(u,v). \tag{6}$$

In addition, vertex features are defined as a load index relative to its adjacent vertices as

$$f^{\mathrm{v}}(u) = \frac{a^{\mathrm{v}}(u)}{\sum_{v \in \{v | a_1^{\mathrm{e}}(u,v)=1\}} a^{\mathrm{v}}(v)}. \tag{7}$$

To introduce routing history information into the selection procedure, features of used vertices and edges are removed from tuple G. Let V_n' be the set of used vertices at step n calculated as

$$V_n' = \{\langle s,t\rangle_i | 0 \leqslant i < n\}. \tag{8}$$

In this way, the modified tuple G' becomes

$$G_n' = (\{u\}, \{(u,v)\}, \{f^{\mathrm{v}}(u)\}, \{f^{\mathrm{e}}(u,v)\}) \, | u \in V, u \notin V_n'. \tag{9}$$

Finally, the node selection procedure is built with fine-processed features as in

$$\langle s,t\rangle_{n+1} = P(\langle s,t\rangle_n, t, G_n'). \tag{10}$$

C. Graph-Aware Neural Network

In layered DNNs, each layer processes the input matrix and produces an output matrix through a series of matrix operations, and these matrix operations in the same layer are called a neural network structure. Since the convergence speed is essential to network routing, the model should use appropriate structures to be able to train as fast as possible. The convolution kernel is a matrix relatively smaller than the input matrix and each different part in the input matrix is convolved by this kernel to produce the output matrix. In this way, the structural complexity and the number of tunable parameters of the neural network model can be reduced, to make it fit better and train faster.

The traditional CNN has shown successful applications in image processing and pattern recognition [37], in that it tremendously boosts neural networks' performance by sharing weights between neighbor elements of the two-dimensional input matrix, reducing trainable variables, and computational complexity. This is done by using a shared convolution kernel, swiping it along the input matrix, and making matrix multiplications between the convolution kernel and the local part of the input matrix, as shown in the right-hand side of Fig. 3.

In network routing, however, the input is in the form of vectors and adjacency matrix. This leads to the fact that a vertex's actual neighbor in the graph might be far away in the input, which in turn means that traditional convolutional neural networks cannot be applied directly in network routing tasks.

As a result, we design a graph-aware convolutional structure, which can be used specially for network routing scenarios. A graph-aware structure is a kind of neural network structure that can take graph topology into consideration during its matrix operations. The proposed graph-aware convolutional structure first extracts topological information from the graph, then processes the input data based on the extracted information, and finally applies convolution on the processed data, as shown on the left-hand side of Fig. 3. In detail, we rank the neighbors of each root by their importance, and select k neighbors to build a transformation operator for each root. The operators are applied to the input matrix to produce a set of regions of interest with respect to each root. At last a shared convolution kernel is used on these regions of interest to produce the final output of the structure.

In the structure, k neighbors are explored for each root vertex and select w vertices in total. First, we define the generation process of these neighbors. For each vertex u, a tree is built from root u using breadth-first traversal on graph G. With this method, the breadth-first traversal depth relative to vertex u is recorded for vertex v and the depth is noted as D_{uv}. As shown in Fig. 4, a graph's vertices can be grouped into multiple partitions, and in this case, the partitions are

$$\{u, (v_0, v_1, v_2, v_3), (v_4, v_5, v_6), (v_7, v_8, v_9)\}. \tag{11}$$

In each partition, the betweenness centrality is used as a labeling criteria to determine the position of each vertex in the final lineup. The betweenness centrality equals the number of shortest paths from all vertices to all others that pass through the given vertex. It is used in the labeling process because it shows the importance of each vertex in the graph. For convenience, a superscript L denotes a vertex's label, as in v^L. After the labeling procedure, for example, (11) would become

$$\{u, (v_0^3, v_1^1, v_2^2, v_3^3), (v_4^3, v_5^2, v_6^1), (v_7^3, v_8^3, v_9^3)\}. \tag{12}$$

Now the vertices in each partition are reordered according to their labels as

$$\{u, (v_1^1, v_2^2, v_0^3, v_3^3), (v_6^1, v_5^2, v_4^3), (v_7^3, v_8^3, v_9^3)\}. \tag{13}$$

To keep computation stability in case of link failures in networking, w vertices are selected as root with a fixed order identical to the indices of vertices in routing task. For each root vertex, k vertices are chosen from the sequence (13). Let us note the k-vertex sequence as S. Subsequently, the graph-aware neural network will use sequence S to transform the input edge features of shape $[n, n, N_{\mathrm{f}}^{\mathrm{e}}]$ and input vertex features of shape $[1, n, N_{\mathrm{f}}^{\mathrm{v}}]$ into neighbor-of-interest feature sets at each root. The transformation can be implemented as matrix operations in

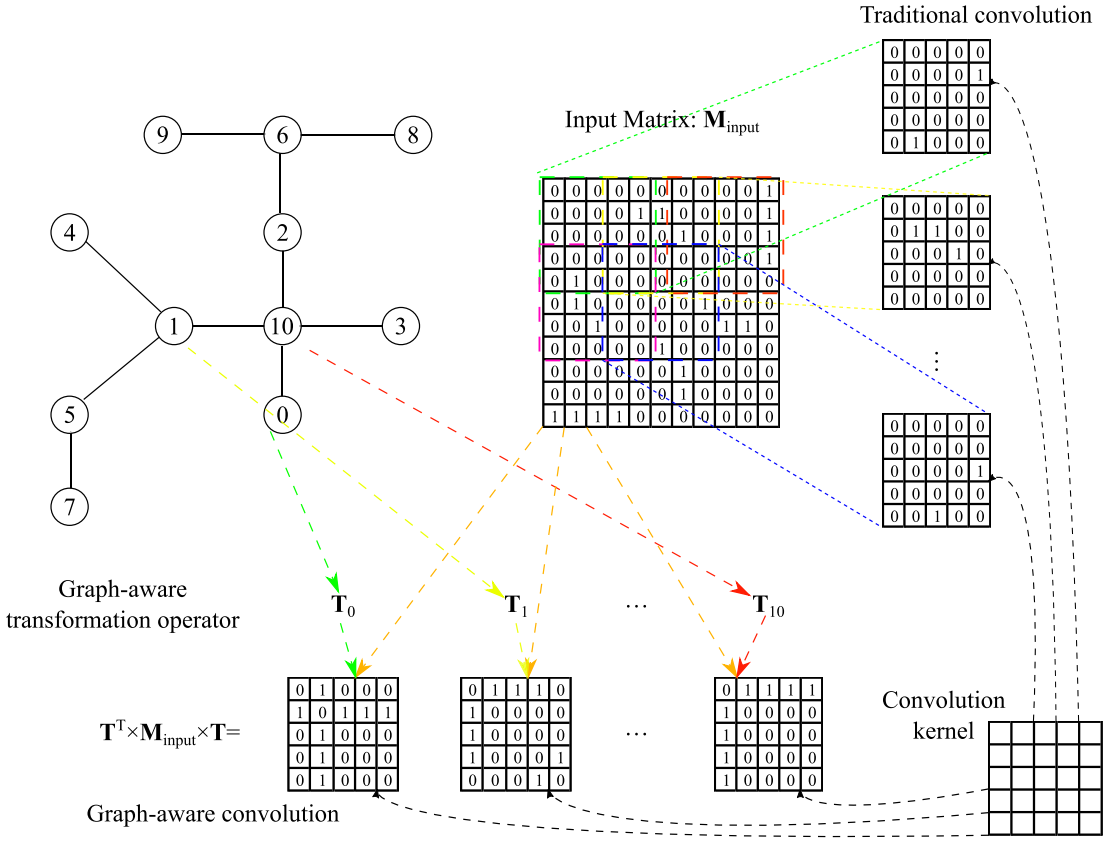

Fig. 3. Comparison between the traditional convolution structure and the proposed graph-aware convolution structure.

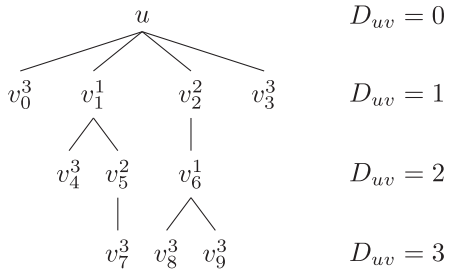

Fig. 4. Breadth-first traversal using vertex u as root.

neural networks. Without loss of generality, we are going to show how this is done when $N_{\mathrm{f}}^{\mathrm{e}}=1$ and $N_{\mathrm{f}}^{\mathrm{v}}=1$. Let $\mathbf{T}$ be the transform matrix of shape $[n,k]$, and each column of $\mathbf{T}$ is the one-hot encoding of each element in S, as follows:

$$\mathbf{T}_{i,j}=\begin{cases}1, & S_j \text{is} v_i\\ 0, & \text{otherwise.}\end{cases} \tag{14}$$

The neighbor-of-interest matrices can be acquired by applying matrix operation $\mathbf{M}_{\mathrm{NOI}}^{\mathrm{e}}=\mathbf{T}^{\mathrm{T}}\times\mathbf{M}_{\mathrm{input}}^{\mathrm{e}}\times\mathbf{T}$ and $\mathbf{M}_{\mathrm{NOI}}^{\mathrm{v}}=\mathbf{M}_{\mathrm{input}}^{\mathrm{v}}\times\mathbf{T}$ for edge features and vertex features, respectively. The transformation will be applied to each and every selected root and feature. Finally, the resulting matrices are of shape $[w,k,k,N_{\mathrm{f}}^{\mathrm{e}}]$ for edge features, and $[w,k,N_{\mathrm{f}}^{\mathrm{v}}]$ for vertex features, where $N_{\mathrm{f}}^{\mathrm{e}}$ and $N_{\mathrm{f}}^{\mathrm{v}}$ are the numbers of input features. As the generation of sequence S only depends on the breadth-first traversal of the graph and the betweenness centrality of each vertex, the sequence S and the transformation matrix $\mathbf{T}$ can be reused across different route computation tasks on condition that the graph topology stayed the same.

Afterward, we apply convolution kernels to the reformed matrices. For vertex features, the matrix of shape $[w,k,N_{\mathrm{f}}^{\mathrm{v}}]$ can be considered as a two-dimensional data with width as w, height as k, and input channel as $N_{\mathrm{f}}^{\mathrm{v}}$. In this way, we can apply a convolution kernel of shape $[1,k]$ to the matrix with a stride by one on the first dimension. For edge features, we can reshape the matrix from $[w,k,k,N_{\mathrm{f}}^{\mathrm{e}}]$ to $[w,k^2,N_{\mathrm{f}}^{\mathrm{e}}]$, and apply a convolution kernel of shape $[1,k^2]$ likewise.

Multiple layers of convolution can be used depending on the size of the problem. In practice, all vertices are selected as root one by one and explored up to the average path length. Also, we use three layers of convolution after the graph-aware transformation operation. In addition, a one-dimensional max-pooling of size two is used after each convolution layer.

At the output layer, a fully connected layer is used with the number of the neurons identical to the number of vertices in the graph, so that each neuron stands for each vertex as the choice. The transformation is done by a matrix operation

$$\boldsymbol{z}=\boldsymbol{W}\boldsymbol{z}'+\boldsymbol{b} \tag{15}$$

where $\boldsymbol{z}'$ is the output of the prior layer, $\boldsymbol{W}$ is a transform matrix, and $\boldsymbol{b}$ is a bias vector. Both $\boldsymbol{W}$ and $\boldsymbol{b}$ are tunable parameters during the phase of training. A sparse SoftMax is also used as the activation function so that the final output is the probability

of choosing each vertex with respect to each given input data. The adjacency vector $\boldsymbol{A_{j,:}}$ is used as the sparse mask as current vertex j. As a result, the final probability for each vertex i being the next vertex chosen is produced by

$$p_i = \frac{\exp(z_i) \cdot A_{j,i}}{\sum_k \exp(z_k) \cdot A_{j,k}}. \tag{16}$$

D. Training the Model

The neural networks provide a mapping between the input and the output data. The mapping is noted as a function

$$\boldsymbol{p} = \pi(\boldsymbol{x}; \boldsymbol{\theta}) \tag{17}$$

where $\boldsymbol{p}$ is the probability vector and $\boldsymbol{\theta}$ are the tunable parameters in the neural networks. Thus, the training of the model means tuning these parameters to reduce the error between model outputs and ground truth values. The ground truth $\boldsymbol{y}$ is a one-hot encoded vector that

$$y_i = \begin{cases} 1, & i \text{ is the true class} \\ 0, & i \text{ is not the true class.} \end{cases} \tag{18}$$

Here, the error is defined by a loss function

$$L(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) = -\sum_i y_i \log p_i + \beta \cdot \|\boldsymbol{\theta}\|_2. \tag{19}$$

The first part of the loss function is the cross entropy between ground truth probability distribution and predicted probability distribution, and the second part is the regularization using L2 norm constraining the complexity of the model to prevent overfitting, where β is a constant that weights the significance of the regularization part.

An efficient optimization algorithm is needed to train the deeper neural networks. A variant of stochastic gradient descent (SGD) algorithm is used in the proposed GADL. In a general SGD algorithm, the parameters are updated along the direction of the gradient $\boldsymbol{g}$ by a learning rate η. At each time step t, the parameters are updated as

$$\boldsymbol{g}_t = \nabla L(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}_t) \text{ w.r.t. } \boldsymbol{\theta}_t \tag{20}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \cdot \boldsymbol{g}_t. \tag{21}$$

A few improvements to SGD are proposed such as Nesterov momentum algorithm [38] and Adam algorithm [39]. The exponential moving averages of both the mean $\boldsymbol{q}^{\mathrm{m}}$ and variance $\boldsymbol{q}^{\mathrm{v}}$ of the gradient are used to find a more stable direction. Let β_1 and β_2 be the exponential decay rate for $\boldsymbol{q}^{\mathrm{m}}$ and $\boldsymbol{q}^{\mathrm{v}}$, respectively. At each time step, the exponential moving averages are calculated as

$$\boldsymbol{q}_t^{\mathrm{m}} = \beta_1 \cdot \boldsymbol{q}_{t-1}^{\mathrm{m}} + (1 - \beta_1) \cdot \boldsymbol{g}_t \tag{22}$$

$$\boldsymbol{q}_t^{\mathrm{v}} = \beta_2 \cdot \boldsymbol{q}_{t-1}^{\mathrm{v}} + (1 - \beta_2) \cdot \boldsymbol{g}_t^2. \tag{23}$$

Additionally, a correction is applied to reduce the bias introduced by the cold-start of the exponential moving average calculations as

$$\widehat{\boldsymbol{q}_t^{\mathrm{m}}} = \boldsymbol{q}_t^{\mathrm{m}} / \left(1 - \beta_1^t\right) \tag{24}$$

$$\widehat{\boldsymbol{q}_t^{\mathrm{v}}} = \boldsymbol{q}_t^{\mathrm{v}} / \left(1 - \beta_2^t\right) \tag{25}$$

and the final update operation is

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \cdot \frac{\widehat{\boldsymbol{q}_t^{\mathrm{m}}}}{\sqrt{\widehat{\boldsymbol{q}_t^{\mathrm{v}}}} + \epsilon} \tag{26}$$

where the ϵ is a small constant used to improve the numerical stability.

E. Learning Theory Considerations

In learning theory, the training of a learning model should follow the principles of empirical risk minimization (ERM) and structural risk minimization [40]. In brief, the risk function $\mathcal{R}(\boldsymbol{\theta})$ of a given learning model $\mathcal{F}(\boldsymbol{x}, \boldsymbol{\theta})$ can be written as follows:

$$\mathcal{R}(\boldsymbol{\theta}) = \int L(\boldsymbol{y}, \mathcal{F}(\boldsymbol{x}, \boldsymbol{\theta})) \mathrm{d}\mathcal{P}(\boldsymbol{x}, \boldsymbol{y}). \tag{27}$$

The joint probability distribution $\mathcal{P}(\boldsymbol{x}, \boldsymbol{y}) = \mathcal{P}(\boldsymbol{y}|\boldsymbol{x})\mathcal{P}(\boldsymbol{x})$ is unknown and we have to train the learning model using m samples. The empirical risk functional is then defined as follows:

$$\mathcal{E}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{y}^i, \mathcal{F}(\boldsymbol{x}^i, \boldsymbol{\theta})). \tag{28}$$

The principle of ERM means that we adjust the parameter $\boldsymbol{\theta}$ to minimize the empirical risk $\mathcal{E}(\boldsymbol{\theta})$ during the training process. The Vapnik–Chervonenkis (VC) dimension is introduced to examine the condition how well the empirical risk $\mathcal{E}(\boldsymbol{\theta})$ converges to risk $\mathcal{R}(\boldsymbol{\theta})$. The VC dimension h of a model $\mathcal{F}(\boldsymbol{x}, \boldsymbol{\theta})$ is a measure of the model's capacity, or complexity.

There is a tradeoff, that as h increases, the minimum empirical risk decreases (the higher structural complexity, the better expressing power), and the confidence interval increases (the higher structural complexity, the more probable the model being overfitting). To limit the model's confidence interval, we have to constrain the model's structural complexity. In practice, this can be done by adding a L2 norm to the loss function as we have done in (19).

Other than adjusting toward a smaller L2 norm during the training phase, researchers can also choose to design a neural network structure suitable for the problem with fewer parameters. As mentioned previously, convolutional neural networks have shown successful applications in solving image classification problems [37]. The concept of convolutional neural networks is that they apply a convolution kernel with a restricted number of parameters to different regions of the input data. Traditional convolutional neural networks are useful only if the key features are closely resided in the two-dimensional input data, for instance, small patterns commonly seen in images. Unfortunately, it cannot be applied to solve the routing task because the input data, although represented in two-dimensional adjacency matrices, are actually graph-based topological data. To make a neural network model generalizable, the design of neural network structures has to be appropriate [41]. The proposed GADL shares the same inspiration of reducing parameters and reusing them in a graph-aware way, as stated in Section IV-C. The design of GADL's structure reduces the model's structural complexity and increases the confidence interval of a trained model.

TABLE III
EXPERIMENT SETUP

Platform Configuration				
Hardware	**Control Plane**	**Data Plane**	**Software**	**Specification**
CPU	2 x Intel Xeon E5-2660 v3 @ 2.60 GHz	Intel Xeon E7-4807 @ 1.87 GHz	Topology Management	Mininet
Memory	256 GB DDR4 @ 1866 Mhz	64GB DDR3 @ 1333 Mhz	Virtual Switch	OpenVSwitch
GPU	4 x NVIDIA K80	-	Controller	Customized POX
# of servers	1	6		

Topology Configuration			
	AT&T	Barabási-Albert	Mediation-driven attachment
Link capacity pattern	1 Gbps	Randomly selected from 0.5, 0.75, 1.0, 1.25, 1.5 Gbps	
Propagation Delay	Calculated from geographical distance	Randomly generated following $X \sim U(0.005, 0.1)$ second	
Traffic Pattern	Traffic appears between each two vertices, with randomly generated traffic intensity following $X \sim U(0, 1)$ Gbps		
# of topologies	1	4 for each category	
# of capacity patterns	1	4 for each category	
# of delay patterns	1	4 for each category	
# of traffic patterns	3600	100 for each category	
# of combinations	3600	6400 for each category	
# of training samples	357k	620k	585k
# of testing samples	343k	595k	561k

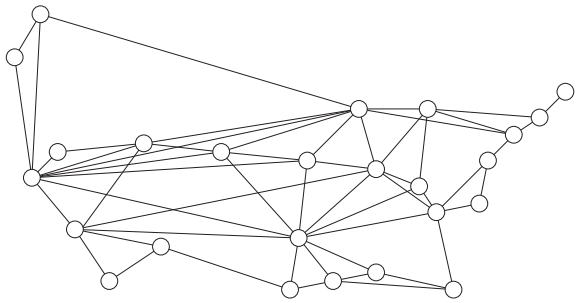

Fig. 5. AT&T backbone network topology used in experiments.

V. PERFORMANCE EVALUATION

A. Experiment Setup

The platform used for experiments is as listed in Table III. The Mininet simulation tool and the OpenVSwitch [42] software switch are used to simulate an SDN environment. Moreover, the data plane is mapped into six high-performance servers so that the computational power for simulation is sufficient. The OpenVSwitch software switch is customized to encode switch level measurements in packet headers. A customized controller is built based on POX to execute proposed routing architecture. The task is to find an optimal path from each node to another. In the experiments, a real-world topology is used as shown in Fig. 5, which has 25 nodes and 112 links. In addition, scale-free topologies generated from Barabási–Albert model and Mediation-driven attachment model are also used to produce more reliable results.

In the experiments, five models are tested. The DBA [16] is a state-of-the-art routing strategy and it is presented here as a baseline. The DNN uses raw network measurements as input and uses a DNN without convolution kernel. Deep neural network with features (DNNF) is similar to DNN but uses the features in Section IV-B as input. CNN [33] uses a traditional CNN. The convolutional neural network with features (CNNF) is similar to the CNN but uses the proposed features as an addition. The proposed GADL uses both the proposed features and the graph-aware neural networks in Section IV-C. In the experiments, GADL uses $k = 6$ for the graph-aware structure, and uses three layers of convolution, each of which is followed by a max-pooling layer of size 2. CNNF and CNN use three layers of convolution of size 2 without pooling. DNNF, DNN, and DBA use five layers of fully connected layers.

In the AT&T backbone topology, each link has 1 Gb/s bandwidth, the propagation delay is calculated from the geographical distance of the nodes, and 3600 traffic patterns randomly generated from uniform distribution U(0, 1) Gb/s. In the Barabási–Albert and mediation-driven attachment topologies, four topologies, four capacity patterns, four delay patterns, and 100 traffic patterns are generated for each category. The capacity is selected from 0.5, 0.75, 1.0, 1.25, and 1.5 Gb/s. The propagation delay is generated from uniform distribution U(0.005, 0.1) s. Traffic flows are generated and replayed based on CAIDA Anonymized Internet Traces [43]. A number of flows are selected and replayed in parallel from each source node to each destination node, so that the average traffic volume for each source–destination pair meets the traffic pattern configuration.

This experimentation setup is specifically chosen so that the results can reflect the performance of methods when applied in real-world scenarios. To start with, the topologies in-use are a combination of both real-world and synthetic topologies. Furthermore, the simulation data-plane is based on production-level virtual switch OpenVSwitch [42]. The traffic pattern of flows is based on CAIDA [43] dataset recorded and anonymized from real-world backbone traffic. In general, we have tried to make experiments and simulations closer to real-world networking in various aspects.

While an optimal is defined by either latency, throughput, or a combination of both, the training data are prepared targeting minimal latency paths in experiments. The reference algorithm used for preparation is from a recent work [34], which provides provable approximation guarantees.

TABLE IV
TIME USED TO TRAIN MODELS AND FINAL ACCURACY

Model	Accuracy	Training Time	Time Used[a]
DBA	0.6921	220.3 minutes	220.3 minutes
DNN[b]	0.0910	–	–
DNNF	0.7207	88.7 minutes	56.7 minutes
CNN	0.7592	70.1 minutes	10.1 minutes
CNNF	0.8069	65.3 minutes	4.1 minutes
GADL	0.8655	53.6 minutes	1.2 minutes

[a]This is the time used in training of each model reaching a same level of accuracy as DBA on training data.
[b]DNN with raw network measurement inputs shows no promising progress after 60 min of training. The best result is recorded here.

B. Evaluation Metrics

To start with, since supervised learning is used to train the models, we adopt the metrics of accuracy from supervised learning to evaluate the learning ability of each model. Accuracy is the fraction of correctly predicted samples versus all the samples being tested calculated as

$$\text{Accuracy} = \frac{\text{Count}\left(C_i^{\text{actual}} = C_i^{\text{predicted}}\right)}{Z}. \tag{29}$$

Let C_i^{actual} be the actual next forwarding node computed from the reference algorithm for sample i, $C_i^{\text{predicted}}$ be the predicted output from the neural network, and Z be the number of samples tested.

The accuracy ranges between $[0, 1]$. The closer the accuracy is to one, the better the model's learning ability is.

Moreover, the cross entropy is used to monitor the possibility of confusion among multiple available next forwarding nodes when each model is making decisions. The cross entropy is defined as

$$\text{CrossEntropy} = -\sum_i y_i \log p_i \tag{30}$$

where p_i is the probability which the model assumes that the vertex i should be chosen as the decision. The y_i is 1 if vertex i is the correct choice. Otherwise, the y_i is zero. The value of cross entropy is in the range of $[0, +\infty)$. The closer the cross entropy is to zero, the more confident the model is when making decisions.

At last, we are going to examine the average end-to-end flow latency in testbed with different routing policies. This shows how these learning-based models perform in real networking scenarios.

C. Result and Analysis

From Table IV, we can find that both DNNF, CNN, CNNF, and GADL outperform state-of-the-art DBA in terms of accuracy and training efficiency. All these three models use a centralized architecture in the network controller. While the deep neural structures used in these models are at least 500X more complex than DBA in terms of computational power required; however, these models train 3.8X–183.6X faster than DBA. This is because a centralized neural network fine-tunes its neuron weights for all pairs of routing tasks rather than training multiple sets of partially overfitted models separately. Moreover, a centralized architecture not only reduces the networking overhead of exchanging parameters, but also makes the model learn faster and better because there are low-level characteristics useful for different tasks and a logically centralized model makes learning these characteristics possible. This phenomenon is seen in other machine learning applications, such as shared neural networks [44] and shared gradient update [45].

Fig. 6 shows the proposed GADL performs well when an optimal path contains multiple hops, which is common in communication networks. In the experiment on AT&T topology, as shown in Fig. 6(a), optimal paths with length from 2 to 6 covers 83.43% cases. These two results demonstrate that GADL is smart enough to deduce a multihop optimal path in complex and swift shifting routing scenarios. The achievement of GADL benefits from both the designed features and graph-aware structure. More experiments show consistent results on topologies generated from the Barabási–Albert model and mediation-driven attachment model, as shown in Fig. 6(b) and (c).

The huge 63.97% accuracy boost from DNN to DNNF shows the strength of well-defined features proposed in Section IV-B. These features turn raw network measurements into meaningful indicators suitable for routing tasks. Consequently, the deep learning models are able to learn the network dynamics more easily. Without the feature processing procedure, a neural network can hardly tell whether an attribute represents a low-load link, a high-load link, a failed link, or a nonexistent link. The comparison between CNN and CNNF also shows similar characteristics.

In general, CNNF is superior to DNNF. However, due to the lack of knowledge of real neighbors in the graph, as mentioned in Section IV-C, CNNF fails to utilize the full potential brought by the proposed features. Figs. 7 and 8 demonstrate that the modified graph-aware neural network helps GADL to train much faster and better, compared to traditional deep learning techniques in CNNF. Although OSPF is not a learning model that needs a training procedure, we add a line in Fig. 8 representing the average calculated accuracy for OSPF, to provide more comprehensive comparison between the learning-based methods and traditional ones. The proposed features also help GADL to keep working under the circumstance of 5% link failure, as shown in Fig. 9.

Now, we analyze the reason why the proposed GADL routing strategy is top effective. Let c_{i} be input channels, c_{o} be output channels, and w be output width. For a network with n switches, the input width is n^2. In each layer of neural networks, the graph-aware structure in GADL requires $k^2 c_{\text{i}} c_{\text{o}}$ trainable variables versus traditional fully connected layer's $w n^2 c_{\text{i}} c_{\text{o}}$. According to [46], in communication networks, the scale of average length is almost surely $O(\log \log n)$. As a result, we need to train $O((\log \log n)^2)$ variables instead of $O(n^2)$. As a result of $O((\log \log n)^2) < O(n^2)$, we find the theoretical basis of the GADL's high efficiency.

In terms of the performance in a real networking scenario, Fig. 10 demonstrates that proposed GADL achieves its goal to

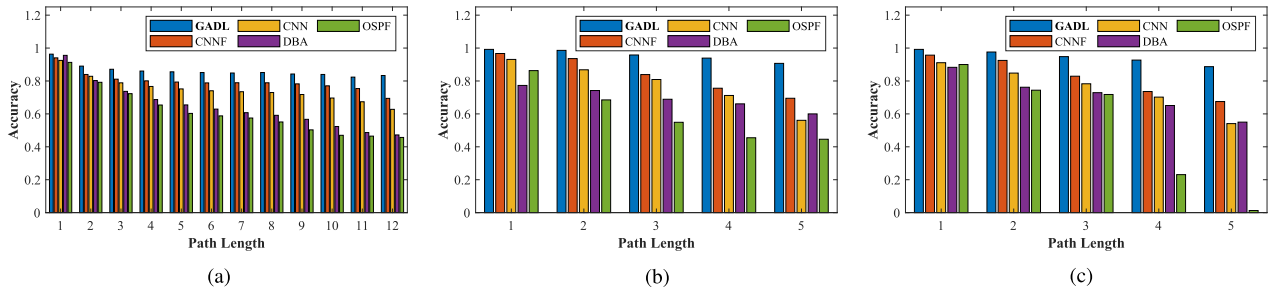

Fig. 6. Model testing performance. (a) AT&T topology. (b) Barabási–Albert topology. (c) Mediation-driven attachment topology.

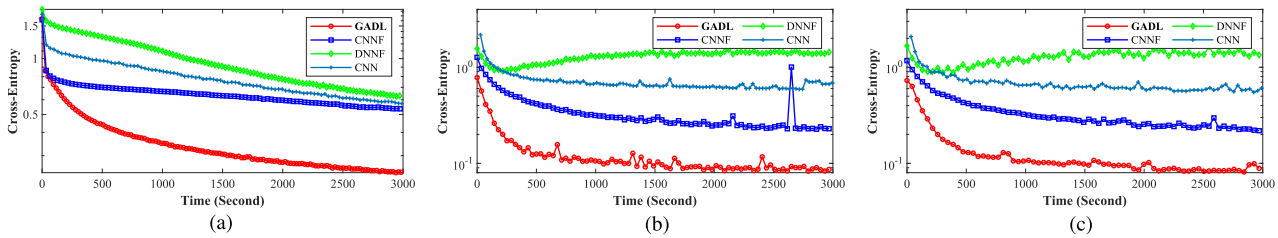

Fig. 7. Error by cross entropy of tested models as the training proceeds. (a) AT&T topology. (b) Barabási–Albert topology. (c) Mediation-driven attachment topology.

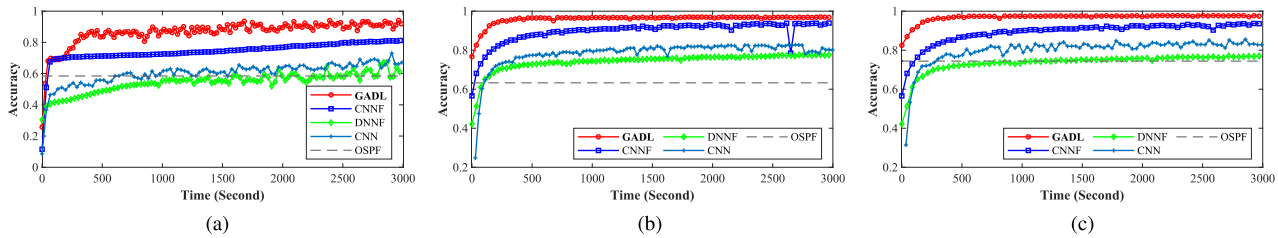

Fig. 8. Accuracy of tested models as the training proceeds. The average calculated accuracy for OSPF is also provided. (a) AT&T topology. (b) Barabási–Albert topology. (c) Mediation-driven attachment topology.

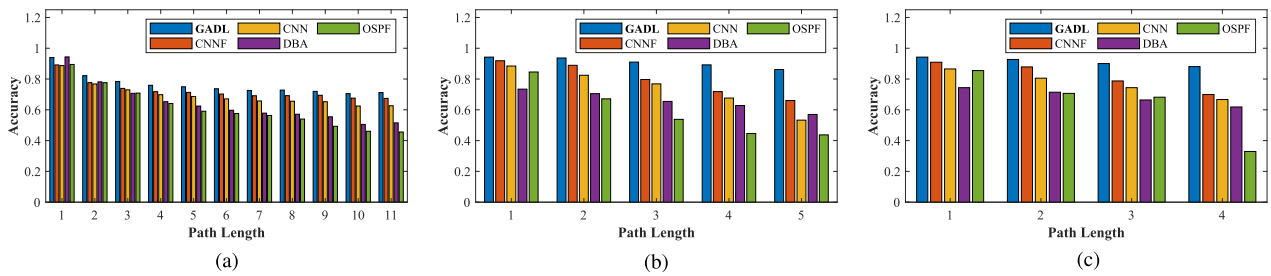

Fig. 9. Model performance under 5% Link Failure. (a) AT&T topology. (b) Barabási–Albert topology. (c) Mediation-driven attachment topology.

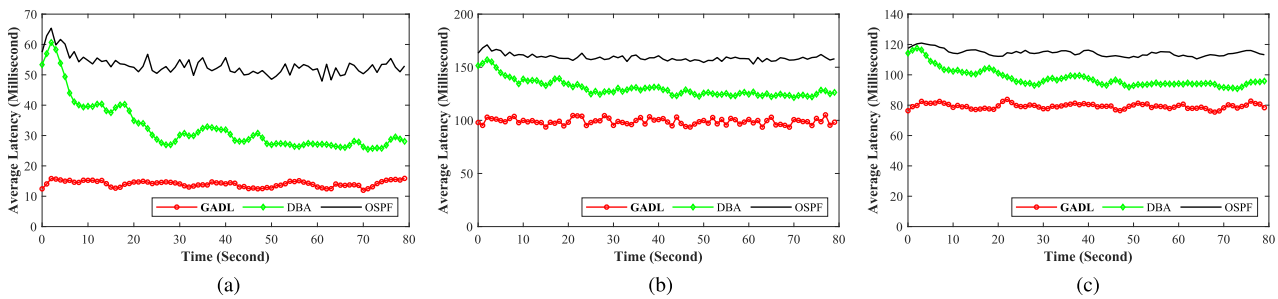

Fig. 10. Average network latency. (a) AT&T topology. (b) Barabási–Albert topology. (c) Mediation-driven attachment topology.

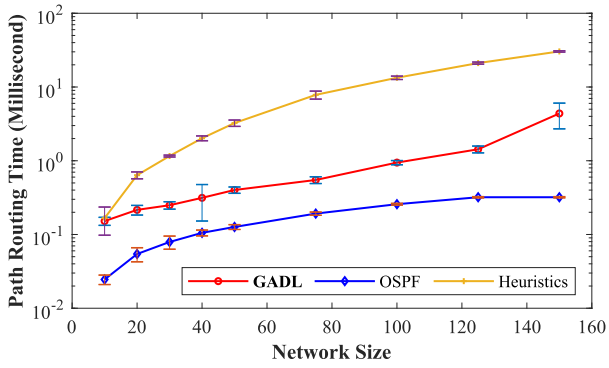

Fig. 11. Model's execution time for networks with different sizes. The error bar shows the 95% confidence interval.

find optimal paths with minimal latency in complex and variant networking scenes. In the experiments, GADL reduces average network latency by about 50% and converges faster against both OSPF and state-of-the-art DBA.

Finally, we further tested the per-path planning time under different network sizes, because the execution time of the model is critical to its application in the real world. The topologies are generated using a Barabási–Albert model, a mediation-driven attachment model, as well as a hierarchical network model, which grows according to the patterns from AT&T topology fractions. Fig. 11 shows that the execution time of GADL is consistently smaller than the reference algorithm under different network sizes. This means the GADL scales well and saves time when producing better routing decisions. It also shows that while providing better routing decisions, the cost compared to OSPF is acceptable.

In addition, we want to discuss how the topology setup affects the models' performance. As illustrated previously in Fig. 6, the traditional models' accuracies decease as the optimal path length increases. This phenomenon still applies to the proposed GADL model, but the gap of accuracies between a longer path and a shorter path is much smaller. The path length varies between different topologies, and as the size of the network increases, the network's diameter and path length increases as well. So it is important that the model's decision accuracy persists against different types and sizes of topologies, which the proposed GADL provides.

In experiments, we train the models to learn the routing policies in the prepared dataset in a supervised way. Hence, the optimality of the learned model is bounded by the optimality of the methods used to prepare the dataset. The preparation can be achieved by offline brute-force search or history data analysis, which provides either optimal or near-optimal policies and usually takes a lot of time to do so. Also, the metric of accuracy used in the result evaluation can be considered as a measure of optimality with respect to the given label preparation method. As a result, we believe the results show that the proposed GADL achieves much better performance using less time at the cost of a small gap to the (near-)optimal policies, which cannot be applied to real network systems due to their heavy-burden of computation.

VI. Conclusion

The advances in software-defined networking and deep learning techniques bring the potential of more intelligent routing strategies. A practical solution, however, needs to use a specially customized deep learning architecture for routing. Following the thought, we propose the GADL model to improve the intelligence in network management. The experiments show that the proposed GADL outperforms state-of-the-art deep learning based routing strategies both in terms of accuracy and efficiency. Our extensive experiments also show that representative features and graph-aware structures are useful in deep learning based network routing. Furthermore, the centralized architecture in software-defined networking benefits more from the power of deep learning models. In general, the praxis of machine learning and artificial intelligence techniques in network route planning, probably as well as in many other aspects of network management, needs extra work customizing the techniques to better suit the scenarios themselves, so that the networking systems can be endued with more powerful intelligence.

References

[1] T. Chen, M. Matinmikko, X. Chen, X. Zhou, and P. Ahokangas, "Software defined mobile networks: Concept, survey, and research directions," *IEEE Commun. Mag.*, vol. 53, no. 11, pp. 126–133, Nov. 2015.

[2] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (SDN) and distributed denial of service (DDOS) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Commun. Surveys Tut.*, vol. 18, no. 1, pp. 602–622, First Quarter 2016.

[3] N. McKeown *et al.*, "Openflow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[4] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[5] W. Wang, W. He, and J. Su, "Boosting the benefits of hybrid SDN," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 2165–2170.

[6] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: A better netflow for data centers," in *Proc. 13th USENIX Symp. Netw. Syst. Des. Implementation*, 2016, pp. 311–324.

[7] A. Hakiri, A. Gokhale, P. Berthou, D. C. Schmidt, and T. Gayraud, "Software-defined networking: Challenges and research opportunities for future internet," *Comput. Netw.*, vol. 75, pp. 453–471, 2014.

[8] Y.-Y. Liu, J.-J. Slotine, and A.-L. Barabási, "Observability of complex systems," *Proc. Nat. Acad. Sci.*, vol. 110, no. 7, pp. 2460–2465, 2013.

[9] T. H. Summers, F. L. Cortesi, and J. Lygeros, "On submodularity and controllability in complex dynamical networks," *IEEE Trans. Control Netw. Syst.*, vol. 3, no. 1, pp. 91–101, Mar. 2016.

[10] M. M. Najafabadi, T. M. Khoshgoftaar, C. Kemp, N. Seliya, and R. Zuech, "Machine learning for detecting brute force attacks at the network level," in *Proc. IEEE Int. Conf. Bioinf. Bioeng.*, 2014, pp. 379–385.

[11] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, "Robust network traffic classification," *IEEE/ACM Trans. Netw.*, vol. 23, no. 4, pp. 1257–1270, Aug. 2015.

[12] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop Hot Topics Netw.*, 2016, pp. 50–56.

[13] A. V. Vasilakos, Y. Zhang, and T. Spyropoulos, *Delay Tolerant Networks: Protocols and Applications*. Boca Raton, FL, USA: CRC Press, 2016.

[14] M. Z. Shafiq, J. Erman, L. Ji, A. X. Liu, J. Pang, and J. Wang, "Understanding the impact of network dynamics on mobile video user engagement," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 1, 2014, pp. 367–379.

[15] A. H. Sayed, "Adaptive networks," *Proc. IEEE*, vol. 102, no. 4, pp. 460–497, Apr. 2014.

[16] B. Mao *et al.*, "Routing or computing? the paradigm shift towards intelligent computer network packet transmission based on deep learning," *IEEE Trans. Comput.*, vol. 66, no. 11, pp. 1946–1960, Nov. 2017.

[17] D. K. Duvenaud *et al.*, "Convolutional networks on graphs for learning molecular fingerprints," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2224–2232.
[18] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3844–3852.
[19] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2014–2023.
[20] J. Hu, Z. Wang, S. Liu, and H. Gao, "A variance-constrained approach to recursive state estimation for time-varying complex networks with missing measurements," *Automatica*, vol. 64, pp. 155–162, 2016.
[21] M. Goyal *et al.*, "Improving convergence speed and scalability in OSPF: A survey," *IEEE Commun. Surveys Tut.*, vol. 14, no. 2, pp. 443–463, Second Quarter 2012.
[22] X. Xiao, A. Hannan, B. Bailey, and L. M. Ni, "Traffic engineering with MPLS in the internet," *IEEE Netw.*, vol. 14, no. 2, pp. 28–33, Mar. 2000.
[23] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The segment routing architecture," in *Proc. IEEE Global Commun. Conf.*, 2015, pp. 1–6.
[24] S. Chatterjee and S. Das, "Ant colony optimization based enhanced dynamic source routing algorithm for mobile ad-hoc network," *Inf. Sci.*, vol. 295, pp. 67–90, 2015.
[25] M. Shen, Z.-H. Zhan, W.-N. Chen, Y.-J. Gong, J. Zhang, and Y. Li, "Bi-velocity discrete particle swarm optimization and its application to multicast routing problem in communication networks," *IEEE Trans. Ind. Electron.*, vol. 61, no. 12, pp. 7141–7151, Dec. 2014.
[26] J. Wang, Q. Qi, S. Qing, and J. Liao, "Elastic vehicular resource providing based on service function-group resource mapping of smart identify network," *IEEE Syst. J.*, vol. 12, no. 2, pp. 1897–1908, Jun. 2018.
[27] J. Wang, Q. Qi, J. Gong, and J. Liao, "Mitigating the oscillations between service routing and SDN traffic engineering," *IEEE Syst. J.*, vol. 12, no. 4, pp. 3426–3437, Dec. 2018.
[28] D. K. Sharma, S. K. Dhurandher, I. Woungang, R. K. Srivastava, A. Mohananey, and J. J. Rodrigues, "A machine learning-based protocol for efficient routing in opportunistic networks," *IEEE Syst. J.*, vol. 12, no. 3, pp. 2207–2213, Sep. 2016.
[29] Y. Lee, "Classification of node degree based on deep learning and routing method applied for virtual route assignment," *Ad Hoc Net.*, vol. 58, pp. 70–85, 2017.
[30] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route," in *Proc. 16th ACM Workshop Hot Topics Networks*, 2017, pp. 185–191.
[31] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proc. Conf. ACM Special Interest Group Data Commun*, 2018, pp. 191–205.
[32] Z. Xu *et al.*, "Experience-driven networking: A deep reinforcement learning based approach," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 1871–1879.
[33] F. Tang *et al.*, "On removing routing protocol from future wireless networks: A real-time deep learning approach for intelligent traffic control," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 154–160, Feb. 2018.
[34] M. Chiesa, G. Kindler, and M. Schapira, "Traffic engineering with equal-cost-multipath: An algorithmic perspective," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 779–792, Apr. 2017.
[35] Y.-l. Boureau *et al.*, "Sparse feature learning for deep belief networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2008, pp. 1185–1192.
[36] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, 2011, pp. 315–323.
[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[38] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1139–1147.
[39] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations*, San Diego, CA, USA, 2015.
[40] V. Vapnik, "Principles of risk minimization for learning theory," in *Proc. Adv. Neural Inf. Process. Syst.*, 1992, pp. 831–838.
[41] V. N. Vapnik, "An overview of statistical learning theory," *IEEE Trans. Neural Netw.*, vol. 10, no. 5, pp. 988–999, Sep. 1999.
[42] B. Pfaff *et al.*, "The design and implementation of open vswitch," in *Proc. 12th {USENIX} Symp. Netw. Syst. Des. Implementation*, 2015, pp. 117–130.
[43] The CAIDA UCSD Anonymized Internet Traces, Jan. 2016. [Online]. Available: http://www.caida.org/data/passive/passive_dataset.xml
[44] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
[45] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.
[46] F. Chung and L. Lu, "The average distance in a random graph with given expected degrees," *Internet Math.*, vol. 1, no. 1, pp. 91–113, 2004.

Zirui Zhuang (S'18) is currently working toward the Ph.D. degree with the Beijing University of Posts and Telecommunications, Beijing, China.

His research interests include network routing and management for next generation network infrastructures, using machine learning and artificial intelligence techniques.

Jingyu Wang (M'10) received the Ph.D. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2008.

He is currently a Professor with the Beijing University of Posts and Telecommunications. He has authored and coauthored more than 50 academic papers in refereed journals and conferences. His research interests span broad areas of networking and communications, including future internet and vehicular network, traffic engineering, software-defined networks, network virtualization, cloud computing, game theory, and optimization.

Qi Qi (M'10) received the Ph.D. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2010.

She is currently an Associate Professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. She has authored and coauthored more than 30 papers in international journal, and obtained two grants from the National Natural Science Foundations of China. Her research interests include edge computing, mobile cloud computing, Internet of Things, ubiquitous services, deep learning, and deep reinforcement learning.

Haifeng Sun (M'19) received the Ph.D. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2017.

He is currently a Lecturer with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His research interests span broad aspects of artificial intelligence, neuro-linguistic programming, big data analysis, object detection, deep learning, deep reinforcement learning, software-defined networking, and processing.

Jianxin Liao (M'10) received the Ph.D. degree from the University of Electronics Science and Technology of China, Chengdu, China, in 1996.

He is currently the Dean of the Network Intelligence Research Center and a Full Professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. He has authored and coauthored hundreds of research papers and several books. His main research interests include cloud computing, mobile intelligent network, service network intelligent, networking architectures and protocols, and multimedia communication.

Dr. Liao was the recipient of a number of prizes in China for his research achievements, which include the Premier's Award of Distinguished Young Scientists from National Natural Science Foundation of China in 2005, and the Specially-Invited Professor of the "Yangtze River Scholar Award Program" by the Ministry of Education in 2009.