Graph Neural Network-based Virtual Network Function Deployment Prediction

Hee-Gon Kim*, Suhyun Park*, Dongnyeong Heo‡,Stanislav Lange†,
Heeyoul Choi‡, Jae-Hyoung Yoo*, James Won-Ki Hong*

*Computer Science and Engineering, Pohang University of Science and Technology, Pohang, South Korea
{sinjint, sh.park11, jwkhong, jhyoo78}@postech.ac.kr

†Information Security and Communication Technology, Norwegian University of Science and Technology, Trondheim, Norway
stanislav.lange@ntnu.no

‡Handong Global University, Pohang, South Korea
{21931011, hchoi}@handong.edu

***Abstract*—Software-Defined Networking (SDN) and Network Function Virtualization (NFV) help reduce OPEX and CAPEX as well as increase network flexibility and agility. But at the same time, operators have to cope with the increased complexity of managing virtual networks and machines, which are more dynamic and heterogeneous than before. Since this complexity is paired with strict time requirements for making management decisions, traditional mechanisms that rely on, e.g., Integer Linear Programming (ILP) models are no longer feasible. Machine learning has emerged as a possible solution to address network management problems to get near-optimal solutions in a short time. In this paper, we propose a Graph Neural Network (GNN) based algorithm to manage Virtual Network Functions (VNFs). The proposed model solves the complex VNF management problem in a short time and gets near-optimal solutions.**

***Index Terms*—Virtual Network Function, Machine Learning, Graph Neural Network.**

I. Introduction

Software-Defined Networking (SDN) and Network Function Virtualization (NFV) enable more efficient network management by allowing administrators to manage the network centrally and dynamically. The SDN controller and NFV manager provide global views of the network and NFV environment, and operators can use those to manage the network and service orchestration. Also, they can prevent the over-provisioning of resources and provide high availability by scaling and optimizing Virtual Network Functions (VNFs). However, although SDN/NFV enable efficient network management, they do not provide the optimal management solutions we need.

Integer Linear Programming (ILP) is one of the optimization methods for network management. ILP aims to minimize a linear cost function and uses a set of linear equality and inequality constraints to get an optimal solution. However, finding the optimal solution based on ILP takes a relatively long time, making it unsuitable for real-time network management.

978-3-903176-31-7 ©2020 IFIP

Recently, Machine Learning (ML) is emerging as a new paradigm to solve various networking problems and to automate network management. ML provides models that automatically learn and improve from experiences without being explicitly programmed [1]. ML takes some time to learn, but takes little time after learning. Also, ML is more effective in learning wide and dynamically changing data than statistical methods [2]. However, while ML has these advantages, it is not easy to apply ML to network management.

In order to apply ML to network management, it is necessary to provide sufficient network data and corresponding optimal management data as label data. Of course, well-designed ML models are also needed. Currently, there are only dozens of data available for network management, and these data target a small range of network management. Also, most of the studies use simple ML models for network management, and these models cannot understand the network structures or topologies [3]. Thus, the current studies are limited to solving simple problems and they do not show enough merit of ML compared to the other ML research areas. In this paper, we generate dynamic and diverse network data, and represent the network states as label data. We proposed to use Graph Neural Network (GNN) for VNF management. Our model uses network data represented by a graph and learns the state embedding [4] of nodes. This model effectively learns the network structure and generates near-optimal solutions in a short time.

This paper extends our previous work [5] in several directions. Firstly, we predict optimal VNF instance number rather than changes of VNF instance number. Secondly, we improve the prediction performance by up to 95% by changing the structure of the model and using additional data. Thirdly, we consider a more realistic environment where some nodes and edges could fail. Fourthly, evaluations regarding the impact of data size and complexity of services provide more deep insights into the practicality and reproducibility of our methods.

II. Related Work

In this section, we introduce several studies that are relevant to our study. The orchestration of VNF is derived from [6]. It points out the execution time problem of ILP and proposes a heuristic method. The proposed heuristic method shows fast execution time, but it generates a sub-optimal solution. ML-based VNF management is presented in [7]–[10]. These papers target only simple problems such as optimal VNF instance number [7], [8] and VNF chaining [9], [10]. There is an approach to graph-based ML in [11]. The approach is a little similar to GNN based network management, but it focuses more on graph theory and introduces core concepts only. The GNN based VNF resource prediction is proposed in [12]. It predicts resource usage using GNN and scales resources sub-optimally. Besides, the proposed ML-based resource scaling method is validated by comparing it with the human manual scaling in terms of the latency measured in post-scaling. However, this study lacks to validate whether the result is optimal because the proposed method does not compare itself with the optimal scaling method. Although GNN can represent graph data well, the method is not adequate to predict future data without recurrent models. The models simply follow historical resource data rather than predict future data.

Our model uses GNN to better represent network information. Our proposed model produces more specific and more realistic VNF management policies rather than merely predicting the overall number of VNF instances or network resource usage. We predicted optimal VNF instance number for all network servers and VNF types.

III. Problem Definition

A. Physical Network

We represent the physical network as an undirected graph $G = (N, E)$, where N and E denotes the set of nodes and links. We classify nodes into the server $s \in N$ that can deploy VNFs and the switch that cannot deploy VNFs. Supposing $c_s \in \mathbb{R}^+$ is the number of CPU cores and V_s is the deployed VNFs on s, server data D has $D_s = (c_s, D_{V_s})$ for all s, where D_{V_s} is the deployed VNF data on server s.

The physical links E have L and C denoted by link data and connection data. For $(i, j) \in E$, the link has $L_{ij} \in L$ defined as (m_{ij}, b_{ij}, d_{ij}), where $m_{ij} \in \mathbb{R}^+$, $b_{ij} \in \mathbb{R}$ and $d_{ij} \in \mathbb{R}^+$ is the maximum bandwidth, available bandwidth and delay between node i and j. The link also has $C_{ij} \in C$, the indicator whether the link is between the nodes.

$$C_{ij} = \begin{cases} 1 & \text{if } i = j \text{ or there is a link between node } i \text{ and } j, \\ 0 & \text{otherwise.} \end{cases}$$

B. Virtual Network Function

Supposing $\mathcal{T}$ is the set of VNF types, each VNF has different VNF type $t \in \mathcal{T}$. The VNF type decides the number of the required CPU cores, processing capacity, processing delay, and deployment cost represented by $\mathcal{E}_t \in \mathbb{R}^+, \tau_t$ (in Mbps), δ_t (in ms), $\mathcal{F}_t \in \mathbb{R}$, respectively. We define deployed t type VNF as $\mathcal{V}_{st} \in V_s$. $\mathcal{V}_{st}$ has dedicated data D_{st}. We assume that $\mathcal{I}_{st} \in \mathbb{R}^+$ is the VNF instance number and τ_t (in Mbps) , κ_{st} (in Mbps) is the maximum capacity and used capacity. We represent the deployed VNF type data D_{st} as follows:

$$D_{st} = \begin{cases} (\mathcal{I}_{st}, \tau_t, \kappa_{st}) & \text{if } V_s \text{ has type } t, \\ 0 & \text{otherwise.} \end{cases}$$

Now, the deployed VNF data D_{V_s} can be expressed as $\bigcup_{t \in \mathcal{T}} D_{st}$.

C. Service Request

Let the network receives many different service requests by users. Ψ is the set of services, and a service is $\nu \in \Psi$ represented by $(w_\nu, u_\nu, d_\nu, \phi_\nu, p_\nu, \beta_\nu, \gamma_\nu)$. $w_\nu, u_\nu \in N$ are the ingress and egress switch, respectively. $d_\nu \in \mathbb{R}^+$ is the running time of the service and $p_\nu \in \mathbb{R}$ is the penalty cost of Service Level Agreement (SLA) violation. ϕ_ν is the service request type that represents ordered VNF sequence. β_ν (in Mbps) , γ_ν (in ms) are the bandwidth demand of the traffic and max latency of SLA violation, respectively.

D. VNF Management

The objective of VNF management is to reduce network OPEX while guaranteeing service requirements [6]. To achieve this objective, the optimal number of VNF instances should be deployed on optimal locations (servers) while considering several requirements, i.e., service constraints and the physical network. We regard this problem as a classification problem and get optimal number of VNF instances for all servers and VNF types.

IV. Graph Neural Network

A. Graph Neural Network

ML is a promising technique in many research fields such as natural language processing and computer vision. Most of these fields use Euclidean domain data, and Convolutional Neural Network (CNN) [13] and Feed Forward Neural Network (FNN) are usually used to learn the data. Recently, ML is used in chemistry and biology, but CNN and FNN cannot learn their data because their data is usually non-Euclidean graph data. This non-Euclidean graph data contains rich relational information between each pair of neighboring elements and represents many kinds of graph structure data, i.e., social networks, physical systems [14]. Thus, the graph data gets attention and motivates to derive GNN.

GNN is the generalized model of CNN. GNN uses graph data as input, but it shares many things with CNN. While CNN uses the local connection between data and shares weights to reduce the computational cost [15], GNN uses the connection in the graph and share weights [14]. GNN is also motivated by graph embedding [4] that learns to represent graph information about nodes, edges, and sub-graphs. However, graph embedding does not share weights and it has a generalization problem [16].

The objective of GNN [17] is to learn state embeddings [4] and obtain outputs. Let x and h denotes the input features and

hidden states. When $co[v]$ is the set of edges connected to v and $ne[v]$ is set of neighbors of node v, we can define state embedding and output using h_v and o_v as follows:

$$h_v = f(x_v, x_{co[v]}, h_{ne[v]}, x_{ne[v]})$$
$$o_v = g(h_v, x_v),$$

$x_v, x_{co[v]}, h_{ne[v]}, x_{ne[v]}$ is the features of v, features of edges connected to v, the states of neighborhood nodes of v, features of the neighborhood nodes of v. f is transition function and g is output function.

$\boldsymbol{H}, \boldsymbol{O}, \boldsymbol{X}, \boldsymbol{X_D}$ are stacking variable of all the states, outputs, graph features and node features, respectively. $\boldsymbol{F}, \boldsymbol{G}$ are global transition function and global output function. The $\boldsymbol{H}$ is the banach's fixed point [18] and uniquely defined with contraction map $\boldsymbol{F}$.

$$\boldsymbol{H} = \boldsymbol{F}(\boldsymbol{H}, \boldsymbol{X})$$
$$\boldsymbol{O} = \boldsymbol{G}(\boldsymbol{H}, \boldsymbol{X_D}),$$

The function f and g can be a neural network (e.g, FNN, CNN, RNN), and the model updates $\boldsymbol{H}$ until the learning process is finished. We define target information and output of node i as r_i and o_i, respectively. Then, the loss can be represented as follows :

$$loss = \sum_{i=1} (r_i - o_i)$$

B. GNN for VNF Management

Some studies try to use ML to manage VNFs. However, they treat the network data as numeric data rather than the graph data [7]–[9]. They just make one long table and put all network data into the table. However, this table data is just numeric values of network components and cannot fully represent network structure. The data does not have connectivity information among network components, and this deficiency is one of the factors of the performance bottlenecks. In this paper, we use GNN and treat network data as a graph. The graph has connectivity information [14] and it can provide the network structure data [14]. Compared to the other VNF management studies [7]–[9], our VNF management problem is much more difficult to solve. We decide the number of instances and locations for all nodes and all VNFs types instead of deriving the only optimal number of VNF instances. This is not an easy task and requires abundant network information and a useful learning model. However, we can get the solution by using GNN. Also, our model can make VNFs management decisions more specific and realistic.

C. Generalizability for VNF Management

Generalizability is one of the issues for AI [19]. Specifically, the generalizability in the AI field means the model's ability to adapt to a variety of data. For example, if the model can attain almost the same performance for different data which are not used for training, the model can be said to have the generalizability. To get the generalizability, the model is usually trained with various data. Through learning, the model gets to understand the difference and attains high performance for the different data. However, there is a problem in the network domain. When the specific network topology is given, a dataset can be prepared by monitoring the network for a long time. We also can generate various network data with as many combinations of service requests on the topology as possible. However, this model cannot be applied to the different networks because the model learned only the one network topology. It also means that the model cannot be applied when the target network topology is changed because of network failures. The solution to this problem might be to collect network data with as many different types of topology as possible. However, if the model cannot understand the characteristics of network topology, the model cannot guarantee performance. GNN uses graph data explicitly so that the model can factor in the network structures. This fact provides GNN with the advantage of generalizability for the network domain.

TABLE I
SERVICE CATALOG [7]

Service id	Type (ϕ)	Proportion
1	NAT - Firewall - IDS	0.2
2	NAT - Proxy	0.3
3	NAT - WANO	0.2
4	NAT - Firewall - WANO - IDS	0.3

V. DATA GENERATION

A. Service Request Generation

We generate service requests and request lists. Whenever a new service request is generated, the list is also generated. The request lists contain several service requests whose service time d is not expired yet. These lists suppose a realistic network situation where multiple services are provided simultaneously. Supposing ν_{new} is the newest generated request and μ is the request list id and $\hat{a}$ is an arrival time of request, we can represent the request list as follows:

$$\pi_{\mu+1} = \{\nu_{\text{new}}\} \cup \pi_\mu \cap \{x : d_{\nu_x} > \hat{a}_{\nu_{\text{new}}} - \hat{a}_{\nu_x}\}$$

We use internet2 traffic pattern for generating service requests. Fig 1 is the one week traffic pattern that generated from real networks [20]. We generate four week traffic pattern by repeating the traffic pattern of Fig 1. We make 10 requests per minutes according to proportion P of the request type ϕ in Table I. Also, we discard $10 * (1 -$ **normalized traffic volume** in Fig 1) requests per minutes to follow the traffic pattern. We suppose that the request lists can contain up to four requests. We define request complexity as $n(\pi)$ that is the number of requests contained in the request list. We define service time d to make the ratio of request complexity as $\pi|(n(\pi)=1):0.48$, $\pi|(n(\pi)=2):0.34$, $\pi|(n(\pi)=3):0.13$, $\pi|(n(\pi)=4):0.05$.

We create two service request datasets, request set A and B with different bandwidth ranges. The bandwidth values for

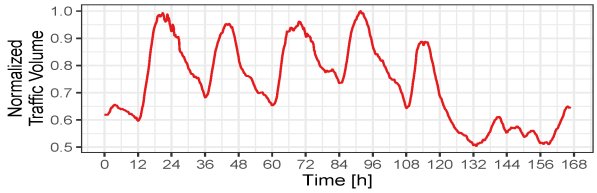

Fig. 1. Normalized traffic pattern of Internet2 network

the request set A are randomly set between 33 Mbps and 38 Mbps. The service requests of the request set B have a bandwidth between 300 Mbps and 390 Mbps. We use VNF catalog data from previous research [21], and Table II shows the specifications. In the case of the request set A, each node in the network requires up to one instance per a VNF type because the bandwidth values of the service requests are much less than the VNF processing capacities. In the case of the request set B, the nodes require up to three instances per a VNF type. Each request in both data has a different max latency for SLA, and the max latency values are randomly set as between 700ms and 750ms. We suppose all service requests have the same SLA penalty ratio, 1E-7.

B. Network Data Generation

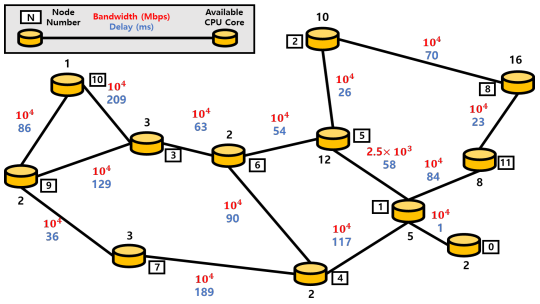

Fig. 2. Internet2 network topology

We use internet2 network topology in Fig 2. The topology consists of 12 nodes and 15 edges. We suppose all nodes are servers, and all nodes can be a source and destination of services. Each server and edge has information of available CPU cores, bandwidth, and delay.

We generate two datasets with different topology and name them as topology set A and B. Both are derived from inet2 topology. To validate the generalizability of GNN, we set the situation in which topology dataset A does not have any structural changes in topology, while dataset B has a structural change. In the best-case scenario, we should make the all different datasets covering every possible change in the topology. However, this scenario requires a too huge amount of data to handle, so we consider only a few local changes. In the dataset B, we assume that node 8 is not working during the time period between 36 and 48 hour time points (from total data of one week which is 168 hours) due to the node failure and the three links, (5, 6), (4, 7), and (9, 10) are disconnected between 72 and 108 hour time points.

Whenever a new request list is generated, we collect current network data (D, L, C). Then, we use ILP to get optimal VNF polices for the current network with the request list. When ILP finds the optimal VNF polices, we change the current network configuration to optimal network configuration. This process is repeated until all of the generated data is handled.

TABLE II
VNF CATALOG [21]

Network Function	CPU Required	Processing Capacity	Processing Delay
Firewall	4	900Mbps	45ms
Proxy	4	900Mbps	40ms
IDS*	8	600Mbps	1ms
NAT	1	900Mbps	10ms
WANO**	4	400Mbps	5ms

IDS* : Intrusion Detection System
WANO** : Wide Area Network Optimizer

C. ILP Calculation

We use specific ILP equations to get optimal VNF policy. This ILP solution reduces OPEX and gets optimal VNF instance numbers and locations [6]. It calculates the VNF deployment cost, energy cost, traffic forwarding cost, SLO violation cost and resource fragmentation cost. As we set all VNF deployment cost as zero, we only consider four costs.

VNF types

Servers	Firewall	Proxy	IDS	Nat	WANO
1	2	0	1	2	0
2	1	0	1	2	0
3	2	1	0	0	1
...	0	0	1	1	1
12	0	1	0	0	0

Fig. 3. Label data

D. Learning Dataset Generation

We generate the learning dataset for ML. Every time a new request arrives, the feature data are set to be generated from the network data (D, L, C) and service request (ν). At the same time, the label data are also generated by classifying the optimal number of VNF instances as Fig 3. The learning data consist of numeric data and categorical data together. We normalize all numeric data and apply one-hot encoding for categorical data. The network data need to be presented as a graph to keep the topology information. We convert D as matrix with the shape of **node number** $\times$ **node feature number**. C and L are also converted as **node number** $\times$ **node number** matrix.

VI. Learning Process

We implement an ML model using the TensorFlow backend. The learning process is shown in Fig 4. At first, we distribute each service request to a set of FNN layers and individually feed request data into the FNN layers. The purpose of the FNN

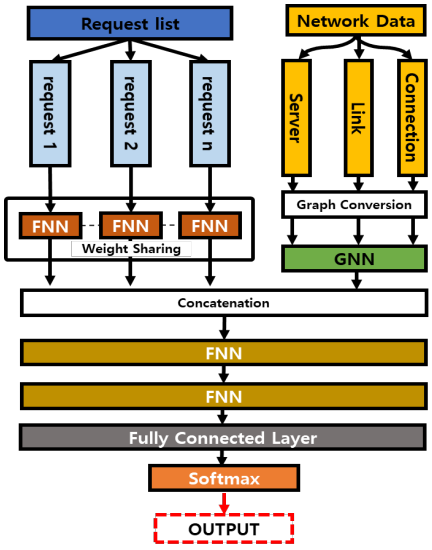

Fig. 4. GNN-based VNF management policy learning model

layers is to learn the request, so these FNN layers have the same role and share the same learning parameters. Simultaneously, the network data are represented as server data, link data, and connection data. Then, these data are converted as graph data and GNN uses the data as input. GNN learns the state embedding of the network nodes, and then we concatenate the output of GNN and output of FNN layers. We use two additional FNN layers for the model to learn the relation between service requests and the network topology. At last, we apply the fully connected layer and use the softmax layer to get class output. Because each dataset is used to classify 60 different VNF instances as shown in Fig 3, the fully connected layer has **60** $\times$ **maximum number of VNF instances** output. For each classification, we use cross-entropy as a sub-loss function. The objective loss function is the sum of the sub-loss function multiplied with class weight. The class weight is the reciprocal of each class ratio of the data. The optimal VNF instance numbers are usually imbalanced, and this imbalance can degrade the performance of learning. Thus, we multiply the sub-loss functions with class weights following the labeled class as below:

$$\textbf{loss function} = \sum_{\substack{s\in N\\ t\in \mathcal{T}}} (\sum_i A_i^{st})(\sum_i \frac{l_i^{st} log(o_i^{st})}{A_i^{st}})$$

(l_i^{st}, o_i^{st}, A_i^{st} : label, output, number of the class when optimal instance number is i and server is s and VNF type is t)

Table III show learning parameters. These parameters can be changed according to data size and request complexity. When the layer is relatively large compared to the number and complexity of the data, overfitting is likely to occur. Dropout and regularizer can be used to solve the problem. We divide the dataset into training data set and test set in a ratio of 4:1.

TABLE III
LEARNING PARAMETERS

Hyper-parameters	
Learning rate	0.01
Optimizer	Adam
Batch size	1024
Layer output size	
FNN (request)	30
GNN (request)	15
FNN1 (after concatenation)	200
FNN2 (after concatenation)	200

TABLE IV
IMPACT OF DATA SIZE AND REQUEST COMPLEXITY ON THE ACCURACY OF VNF DEPLOYMENT PREDICTION. THE RESULT IS REPRESENTED AS ACCURACY WITH (LOSS).

Lists	Weeks 1	Weeks 2	Weeks 4
$\pi\|(n(\pi) = 1)$	100% (0.01)		
$\pi\|(n(\pi) \leq 2)$	98.1% (8.3)	98.6% (4.4)	99.4% (2.4)
$\pi\|(n(\pi) \leq 3)$	95.6% (13.5)	96.2% (12.1)	97.2% (7.3)
$\pi\|(n(\pi) \leq 4)$	93.0% (18.7)	94.2% (15.3)	95.0% (11.5)
$\pi\|(n(\pi) = 2)$	92.7% (26.4)	96.4% (15.5)	98.3% (6.7)
$\pi\|(n(\pi) = 3)$	79.6% (50.3)	81.2% (46.6)	81.3% (45.1)
$\pi\|(n(\pi) = 4)$	72.2% (64.1)	70.1% (62.3)	74.2% (61.6)

VII. EXPERIMENT

A. Impact of data size and request complexity

We use request dataset A and topology dataset A to compare the performance of models according to the data size and request complexity. The accuracy is measured at the time when the loss of the test dataset is the lowest. Table IV shows the results. We find that the accuracy decreases as the request complexity increases. $\pi|(n(\pi) = 3)$ has small accuracy than $\pi|(n(\pi) = 2)$ and $\pi|(n(\pi) \leq 4)$ also has small accuracy than $\pi|(n(\pi) \leq 3)$. We find two reasons for this result. First, our data set is not balanced, and the number of requests is smaller as the request complexity is bigger. Second, the model should consider more cases when request complexity is big. It means that the model requires more data for high complexity. We also find that a less complex dataset can help learn the complex dataset. Through these experiments, we can confirm that the model training requires sufficient data for each type of dataset with different levels of request complexity.

B. Multi-class Learning

We compare the experimental results of the request dataset A and B. The dataset A possibly has up to one VNF instance for each VNF type on server, while the dataset B has up to two VNF instances. We define the class as the number of VNF instances of the target VNF type on the target server. For example, class 0 means zero VNF instances, class 1 is one instance, and class 2 is two instances. Fig 5 and Fig 6 show the recall values of each class for each VNF type and each server for the corresponding datasets. For the overall performance, models trained with the dataset A show better performance than models trained with the dataset B. That is because the classes

VNF types \ Servers	class0 class 1	1	2	3	4	5	6	7	8	9	10	11	12
	Firewall	90% 97%	90% 94%	91% 96%	94% 95%	90% 94%	90% 93%	88% 97%	96% 97%	98% 97%	97% 95%	100% 0/0	92% 94%
	Proxy	100% 0/0	90% 97%	91% 97%	100% 0/0	100% 0/0	92% 94%	100% 0/0	100% 0/0	97% 95%	100% 0/0	100% 0/0	96% 94%
	IDS	93% 96%	89% 96%	96% 96%	97% 96%	92% 94%	94% 90%	91% 92%	98% 95%	99% 95%	99% 97%	97% 96%	93% 96%
	Nat	97% 96%	94% 96%	98% 97%	94% 97%	94% 98%	94% 95%	94% 97%	97% 98%	98% 97%	97% 96%	100% 0/0	96% 97%
	WANO	90% 97%	88% 97%	93% 95%	91% 97%	90% 97%	90% 95%	90% 97%	95% 93%	98% 94%	96% 97%	100% 0/0	93% 96%

Fig. 5. Recall of VNF deployment prediction for servers and VNF types on the request dataset A and topology dataset A. Total accuracy is 95.0%. The optimal number of VNF instances is classified as a class and class 0 and class 1 have an average accuracy as 95.0%, 95.2%, respectively. 0/0 indicates that true positive + false positive is zero.

VNF types \ Servers	class 0 class 1 class 2	1	2	3	4	5	6	7	8	9	10	11	12
	Firewall	85% 94% 0/0	80% 91% 0/0	84% 93% 0/0	83% 94% 0/0	83% 89% 0/0	76% 92% 0/0	80% 92% 0/0	96% 92% 0/0	92% 94% 0/0	92% 90% 0/0	100% 0/0 0/0	83% 93% 0/0
	Proxy	100% 0/0 0/0	83% 91% 0/0	86% 71% 70%	100% 0/0 0/0	100% 0/0 0/0	81% 99% 0/0	100% 0/0 0/0	100% 0/0 0/0	89% 78% 0/0	100% 0/0 0/0	100% 0/0 0/0	82% 97% 0/0
	IDS	86% 93% 0%	83% 67% 91%	90% 86% 79%	89% 90% 44%	89% 81% 76%	85% 72% 93%	79% 79% 89%	89% 95% 0%	89% 96% 15%	94% 93% 0%	90% 93% 0/0	85% 93% 50%
	Nat	94% 95% 0/0	91% 95% 0/0	97% 94% 0/0	87% 98% 0/0	87% 97% 0/0	87% 96% 0/0	90% 97% 0/0	92% 97% 0/0	96% 97% 0/0	93% 95% 0/0	100% 0/0 0/0	94% 94% 0/0
	WANO	91% 96% 0/0	83% 72% 89%	85% 89% 47%	88% 96% 0/0	88% 96% 0/0	83% 80% 95%	86% 97% 0/0	89% 97% 0/0	91% 91% 48%	91% 96% 0/0	100% 0/0 0/0	89% 79% 92%

Fig. 6. Recall of VNF deployment prediction for servers and VNF types on the request dataset B and topology dataset A. Total accuracy is 89.54%. The optimal number of VNF instances is classified as a class and class 1, class 2, and class 3 have an average accuracy as 90.1%, 87.1%, 91.88%, respectively. 0/0 indicates that true positive + false positive is zero.

of dataset B are over unbalanced. The gray box in the figures indicates whether classes are over unbalanced and positive of some class is zero. The dataset A has nine gray boxes but, dataset B has 44 gray boxes. This class unbalance makes the model more focus on the less class and decreases the relative learning importance for other classes. Actually, the number of class 3 in dataset B is merely 0.03% of the entire dataset and it really decreases the relative learning importance for class 1 and class 2 and it also sequentially decreases overall performance.

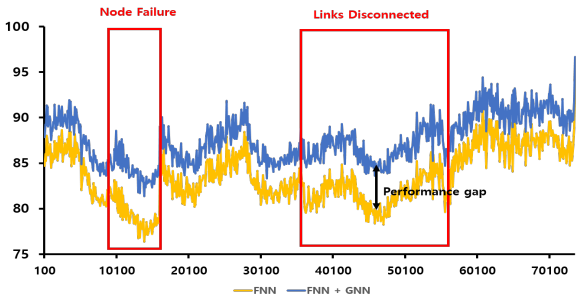

Fig. 7. Accuracy comparison on FNN and FNN-GNN

C. Topology-aware learning

We experiment for generalizability of GNN. We make the FNN model that uses flatten network data and service list data. We compare the accuracy for our proposed model (FNN-GNN) and the FNN model using request dataset B and topology dataset B. We use three weeks of data as training data and the remaining 1-week data as test data. Fig 7 shows the accuracy trend of models. The horizontal axis on the Fig 7 is the data generation number, and the vertical axis is the prediction accuracy of the data. We divide data into groups of 100 counts and evaluated the average of each group. The node failure occurs between data number 10000 and 17000. The link disconnections occur between data number 36000 and 56000. The GNN model shows higher performance at all times than the FNN model. Our model has 88% accuracy and each three class has 87.6%, 86.5%, 88.5% accuracy, respectively. FNN model has 83% accuracy and each three class has 83.5%, 83.4%, 89.3% accuracy, respectively. Also, both models tend to have low accuracy when the node is failed and the links are disconnected. However, the GNN model shows a more stable performance in the period when topology changes.

VIII. Conclusion

In this paper, we proposed a method of managing VNF using GNN. Our model uses network data as graph data and apply GNN to learns the state embedding of each node. We also learn service requests and concatenate the states and requests to find the optimal number of VNF instances. Our model solves complex VNF deployment problems. The model considers many network constraints that include limitations of physical network and requirements of service requests and VNFs. In the experiment, our model provides near-optimal VNF deployment prediction in a reliable time.

IX. Acknowledgment

This work was supported in part by Institute of Information & communications Technology Planning & Evaluation (IITP, Development of virtual network management technology based on artificial intelligence) under Grant 2018-0-00749.

REFERENCES

[1] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[2] Altman N. Krzywinski M. Bzdok, D. Statistics versus machine learning. *Nat Methods*, 15:233—-234, 2018.

[3] Raouf Boutaba, Mohammad A Salahuddin, et al. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):16, 2018.

[4] Hongyun Cai, Vincent Zheng, et al. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[5] Hee-Gon Kim, Suhyun Park, et al. Graph neural network-based virtual network function management. In *The 21th Asia-Pacific Network Operations and Management Symposium*, 2020.

[6] Faizul Bari, Shihabur Rahman Chowdhury, et al. Orchestrating virtualized network functions. *IEEE Transactions on Network and Service Management*, 13(4):725–739, 2016.

[7] Stanislav Lange, Hee-Gon Kim, et al. Predicting VNF Deployment Decisions under Dynamically Changing Network Conditions. In *International Conference on Network and Service Management*, 2019.

[8] Sabidur Rahman, Tanjila Ahmed, et al. Auto-scaling vnfs using machine learning to improve qos and reduce cost. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

[9] Jianing Pei, Peilin Hong, and Defang Li. Virtual network function selection and chaining based on deep learning in sdn and nfv-enabled networks. In *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE, 2018.

[10] R. Shi, J. Zhang, W. Chu, et al. Mdp and machine learning-based cost-optimization of dynamic resource allocation for network function virtualization. In *IEEE International Conference on Services Computing*, 2015.

[11] Wolfgang Kellerer, Patrick Kalmbach, et al. Adaptable and data-driven softwarized networks: Review, opportunities, and challenges. *Proceedings of the IEEE*, 107(4):711–731, 2019.

[12] R. Mijumbi, S. Hasija, et al. A connectionist approach to dynamic resource management for virtualised network functions. In *International Conference on Network and Service Management (CNSM)*, 2016.

[13] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series, the handbook of brain theory and neural networks, 1998.

[14] Z. Liu and J. Zhou. *Introduction to Graph Neural Networks*. 2020.

[15] Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

[16] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[17] Franco Scarselli, Marco Gori, et al. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[18] Mohamed A Khamsi and William A Kirk. *An introduction to metric spaces and fixed point theory*, volume 53. John Wiley & Sons, 2011.

[19] Kenji Kawaguchi, Leslie Pack Kaelbling, and Yoshua Bengio. Generalization in deep learning. *arXiv preprint arXiv:1710.05468*, 2017.

[20] Yin Zhang. Abilene traffic matrices, 2004.

[21] NM Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862–876, 2010.