Generalizable and Interpretable Deep Learning for Network Congestion Prediction

Konstantinos Poularakis[1], Qiaofeng Qin[1], Franck Le[2], Sastry Kompella[3], and Leandros Tassiulas[1]

[1]Department of Electrical Engineering and Institute for Network Science, Yale University, USA
[2]IBM Research, USA
[3]Naval Research Laboratory, USA

***Abstract*—While recent years have witnessed a steady trend of applying Deep Learning (DL) to networking systems, most of the underlying Deep Neural Networks (DNNs) suffer two major limitations. First, they fail to generalize to topologies unseen during training. This lack of generalizability hampers the ability of the DNNs to make good decisions every time the topology of the networking system changes. Second, existing DNNs commonly operate as "blackboxes" that are difficult to interpret by network operators, and hinder their deployment in practice. In this paper, we propose to rely on a recently developed family of graph-based DNNs to address the aforementioned limitations. More specifically, we focus on a network congestion prediction application and apply Graph Attention (GAT) models to make congestion predictions per link using the graph topology and time series of link loads as inputs. Evaluations on three real backbone networks demonstrate the benefits of our proposed approach in terms of prediction accuracy, generalizability, and interpretability.**

***Index terms*—** Deep Learning, Graph Attention Networks, Interpretability, Congestion Prediction.

I. Introduction

A. Motivation

While prevalent in various areas in computer science (e.g., computer vision, speech recognition, machine translation, etc), Deep Learning (DL) [1] has only recently attracted the interest of the networking community. DL uses Deep Neural Networks (DNNs) to find patterns in massive measurement data and fit complex models accordingly [2]. This capability can be exploited for the prediction of network events (e.g., traffic congestion) and the recommendation of appropriate network control policies (e.g., traffic routing over less-congested paths). In fact, recent work has demonstrated significant benefits of using DNNs for various networking applications including video streaming [3], cloud resource allocation [4], and traffic routing [5].

Despite the recent progress, the application of DL in networking faces unique challenges compared to other areas. More specifically, DNNs have been originally designed to learn models of data defined on Euclidean domains (e.g., images, text, and videos). For example, an image data can be represented as a regular grid in the Euclidean domain. However, for networking applications, *the topology of the network is often a critical component to be considered as an input to the DNN models*, but such those network topologies consist of a graph-based, and non-Euclidean, format. Indeed, it can be helpful for the DNN to know which links are adjacent, as the behavior of these links is typically correlated and could therefore be exploited to make more accurate predictions, e.g., about future traffic loads.

When applied to networking problems, most of the existing DNN architectures (e.g., feedforward, recurrent, convolutional) represent the network to a certain extent only through time series of traffic matrices [6] or link load vectors [7] whose format are in the Euclidean domain, just like images. Paradoxically (and ironically), while these architectures do not take the network topology as input, they end up building models that are specific and dependant to the topology (or topologies) seen during training. Therefore, if the topology changes (e.g., due to node or link failure, or addition) or if only part of the topology can be observed (due to malfunction of the network monitoring system) the network operator would have to *re-train a new model from scratch* with traffic matrices and link load vectors (of possibly fewer or more elements than before) corresponding to the new topology. Besides, in case the same operator manages multiple networks of different topologies (and possibly sizes), he would have to train a separate model for each network; *re-using the same model across the networks is not possible*[2].

Continuously re-training a DNN model every time the network topology changes is not computation-efficient and can disrupt network operations, especially in highly-dynamic settings (e.g., wireless and ad hoc networks, Internet of Things, etc.) where topology changes are the norm rather than the exception. Meanwhile, we cannot train in advance separate models for all possible topologies, numbers and permutations of links and nodes, due to their exponentially large number. *Ideally, we would like to train a single model that can be applied and generalized to topologies unseen during training.*

This research was supported by the U.S. Office of Naval Research under Grant N00173-21-1-G006, the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001, and the National Science Foundation in U.S. under Grant CNS 1815676.

978-1-6654-4131-5/21/$31.00 ©2021 IEEE

[2]Note that this issue does not exist for other non-networking applications like image classification where an operator can re-use the same DNN model (taking as input the image features and trained once) among multiple networks.

In addition, network operators are still skeptical and hesitant about deploying DNN-based systems in their production environments. The reason is that DNNs typically require a large number (e.g., on the order of millions) of parameters to achieve universal function approximation. This makes it difficult to *interpret* how a decision is made, and what the hypothesis behind the prediction outcome is. Consequently, network operators regard DNNs as "blackboxes" that are hard to explain and trust despite their promising performance benefits. Several attempts have been made recently towards interpreting the decision-making process of DNNs, with most of them focusing on image classification and language processing applications, and only a few on networking applications [8], [9].

From the above discussion, the following question naturally arises: *is it possible to design DNN-based networking systems that at the same time (i) exploit topological information from the network for better predictions, (ii) are generalizable to different topologies even unseen during training, and (iii) whose prediction outcome can be easily interpreted by network operators?*

The above question remains open, since the application of DNNs in networking is a relatively new area of research and their interpretability an even newer one, almost totally unexplored. To answer this question we need a DNN model that will be *by its design* aware of the network topology, yet robust to topology changes, and will offer the means to interpret its decision-making process. However, networking systems typically support a diverse set of applications that use different data structures to make their decisions (e.g., time-series of traffic matrices, link loads, buffer levels, routing paths, etc.). Therefore, coming up with a model that is suitable for all different applications is highly unlikely. In this paper we rather focus on a basic, yet fundamental, application of predicting whether links in a network would become congested, using as input data time series of link loads. We note that this is an important application that all network operators need to support in order to guide in time their traffic engineering decisions and prevent the network from being congested and the quality of service of their users being degraded.

B. Methodology and Contributions

To address the challenges mentioned above, we propose to rely on a recently developed family of graph-based DNNs that have attracted increasing interest within the research community (e.g., see the survey in [10]). These models can learn the relationships among graph elements (vertices and edges) and generalize over graphs unseen during training. Among the various models known within this family, we choose to build upon the Graph Attention Network (GAT) model [11]. An attractive property of this architecture is the attention mechanism which can be directly used for interpreting its decisions. More specifically, the attention weights can indicate the links that play an important role towards the prediction outcome.

We formally show how to apply the GAT model to predict congestion on the links in a network. This is a binary classification problem where the objective is to "minimize the error between the predicted class (congestion or not for each link) and the actual class" as a mean to train the network (i.e., learn the weights). More specifically, the objective is to "predict, given the traffic loads of the different links from time $t - M - 1$, to t, whether a link would be congested at time $t + 1$". In the literature, this problem has already been addressed by using DNN methods (e.g., see [6], [7] and the discussion of related work in Section V). However, to the best of our knowledge, this is the first time that a topology-aware, generalizable and interpretable DNN method is employed to perform this task. Evaluations using real data gathered from three real networks demonstrate the performance benefits of our proposed approach compared to several baselines, quantify the performance loss when generalizing over unseen topologies, and explain the decisions by commenting on the attention coefficient values and the importance of each link on the prediction outcome. Overall, the contribution of this paper can be summarized as follows:

- *Generalizable & Interpretable DL.* We leverage graph-based models to design DNN-based networking systems that at the same time exploit topological-information, generalize over different topologies, and make interpretable decisions.
- *Congestion Prediction.* We specifically apply the GAT model for predicting whether the links in a network would become congested or not, using as input data the graph topology and time series of link loads.
- *Evaluation on Real Data.* We implement our GAT approach on Tensorflow [12] and perform evaluations on three real backbone networks and using real link load measurements. We find that our approach: (i) achieves better prediction accuracy than three popular state-of-the-art DNN models (MLP, CNN and LSTM) as well as a traditional k-nearest neighbors learning algorithm, (ii) when trained on one network it can generalize on other networks of similar sizes with small loss in accuracy; however the generalizability has a limit and might not perform well for networks of much different sizes and structures, and (iii) by interpreting the attention coefficient values we can identify which links are important towards predicting congestion, and this way provide practical guidelines to network operators for deciding which subsets of links to monitor in their networks.

The rest of the paper is organized as follows. Section II briefly reviews background concepts, while Section III defines the congestion prediction problem and describes the proposed approach. Description of the evaluation settings and presentation of results is given in Section IV. Section V reviews our contribution compared to related works, while Section VI concludes the paper.

II. Background

In this section, we provide a brief overview of the key concepts in graph-based neural network models, and also those more specific to graph attention network models.

A. Graph Neural Networks

Graph Neural Networks (GNNs) are a general neural network architecture able to process graph structures as input data. The main idea is to learn hidden states of vertices in a graph based on the hidden states of their neighbor (adjacent) vertices. Note that by restricting the learning of hidden states between adjacent vertices only we essentially inject the graph structure (topology) into the process.

There exist many variants of GNNs in the literature. In this subsection, we briefly review the pioneer work in [13] which suggests that GNNs can learn the vertex representations (states) by adopting *recurrent* neural network architectures. They work in an iterative (recursive) manner where a vertex in a graph constantly exchanges messages with its neighbors for updating its hidden state until a stable equilibrium point is reached. We define below the notations required to understand these GNNs. A graph is represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertices $v \in \mathcal{V}$ and edges $e \in \mathcal{E}$. Edges can be represented as pairs of vertices $e = (v, u)$. $\Phi(v)$ denotes the set of neighboring vertices of v. The notations i_v and o_v represent the input and target values of vertex v, respectively. The operation of the GNN can be described by the following equations:

$$h_v^{(t)} = f\big(\{h_u^{(t-1)}\}|u \in \Phi(u)\big) \tag{1}$$

$$o_v = g\big(h_v^{(t\to\infty)}\big) \tag{2}$$

$$h_v^{(t=0)} = \text{init}(i_v) \tag{3}$$

where $h_v^{(t)}$ represents the hidden state of vertex v at time (iteration) t. $f(.)$ and $g(.)$ are parametric functions that can be implemented as feed-forward neural networks. The role of $f(.)$ is to update the hidden states while the role of $g(.)$ is to transform the final hidden state to the output value. Function init(.) initializes $h_v^{(t=0)}$ based on the input features, where zero padding is used to fit the dimensions if they are different for the input features and the hidden states. The training of the GNN, namely the parameters of $f(.)$ and $g(.)$, is possible via the Almeida-Pineda algorithm [14] which works by running the propagation of the hidden states to convergence, and then computing gradients based upon the converged solution.

An important observation here is that the functions $f(.)$ and $g(.)$ above do not depend on the topology of the graph. Therefore, once trained, the same model can be applied (without re-training) to different graphs of potentially different topologies (different neighborhood sets $\Phi(v)$). However, this type of GNNs do not provide any mechanism to interpret the decision-making process which was one of the main design goals we set in the introduction of this paper.

B. Graph Attention Networks

Graph Attention Networks (GATs) are a variant of GNNs that have been recently introduced [11]. Similar to the pioneer GNNs in [13], GATs exploit the graph structure (topology) during the learning process. However, GATs adopt a *convolutional* rather than a recurrent neural network architecture. That is, instead of recursively learning the values of hidden states of the vertices, GATs learn a set of weight values so that output features of vertices are equal to weighted sums of input features[3]. A novel aspect of GATs compared to other convolutional-based GNNs is the adoption of an *attention mechanism* that aims to learn relative weights between two neighboring vertices, i.e., the contributions of vertices' features to their neighboring vertices. Such mechanism is important in order to understand the behavior of the model and interpret how decisions are made, as we will elaborate and illustrate it further in the next sections.

We formally define a GAT model below. The model may consist of multiple graph attentional layers. We will describe below a single of these layers. The input to our layer is a set of vertex features $h_v \in \mathbb{R}^F$ for each vertex v in the input graph where F is the number of features in each vertex. The layer produces a new set of vertex features (of potentially different cardinality F'), $h'_v \in \mathbb{R}^{F'}$, as its output. To transform the input features into the output features two steps are required. In the first step, a shared linear transformation, parameterized by a weight matrix, $\mathbf{W} \in \mathbb{R}^{F' \times F}$, is applied to every vertex. In the second step, we perform a so-called self-attention on the vertices, i.e., a shared attentional mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'}$ that computes attention coefficients for each pair of vertices v, u:

$$e_{vu} = a(\mathbf{W}h_v, \mathbf{W}h_u) \tag{4}$$

The above coefficients indicate the importance of vertex $u's$ features to the features of vertex v. In its most general formulation, the model allows every vertex to attend on every other vertex, dropping all structural (topological) information [11]. We inject the graph structure into the mechanism by performing *masked attention*, i.e., we only compute e_{vu} for the neighbor vertices $u \in \Phi(v)$. To make coefficients easily comparable across different vertices, we normalize them across all choices of u using the softmax function, and compute the new coefficients:

$$\alpha_{vu} = \text{softmax}_u(e_{vu}) = \frac{\exp(e_{vu})}{\sum_{y\in\Phi(v)} \exp(e_{vy})} \tag{5}$$

The attention mechanism a in Equation (4) is a single-layer feedforward neural network, parametrized by a weight vector $\mathbf{a} \in R^{2F'}$, and applying the LeakyReLU activation function. Therefore, and by using Equation (5), we can expand the α_{vu} values as:

$$\alpha_{vu} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}h_v||\mathbf{W}h_u]))}{\sum_{y\in\Phi(v)} \exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}h_v||\mathbf{W}h_y]))} \tag{6}$$

where ^{T} represents transposition and $||$ is the concatenation operation.

Once obtained, the normalized attention coefficients are used to compute a linear combination of the features corre-

[3]Note the different terms used in different types of DNNs; *hidden states* for recurrent architectures and *input/output features* for convolutional architectures.

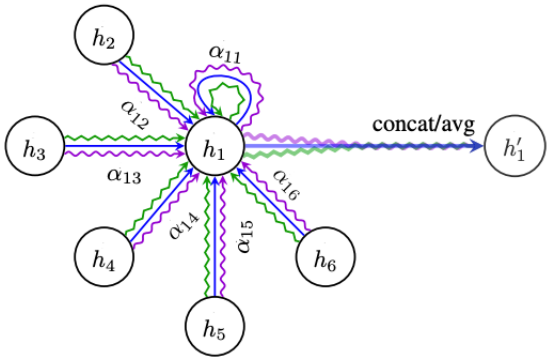


Figure 1. An illustration of multihead attention (with $K = 3$ heads) by node 1 on its neighborhood [11]. Three different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain h'_1.

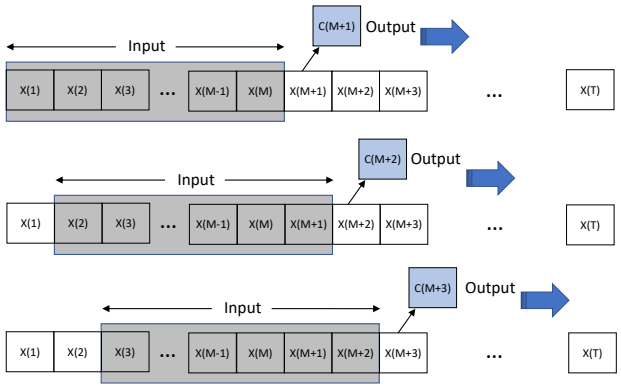


Figure 2. Sliding window for building training data set. Link loads in $M+1$ time slots are used to create a sample in the dataset; M for the input features and 1 for the output features.

sponding to them, to serve as the final output features for every vertex (after potentially applying a nonlinearity, σ):

$$h'_v = \sigma\left(\sum_{u\in\Phi(v)} \alpha_{vu}\mathbf{W}h_u\right) \tag{7}$$

The stability of the learning process can be improved by utilizing K heads for each vertex, therefore performing the above operation in Equation (7) separately K times with K weight and attention factors and then concatenate the K outputs to form an output in $\mathbb{R}^{K\times F'}$:

$$h'_v = ||_{k=1}^{K}\sigma\left(\sum_{u\in\Phi(v)} \alpha_{vu}^{k}\mathbf{W}^{k}h_u\right) \tag{8}$$

where α_{vu}^{k} are normalized attention coefficients computed by the k-th attention mechanism (a^k), and $\mathbf{W}^k$ is the corresponding input linear transformation's weight matrix. The aggregation process of a multi-head graph attentional layer is illustrated by Figure 1.

We note that once the model is trained and the values of the parameters **a** and **W** are learned, then we can apply the same model (without re-training) to a different graph (of possibly different topology and even different numbers of vertices and edges); *we just need to compute the value of α_{vu} for each pair of neighbor vertices v and u (as many pairs as they may be) in the new graph* by evaluating Equation (6). Then, we can simply pass the α_{vu} values we computed to evaluate Equation (7) (or Equation (8) if we choose to stabilize the learning process with multiple heads). These are simple computations that can be easily performed with low computational power. Therefore, the GAT model can be easily generalized over topologies unseen during training. Besides, our second design goal of interpretability is achieved thanks to the attention coefficient values α_{uv}, as we will explain in more detail in the evaluation Section IV.

III. Congestion Prediction

In this section, we show how to employ the GAT architecture for predicting congestion of links in a network based on historical time-series of link loads.

A. Network Model

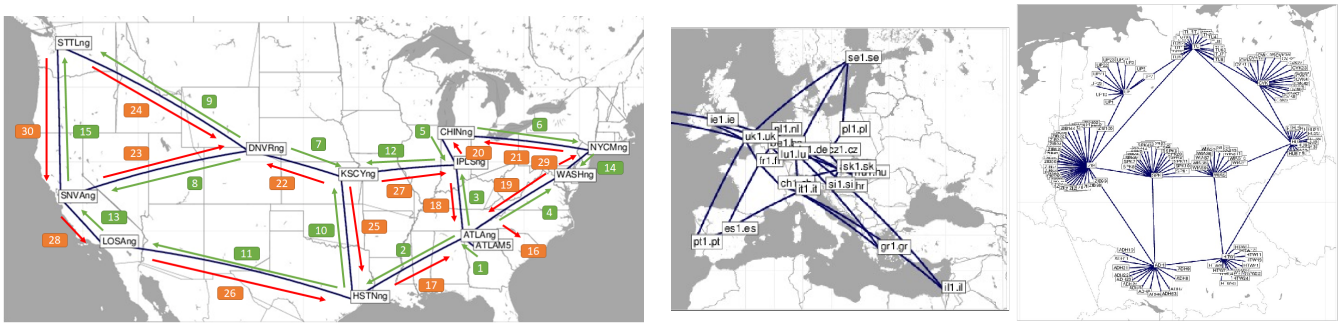
We consider a telecom network composed of set of nodes $\mathcal{N}$ (e.g., routers, switches, base stations) and links $\mathcal{L}$ (wireline or wireless) connecting them[4]. The links can be directed or undirected. Regardless, each link connects a pair of endpoint nodes. We say that two links are neighbors if they have a common endpoint node. Formally, we define by $\Phi'(l) \subseteq \mathcal{L}$ the neighbor links[5] of link l. The set $\Phi'(l)$ contains topological information that is important to know since often the behavior of neighbor links is correlated and can be used by the network operator to make more accurate predictions.

The traffic exchanged among the nodes in the network is routed according to a routing protocol such as OSPF or MPLS or any other of the standard protocols. The resulting traffic load measured on the network at time t can be represented as a vector $X(t) \in \mathbb{R}^{L\times 1}$, where $L = |\mathcal{L}|$ is the number of links in the network. We say that a link l is congested at time t when its load exceeds a given threshold τ_l, i.e., when $X_l(t) \geq \tau_l$. Here, τ_l can be set equal to a portion of the capacity of the link. We denote by $C_l(t)$ the binary variable indicating whether link l is congested at time t ($C_l(t) = 1$) or not ($C_l(t) = 0$), and by $C(t)$ the respective vector for all links. Such binary indicator is a typical way to handle congestion. For example, mechanisms such as Explicit Congestion Notification (ECN) [32] enable the switches or routers to modify specific bits in the packet header to indicate the occurrence of a congestion event.

It is critical for the network operator to monitor and if possible predict the congestion of the links so as to take appropriate actions in time, e.g., re-route traffic flows away from the congested links to balance the loads across the network.

[4]Note how we use the terms nodes and links to refer to the elements in the telecom network, while we used the terms vertices and edges for the elements in the abstract graph in Section II.

[5]We note that the definition of $\Phi'(l)$ looks similar to $\Phi(u)$ presented in Section II. This time, however, we refer to the neighborhood relationship among links in the telecom network rather than among vertices in the abstract graph $\mathcal{G}$.



(a) Abilene topology with 30 unidirectional links. Link 16-30 connect the same nodes as Link 1-15 while being in an opposite direction.

(b) GEANT (72 links) and BRAIN topology (332 links).

Figure 3. Different topologies for congestion prediction evaluations.

Given the sequence of the link loads $X(t-M-1),\ldots,X(t)$ measured in the past $M > 0$ time slots, the goal of the network operator is to predict the congestion vector in the next time slot, $C(t+1)$. The dataset of input/output features (link load sequences/congestion events) can be easily built by the network operator by processing the time series of link loads in a sliding window manner, as shown in Figure 2. For each sequence of link loads within the sliding window (input feature values), the operator can simply observe the link loads in the next time slot, right after the sliding window, to compute whether there is a congestion event at each link, and this way label the dataset with the appropriate output feature values.

We emphasize that the above objective is a binary classification task that can be addressed using common DL models (e.g., see the discussion of related work in Section V). In the next subsection, we show how we can address it using the GAT model in order to take advantage from the topological structure of the network, as well as the generalizability and interpretability properties of this architecture.

B. Graph Representation

The GAT model takes as input an abstract graph $\mathcal{G}$ with features on the vertices. To construct such a graph for our congestion prediction problem, we map each link $l \in \mathcal{L}$ in the telecom network to a vertex $v_l \in \mathcal{V}$ in the graph. The GAT model may consist of multiple graph attentional layers depending on the actual implementation (e.g., see the evaluation Section IV). We need to define the input and output features of the vertices separately for each of those layers. For the first layer, the input features h_{v_l} of vertex v_l are the sequences of traffic loads $X_l(t-M-1),\ldots,X_l(t)$ of the respective link l. For the last layer, the output features h'_{v_l} of vertex v_l are the congestion event binary values $C_l(t+1)$. For an intermediate layer, the input features are the output features of the previous layer, and the number of these features depend on the actual implementation. We also say that a vertex v_l is a neighbor to another vertex $v_{l'}$ (formally, $v_l \in \Phi(v_{l'})$) if the two corresponding links l, l' in the telecom network problem instance have a common endpoint node (formally, $l \in \Phi'(l')$). We then restrict to zero each attention coefficient value $\alpha_{v_l v_{l'}}$ (see Equation (6)) for which the vertices $v_l, v_{l'}$ corresponding to links l, l' are not neighbors.

IV. Evaluation

In this section, we evaluate the impact of the proposed approach using datasets of topologies and traffic matrices collected from real networks. Overall, we find that our approach predicts network congestion more accurately than state-of-the-art DL algorithms, while it has the extra benefit of generalizing to other networks than the one trained. The generalizability incurs a loss in accuracy that is small for networks of similar sizes but might be larger for networks of much different sizes and structures. The information revealed by the attention mechanism can be useful to network operators for deciding which subsets of links to monitor in their networks. In the rest of this section, we discuss these results in detail; we begin by describing the setup used in the later evaluations.

A. Setup

Datasets. Throughout our evaluations, we consider the topology and dynamic traffic traces of three real backbone networks captured in the SNDlib project [15]. The first network, named Abilene, contains 12 nodes and 30 unidirectional (i.e., 15 bidirectional) links, as depicted in Figure 3(a). The traffic demands between each pair of source and destination nodes are recorded dynamically as matrices for a time period of 6 months. The dataset also provides with link weights for calculating the shortest routing paths. Therefore, we are able to get the total traffic volume on each link. Similarly, we also consider two more networks, GEANT (22 nodes and 72 links) and BRAIN (161 nodes and 332 links) depicted in Figure 3(b). Although they vary in the size and topology, we will show that our approach is capable to generalize for different network graphs. The time granularity of the three traces are 5 min, 15 min, 1 hour respectively. To make meaningful comparisons, we aggregating the traffic matrices of the first two traces into 1-hour time slots as well in most evaluations. Then, we choose the loads of past 10 time slots as the input features to base our prediction ($M = 10$). Each dataset is split into training, validation and testing sets at the ratio $7:1:2$.

Metrics. To define congestion events, which are not specified in the original dataset, we calculate the average load of each link over all time slots as a constant value, and then multiply it by a *threshold factor* denoted by β. If the load of a link l exceeds this threshold in a specific time slot t, we regard this link as congested, i.e., $C_l(t) = 1$. In this way, we can emulate different levels of link capacity constraints by adjusting the β value. The same approach was used in [7].

One common metric to measure the prediction ability is the *accuracy*. However, the dataset is not balanced, where congestion events usually account for less than 10% among all records. Therefore, we also report the *precision* and *recall* rate metrics. Besides, we measure the F_1 *score*, which is the harmonic mean of the precision and recall, as well as the *Area under the Precision-Recall Curve (AUC)* metric [16].

Algorithms. We deploy a GAT model with three attention layers. In each layer, there are $K = 8$ attention heads with an Exponential Linear Unit (ELU) activation function. The number of output features per head (F') are 8, 6 and 4 for the first, second and third layer, respectively. We adopt a similar approach as in [17], concatenating the outputs of all heads together with the original input features of this layer. Finally, we add an output layer with sigmoid activation function to generate a binary prediction for each link.

We train the neural network with the Adam optimizer [18]. During the training process, we monitor the AUC metric of the validation set and stop training when the AUC cannot be further improved. The network architecture as well as the estimation of AUC are implemented using TensorFlow [12] and Spektral [19].

In order to show the advantages of our method, we implement several state-of-the-art algorithms to compare with, including the Multi-Layer Perceptron (MLP), which only contains fully-connected neural network layers, and the Convolutional Neural Network (CNN). We also consider a recurrent architecture, the Long Short-Term Memory (LSTM), which is adopted in [6] for traffic prediction. To ensure a fair comparison, we deploy the above state-of-the-art algorithms with similar depth (three layers) and amount of trainable parameters as our algorithm. We also consider more traditional methods without the usage of neural networks. We implement the k-nearest neighbors algorithm (KNN), which is used for network traffic analysis [31] by comparing the input vector with the most similar samples in the training set. *The details of the implementation of all algorithms can be found online in our publicly available code in [30], which ensures the reproducibility of all the evaluation results presented in this paper.* In the next three subsections, we present the evaluation results on performance, generalizability and interpretability, respectively.

B. Performance

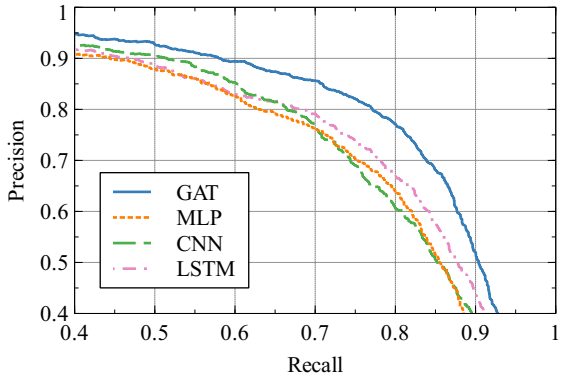
We evaluate the performance of the above algorithms on the Abilene dataset for different threshold factor values from $\beta = 1.5$ to $\beta = 3.0$. The results are summarized in Table I. We observe that *in all cases GAT achieves the highest prediction accuracy compared with the MLP, CNN and LSTM methods.* Although KNN reaches slightly better accuracy when $\beta = 2.5$ and 3.0, in such cases, the larger threshold factor value β implies a scenario where there is less limitation on the link capacity and so congestion happens less frequently. In other words, the dataset is largely biased, where the overall accuracy becomes less important. In addition to it, we should also prevent the algorithm from raising *false alarms*. Therefore, we also investigate the ability of the algorithms to achieve a trade-off between precision and recall. By setting different thresholds to the output layer, we plot the precision-recall curves in Figure 4. We observe that *for the same precision, our GAT algorithm can always achieve higher recall rates than other methods, and vice versa.* This is also indicated by the F_1 score and AUC, where GAT has the best performance for all β values in Table I. We note that the KNN algorithm outputs binary values rather than probabilities and therefore is not capable of realizing the trade-off between precision and recall. For this reason, precision-recall curves and AUC calculation are not applicable and are not reported in Table I and Figure 4.

β	Method	Accuracy	Precision	Recall	F_1 Score	AUC
1.5	GAT	**0.9605**	0.8246	0.7441	**0.7822**	**0.8354**
	MLP	0.9483	0.7254	0.7350	0.7302	0.7590
	CNN	0.9522	0.7874	0.6818	0.7308	0.7952
	LSTM	0.9538	0.7991	0.6879	0.7393	0.7937
	KNN	0.9500	0.8118	0.6176	0.7015	N/A
2.0	GAT	**0.9763**	0.8060	0.6463	**0.7173**	**0.7544**
	MLP	0.9711	0.7042	0.6494	0.6757	0.6564
	CNN	0.9722	0.7676	0.5757	0.6580	0.7032
	LSTM	0.9709	0.6844	0.6929	0.6887	0.7160
	KNN	0.9751	0.8237	0.5913	0.6884	N/A
2.5	GAT	0.9819	0.6649	0.6673	**0.6661**	**0.6802**
	MLP	0.9801	0.6511	0.5712	0.6085	0.5426
	CNN	0.9816	0.7217	0.5214	0.6054	0.6120
	LSTM	0.9812	0.6982	0.5391	0.6084	0.6320
	KNN	0.9833	0.7851	0.5267	0.6305	N/A
3.0	GAT	0.9876	0.7238	0.5729	**0.6396**	**0.6581**
	MLP	0.9853	0.6332	0.5553	0.5917	0.4933
	CNN	0.9870	0.7529	0.4824	0.5881	0.5806
	LSTM	0.9854	0.6361	0.5578	0.5944	0.5563
	KNN	0.9883	0.8112	0.5075	0.6244	N/A

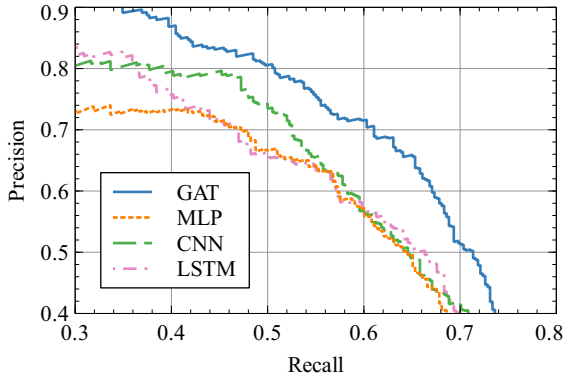
Table I
COMPARISON OF ALGORITHMS PERFORMING CONGESTION PREDICTION WITH DIFFERENT THRESHOLD FACTORS IN THE ABILENE NETWORK.

It is worth noting that the performance values of the CNN and LSTM algorithms we report in Table I are much better than those for the same algorithms reported in [7] which used the same dataset (Abilene) and the same setup with our work in its evaluation, implying that our implementation of these algorithms (number of layers and parameters) is more efficient (and therefore the comparison with our algorithm is more fair). Besides, [7] proposed an algorithm named DCRNN (Diffusion Convolutional Recurrent Neural Network) which for the same setup was shown to achieve worse performance than our GAT algorithm (0.9667 accuracy and 0.5414 F_1 score for $\beta = 3.0$).

Although we focus on generating binary congestion indicators, our method can be easily extended to other types of predictions, such as the multiclass classification or even

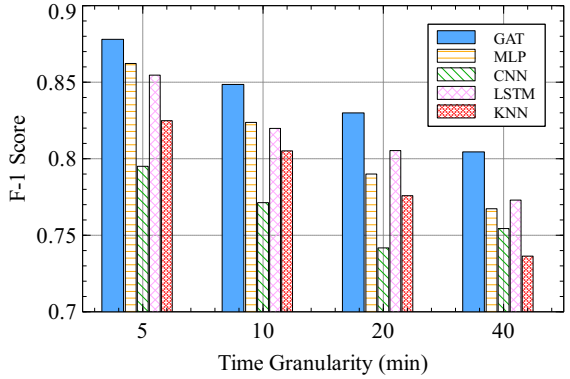


(a) $\beta = 1.5$.

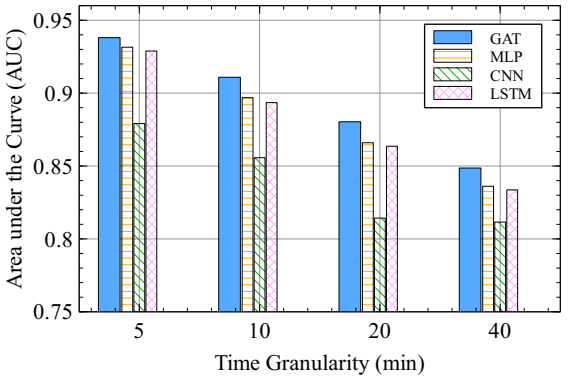


(b) $\beta = 3.0$.

Figure 4. The precision-recall trade-off curves for different threshold factor values in the Abilene network.



(a) Impact of time granularity on F_1 scores.



(b) Impact of time granularity on AUC values.

Figure 5. Evaluation results with different time granularity in the Abilene network ($\beta = 1.5$).

Method	GAT	MLP	CNN	LSTM
RMSE	**0.7059**	0.8226	0.7518	0.7655
MAE	**0.2601**	0.2797	0.3107	0.2779

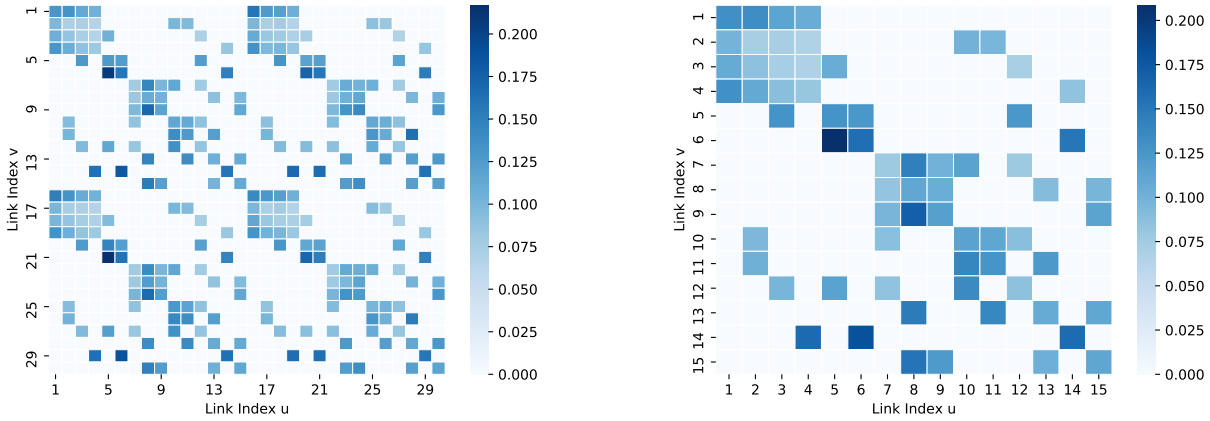
Table II
COMPARISON OF ALGORITHMS PERFORMING REGRESSION IN THE ABILENE NETWORK.

regression, where the algorithm outputs specific traffic load values of the future time slots. For example. in Table II we deploy the same types of neural networks but train them with the output $C_l(t+1)$ replaced by $X_l(t+1)$. Similar to the classification approach all values are divided by the average load of link l. Our goal is to minimize the root mean squared error (RMSE) between the predicted value and the ground truth. The results indicate that our GAT method can reach a much smaller error than the other algorithms. At the same time, the performance is also superior on the mean absolute error (MAE), which is another common measure of predictions.

So far, we focus on per-hour traffic matrices of the Abilene network to facilitate the comparison with other network topologies, which will be presented in the next subsection. However, we emphasize that our method can be applied to make predictions with different time scales. In Figure 5, we process the same dataset into traffic matrices of different granularity, from 5 min to 40 min, and measure the F_1 scores and AUC values again. The results demonstrate that errors of short-term predictions are smaller than the long-term ones, while the GAT has better performance than other algorithms in general.

C. Generalizability across Different Topologies

GAT is known for its *inductive learning* ability (also known as generalizability). In other words, the prediction can be performed in graphs that never appeared in the training set. This is usually not true for traditional neural network architectures, including the MLP, CNN and LSTM. Indeed, when a new graph with a different number of vertices is the input data, it requires a weight matrix of a different size in each MLP, CNN and LSTM layer. Even if the new graph happens to have the same size with the original graph that training took place, we cannot expect good performance because all the neuron weights are bound to specific vertices in the training set, rather than the new graph. On the other hand, the GAT model alleviates this issue since it concerns only with the feature values of each pair of vertices when calculating the attention coefficients, rather than their actual indices.



(a) Full topology.

(b) Partial topology with links in one direction.

Figure 6. The heat map of average attention coefficients in the Abilene topology. (a) All links are displayed. (b) We zoom-in on the links in one direction marked in Figure 3(a).

Test Set / Training Set	Abilene	BRAIN	GEANT
Abilene	0.9605	0.7803	0.9505
BRAIN	0.9182	0.9402	0.9023
GEANT	0.9406	0.9168	0.9584

Table III
ACCURACY WHEN PERFORMING TRAINING AND TESTING ACROSS DIFFERENT GRAPHS.

Test Set / Training Set	Abilene	BRAIN	GEANT
Abilene	0.7822	0.4422	0.7963
BRAIN	0.2605	0.7462	0.3966
GEANT	0.6446	0.6484	0.8243

Table IV
F_1 SCORES WHEN PERFORMING TRAINING AND TESTING ACROSS DIFFERENT GRAPHS.

Test Set / Training Set	Abilene	BRAIN	GEANT
Abilene	0.8354	0.2861	0.7351
BRAIN	0.3886	0.8130	0.5218
GEANT	0.7169	0.5980	0.8844

Table V
AUC WHEN PERFORMING TRAINING AND TESTING ACROSS DIFFERENT GRAPHS.

In order to verify the generalizability of congestion prediction across different topologies by our GAT algorithm, we apply the same training process as in the previous subsection to additional network traffic datasets, BRAIN and GEANT. In each experiment, we train the GAT model with one of these three datasets, and test it with another dataset. We measure the accuracy, F_1 score and AUC performance metrics for all nine combinations. The results are presented in Table III, IV and V, respectively. *Compared with the diagonal elements in these tables for which the training and test sets are identical, the accuracy of our GAT algorithm remains at a high level, with small loss in most cases.* The results of F_1 score and AUC are similar, except the interaction between Abilene and BRAIN where the performance drops more notably. This may be attributed to the fact that the BRAIN network contains a much larger number of links than Abilene (332 versus 30 links), and has quite different distribution of node degrees, further challenging the generalization capability of GAT [27]. Especially, *training with the Abilene dataset leads to only* 3.4% *lower* F_1 *score and* 16.9% *lower AUC when testing with the GEANT dataset, compared with the case where the model is trained with GEANT dataset itself.*

D. Attention and Interpretability

Besides improving the performance and making the model generalizable to different topologies, the attention mechanism of the GAT model also brings some interpretability to the network operator. By comparing the value α_{vu} of each adjacent link $u \in \Phi(v)$, we are able to figure out which links have a stronger impact on the prediction for the congestion events happening on link v. As an example, we plot the α_{vu} values of each pair of links in the Abilene dataset when $\beta = 1.5$ in a heat map, as depicted in Figure 6(a). Here, we take average values over all attention heads of the first attention layer, and over all input samples of the test set. Figure 6(b) is the same graph where we zoom in and only keep the links in one direction (the 15 links marked in Figure 3(a)). It indicates that *besides the link itself (i.e., the diagonal element), it is common that some adjacent links may raise a strong attention when predicting a link's congestion.* In this way, it is possible for the network operator to understand the decision of the learning model and trace the congestion events to some extent.

While we always have $\sum_u \alpha_{vu} = 1$ for each link v, we can also calculate the value of $\sum_v \alpha_{vu}$, which reflects the total impact a link u exerts on the whole network. We refer to this value as the *attention score*, and depict it in the

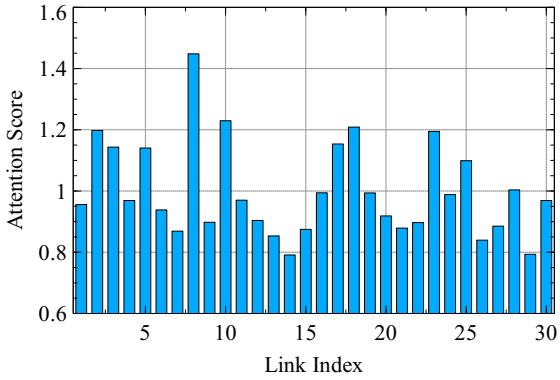


Figure 7. Importance of links in Abilene network by summing up the attention coefficients a link contributes to other links.

Training Set	Accuracy	Precision	Recall	F_1 Score	AUC
Full Abilene Topology	0.9605	0.8246	0.7441	0.7822	0.8354
15 Links with Low Scores Removed	0.9531	0.8017	0.6748	0.7328	0.7884
15 Links with High Scores Removed	0.9410	0.7389	0.5883	0.6550	0.6711

Table VI
THE IMPACT ON PERFORMANCE METRICS WHEN MASKING SOME LINKS OUT OF THE TRAINING SET.

Figure 7. The score varies a lot from about 0.8 to 1.4, implying that some links play a more important role than others in the congestion prediction. To verify this intuition, we design another experiment scenario where the traffic traces of half links are masked off (removed) from the training set, while the tests are still performed on the complete topology. As shown in Table VI, if we remove the 15 links with the lowest attention scores from the training set, there is only a minor performance loss (6.3% in F_1 score and 5.6% in AUC) compared with the complete training set. However, if we remove the 15 links with the highest attention scores, both F_1 score and AUC reduce a lot.

Such information revealed by the attention mechanism is especially useful when the resources for network monitoring are limited, since network monitoring usually introduces non-negligible additional traffic overheads. By using such information, a network operator can gain similar performance of congestion prediction by monitoring and collecting network statistics from only a subset of all network links (those with the highest attention scores).

V. Related Work

A. Network Traffic Prediction

Accurately capturing and predicting the traffic dynamics in a network is of utmost importance for network operators and has become prerequisite in many advanced traffic control and management practices. Related works have proposed several

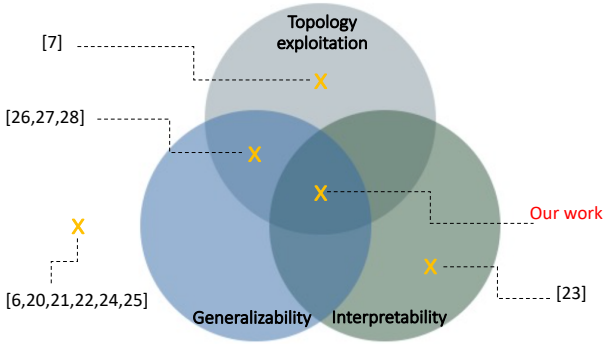


Figure 8. Classification of related works. Our work in this paper is the only one fulfilling all three design goals at the same time.

Machine Learning-based methods targeting different variants of this problem.

The most well-investigated variant is the *prediction of traffic matrices* in a network from previous ones, a task that can be performed with Recurrent Neural Networks (RNNs) equipped with Long Short-Term Memory (LSTM) units [6], combinations of convolutional neural networks (CNNs) and RNNs (to extract both spatial and temporal information from the traffic flows for better prediction accuracy) [20], LSTM models with autocorrelation coefficients [21], combinations of Gated Recurrent Units (GRU) and CNNs [22], and Attention-based Convolutional Recurrent Neural Network models (able to capture intra-flow dependencies and inter-flow correlations) [23].

A second variant of the problem targets the *prediction of the link loads* in a network. We note that we cannot directly infer the link loads from a traffic matrix, because link loads are also affected by the network topology and routing strategies. Therefore, this is a different problem. Previous works addressed this problem by using Support Vector Machine (SVM) [24], and Diffusion Convolution Recurrent Neural Network (DCRNN) [7] methods.

While traffic matrix and link load predictions are kinds of regression problems, another question is how to *predict congestion events* in a network which is a classification kind of problem. This problem can be addressed by solving first the regression problem of predicting the link load values and then defining a threshold above which the load is regarded as overflowing causing congestion (e.g., see [7]). Alternatively, a neural network can be trained directly to output the binary classification result, as modeled in [25]. In our evaluation, we considered both the classification and regression versions of the problem.

Despite relevant and interesting, the above methods neither exploit the topological information for better predictions (with the exception of [7]), nor generalize over topologies unseen during training (which is a capability of graph-based NNs), nor provide any mechanism for interpretation (with the exception of [23]). A classification of the related works is illustrated in Figure 8.

B. Graph-based Neural Networks

Graph-based NNs have been applied to networking applications in the past to benefit from their ability to generalize over topologies unseen during training [26], [27], [28]. Nevertheless, these works focused on routing applications rather than prediction of traffic. Besides, none of these works considered the specific GAT model which has extra benefits, including the attention mechanism which facilitates interpretability of the decision-making process.

C. Interpretability of Neural Networks

For completeness, we note that there exist several other methods for achieving interpretability of the DNN model, including saliency maps based on gradients methods [29]. Nevertheless, the GAT model offers intermediate representations (attention coefficients) that can be directly used for interpreting the decisions.

Overall, while the area of network traffic prediction is well-investigated, the application of graph-based NN models in general and the GAT model in particular is novel and promising as we showed by our evaluation results in Section IV.

VI. Conclusion

In this paper, we employed a recently developed graph-based DNN model, the GAT, to address fundamental limitations of other models in literature when applied to networking applications. We demonstrated the power of this model for a fundamental application of predicting congestion events in a network. Evaluations on three real networks showed superior prediction accuracy compared to four state-of-the-art learning methods with a small loss in accuracy when generalizing to a topology of similar size to the original topology the model was trained. However the generalizability has a limit and the loss may be larger for networks of much different sizes and structures. The information revealed by the GAT's attention mechanism can be useful to network operators for designing practical lightweight network monitoring solutions. As a topic of future work, we plan to investigate and compare with alternative interpretability methods (such as saliency maps) for the same congestion prediction application, as well as extend our work for additional networking applications.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", Nature, vol. 521, pp. 436–444, 2015.

[2] M. Leshno, V.Y Lin, A. Pinkus, S. Schocken, "Multilayer Feedforward Networks with a Nonpolynomial Activation Function Can Approximate Any Function", Neural networks, vol. vol 6, no. 6, pp.861–867, 1993

[3] H. Mao, R. Netravali, M. Alizadeh, "Neural Adaptive Video Streaming with Pensieve", ACM Sigcomm, 2017.

[4] H. Mao, M. Schwarzkopf, S. Bojja Venkatakrishnan, Z. Meng, M. Alizadeh, "Learning Scheduling Algorithms for Data Processing Clusters", ACM Sigcomm, 2019.

[5] A. Valadarsky, M. Schapira, D. Shahaf, A. Tamar, "Learning to Route", ACM HotNets, 2017.

[6] A. Azzouni, G. Pujolle, "NeuTM: A Neural Network-based Framework for Traffic Matrix Prediction in SDN", NOMS 2018.

[7] D. Andreoletti, S. Troia, F. Musumeci, G. Silvia, G.A. Maier, M. Tornatore, "Network Traffic Prediction based on Diffusion Convolutional Recurrent Neural Networks", IEEE Infocom, 2019.

[8] Y. Zheng, Z. Liu, X. You, Y. Xu, J. Jiang, "Demystifying Deep Learning in Networking", ACM Sigcomm APNet Workshop, 2018.

[9] Z. Meng, M. Wang, J. Bai, M. Xu, H. Mao, H. Hu, "Interpreting Deep Learning-based Networking Systems", ACM Sigcomm, 2020.

[10] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks", IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 1, pp. 4-24, Jan. 2021.

[11] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph Attention Networks", ICLR, 2017.

[12] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A System for Large-scale Machine Learning", USENIX OSDI, 2016.

[13] F. Scarselli, M. Gori, A.C Tsoi, M. Hagenbuchner, G. Monfardini, "The graph Neural Network Model", IEEE Transactions on Neural Networks vol. 20, no. 1 pp. 61-80, 2009.

[14] L.B. Almeida, "Artificial Neural Networks", IEEE Press, Piscataway, NJ, USA, Chapter A Learning Rule for Asynchronous Perceptrons with Feedback in a Combinatorial Environment, pp. 102–111, 1990.

[15] S. Orlowski, M. Pioro, A. Tomaszewski, R. Wessaly, "SNDlib 1.0—Survivable Network Design Library", Networks: An International Journal, vol. 55, no. 3, pp. 276-286, 2010.

[16] T. Fawcett, "An Introduction to ROC Analysis", Pattern Recognition Letters, vol. 27, no. 8, pp. 861–74, 2006.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, "Attention Is All You Need", Advances in neural information processing systems, pp. 5998-6008, 2017.

[18] D.P. Kingma, J. Ba. "Adam: A Method for Stochastic Optimization", arXiv preprint arXiv:1412.6980, 2014.

[19] D. Grattarola, C. Alippi, "Graph Neural Networks in Tensorflow and Keras with Spektral", arXiv preprint arXiv:2006.12138, 2020.

[20] Y. Liu, H. Zheng, X. Feng, Z. Chen, "Short-term Traffic Flow Prediction with Conv-LSTM", WCSP, 2017.

[21] S. Xiang, Y. Qin, C. Zhu, Y. Wang, H. Chen, "Long Short-term Memory Neural Network with Weight Amplification and its Application into Gear Remaining Useful Life Prediction", Engineering Applications of Artificial Intelligence, vol. 91, 2020.

[22] X. Cao, Y. Zhong, Y. Zhou, J. Wang, C. Zhu and W. Zhang, "Interactive Temporal Recurrent Convolution Network for Traffic Prediction in Data Centers", IEEE Access, vol. 6, pp. 5276-5289, 2018.

[23] K. Gao, D. Li, L. Chen, J. Geng, F. Gui, Y. Cheng, Y. Gu, "Incorporating Intra-flow Dependencies and Inter-flow Correlations for Traffic Matrix Prediction", IEEE/ACM IWQoS, 2020.

[24] P. Bermolen, D. Rossi, "Support Vector Regression for Link Load Prediction", Computer Networks, vol. 53, no. 2, pp. 191-201, 2009

[25] E. Rapaport, I. Poese, P. Zilberman, O Holschke, R. Puzis, "Predicting Traffic Overflows on Private Peering", arXiv preprint arXiv:2010.01380, 2020.

[26] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, A. Cabellos-Aparicio, "RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN", IEEE Journal on Selected Areas in Communications, vol. 38, no. 10, pp. 2260-2270, 2020.

[27] J. Suárez-Varela, S. Carol-Bosch, K. Rusek, P. Almasan, M. Arias, P. Barlet-Ros, A. Cabellos-Aparicio, "Challenging the Generalization Capabilities of Graph Neural Networks for Network Modeling", ACM SIGCOMM Conference Posters and Demos, 2019.

[28] P. Almasan, J. Suárez-Varela, A. Badia-Sampera, K. Rusek, P. Barlet-Ros, A. Cabellos-Aparicio, "Deep Reinforcement Learning Meets Graph Neural Networks: Exploring a Routing Optimization Use Case", arXiv preprint arXiv:1910.07421, 2019.

[29] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, B. Kim, "Sanity Checks for Saliency Maps", NeurIPS, 2018.

[30] https://www.dropbox.com/s/ghetqx9mczts1vc/gat_congestion_prediction.zip

[31] L. Jun, Z. Shunyi, L. Yanqing, Z. Zailong, "Internet traffic classification using machine learning", Second International Conference on Communications and Networking in China, pp. 239-243, 2007.

[32] S. Floyd, "TCP and explicit congestion notification", ACM SIGCOMM Computer Communication Review, vol. 24, no. 5, pp. 8-23, 1994.