A directed graph convolutional neural network for edge-structured signals in link-fault detection

Michael Kenning, Jingjing Deng, Michael Edwards, Xianghua Xie

January 4, 2022

Abstract

Kipf and Welling [1] The growing interest in graph deep learning has led to a surge of research focusing on learning various characteristics of graph-structured data. Directed graphs have generally been treated as incidental to definitions on the more general class of undirected graphs. The implicit class imbalance in some graph problems also proves difficult to tackle. Moreover, a body of work has begun to grow that considers how to learn signals structured on the edges of graphs. In this paper, we propose the directed graph convolutional neural network (DGCNN), and describe a simple way to mitigate the inherent class imbalance in graphs. The model is applied to edge-structured signals from datacenter simulations using the structure of a directed linegraph to represent the second-order structure of its underlying graph. We demonstrate that the DGCNN's improves over undirected models and other directed models by applying our model to locating link-faults in a datacenter simulation.

1 Introduction

Until recently convolutional models were only suitable to Euclidean domains, such as images or sound, because the convolution kernel and the structure of such domains have a grid-like structure in common. Innovations over the last decade have however permitted the extension of conventional deep learning to non-Eucildean structures; graphs are among the more popular structures. The convolution operation is redefined for graphs in two primary ways: spectrally using spectral filters to approximate spatial convolution; and spatially, defined analogously to conventional spatial convolution. Under certain conditions the two are identical [1], but the spatial approaches exhibit the greatest variance of definition. Graph deep learning has been successfully applied to a number of domains. A graph can represent the atoms and chemical bonds of molecules, for example—one of the earliest applications was for learning molecular fingerprints [2]—but it has also been successfully applied to prediction on road-traffic networks [3].

1.1 Deep Learning on Graphs

The first convolutional methods on graphs were spectral. Bruna et al. [4] proposed implementing convolution on graphs as an filtering operation in the spectral domain. As spectral decomposition is however computationally expensive to compute, subsequent work sought to approximate it using Chebyshev polynomials [1, 5] and Cayley polynomials [6]. Few spectral approaches exist in the literature for directed graphs. Ma et al. [7] adapted spectral convolution to directed graphs with Perron vectors; the formulation was simplified by Li et al. [8] who also included both directions of the graph signals by using the in- and out-degree matrices.

An alternative formulation is spatial convolution on graphs, analogous to spatial convolution on Euclidean domains. Broadly speaking, for each vertex of the graph, a set of vertices constituting its neighbors is aggregated and a learned function is computed over all the signals. Owing to the variability of vertex degree across the graph, the chief difficulty is choosing a way to bind parameters to a neighborhood of signals. There is much variance among the spatial approaches because of the different definitions of neighborhoods and the different kinds of functions that can be applied to the neighborhoods' signals.

To learn molecular fingerprints, Duvenaud et al. [2] applied a different function to each vertex neighborhood according to its degree. The process is repeated over multiple layers. Niepert et al.'s PATCHY-SAN [9] selects a subset of vertices first by labeling the graph, then aggregates a constant number of vertices and finally normalizes the selected vertices in some order to multiply them by a set of parameters. MoNet [10] projects the graph vertices into a pseudo-coordinate space where Gaussian distributions are learned, from which the parameters of the model are sampled. GraphSAGE's mean aggregator [11] simply averages the signals of a target vertex and its first-order neighbors and multiplies the result by a parameter matrix. Xu et al. [12] in contrast propose a summation of the signals, passing the result through a multi-layer perceptron. Veličković et al. [13] introduced attention to graph deep learning. An attention mechanism assigns every pair of vertices in a first-order neighborhood an attention coefficient.

To the best of our knowledge, there are very few spatial approaches explicitly designed for directed graphs. The diffusion convolutional recurrent neural network (DCRNN) [3] uses directed information to learn a diffusion matrix of graph signals to model the spatial patterns in temporal data. Tong et al. [14] proposed the Fusion model, a spatial approach to convolution that fuses three components of proximity matrices. The first proximity matrix captures and sums each vertices' neighbors irrespective of direction. The second two proximity matrices capture vertices structurally similar to the target vertex according to the existence of common in- or out-neighbors.

The processing of edge features on graphs is the subject of recent attention [15]. Edge features are frequently used with graphs to describe quantitative properties of edges, for example the bond-distance in molecular graphs [16]. Zheng et al. [17] used recurrent neural networks to model geographical in-flow

of COVID-19 infections, represented as graph edges. Chen et al. [18] constructed a directed linegraph from an undirected underlying graph for the purposes of community detection, where the edges features are a function of oriented pairwise vertex features.

1.2 Machine Learning and Datacenters

A datacenter's network administrator faces numerous possible sources of failure. At hand are the various streams of diagnostic information describing different aspects of the network, no single stream providing a total description [19]. Diagostic aids can summarize data into understandable representations [20, 21]. There are software agents that monitor the health of traffic flows in a network, such as PingMesh [22] and 007 [23], the latter 007 being particular interest in the present work. Simple linear analysis of traffic flows has also been proposed for the location of failures [24]. Some conventional machine learning approaches have been applied in conjunction with diagnostic agents [25].

Deep learning approaches such as plain neural networks [26] and convolutional neural networks [27] have also been applied to learning tasks on datacenters. Yet although the datacenter's structure is well represented by a graph, seemingly few approaches exist in the literature that use them. Fang et al. [28] modeled demand on a cellular network using a Graph Convolutional Network [1] and long short-term memory. Wang et al. [29] predicted datacenter traffic with the structure of a directed graph. Most relevant is Andreoletti et al.'s work [30]; they used the DCRNN [3] to predict traffic loads on the links of a datacenter availing the structure of a directed graph.

1.3 Contributions

In this paper we tackle the task of locating link-failures in a datacenter. Few approaches in the literature use graphs to provide structural information to models, despite the recent surge of innovation. The machines and physical connections of datacenters can be represented by vertices and *directed* edges, forming a *directed graphs* Andreoletti et al. [30]. Unlike most situations, however, the data in our problem is structured on the edges of the datacenter graph. While we represent the datacenter as a directed graph, in distinction to earlier work, we also construct a *directed linegraph* [31] in order to structure an link-wise learning task. Using this representation, we describe and evaluate a new model for directed graphs. To the best of our knowledge, our paper is the first to propose the construction of a directed linegraph and the first to use its structure to learn on edge-structure signals to locate faults. Additionally we believe that we are the first to discuss the attendant issues of what we call *inverse edges*. Additionally we propose a new technique to counteract the inherent class imbalance in the graph, which cannot be remedied by simply oversampling.

Our contributions are as follows:

- we present the directed graph convolutional neural network (DGCNN), a new model for learning on directed graphs,

- we use a directed linegraph to learn a task on the edges of a graph, while describing ways to tackle *inverse edges*, a unique problem that arises from the use of directed linegraphs in this context; and
- we present a technique to mitigate the inherent class imbalance in the graph, by using a domain-motivated threshold to exclude vertex-wise losses from the training, and use a vertex-wise odds-ratio weighting to scale the losses of the remainder.

2 Methodology

2.1 Graph-theoretical Definitions

A graph $G = \langle V, E \rangle$ is defined by a set of *vertices* V and *edges* E. The order of the graph is $n = |V|$, the number of vertices, and its size is the number of edges $m = |E|$. If two vertices $x, y \in V$ are *joined* by an edge $xy \in E$, they are called *adjacent*. The vertices x, y are referred to as *end-vertices* of xy. (For simplicity's sake we also write $x, y \in G$ and $xy \in G$ respectively.) The set E is hence a subset of all unordered pairs of V. If the edges in E are ordered, the graph and its edges are described as *directed*. The directed edge xy has a *start-vertex* x and *end-vertex* y, and $xy \neq yx$.

The neighborhood of a *target vertex* x, denoted $\Gamma(x)$, is the set of vertices incident to x: $\Gamma(x) = \{y \mid xy \in E\}$. The vertex x is sometimes implicitly included in $\Gamma(x)$ as a self-loop. A vertex x's degree $d(x)$ is its neighborhood's cardinality $d(x) = |\Gamma(x)|$ if only the first-order neighbors are included by $\Gamma(x)$. The kth order neighborhood is denoted by $\Gamma_k(x)$. A neighborhood of a directed graph is factored into its in-neighbors $\Gamma_-(x) = \{y \mid yx \in E\}$ and out-neighbors $\Gamma_+(x) = \{y \mid xy \in E\}$. Vertex degree is likewise factored: $d_-(x) = |\Gamma_-(x)|$ and $d_+(x) = |\Gamma_+(x)|$.

Aspects of the graph can be represented as matrices indexed by the vertices. The adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a binary matrix where $\forall xy \in G,\ A_{xy} = 1$. The degree matrix $\mathbf{D} = \mathrm{diag}(\mathbf{A}\mathbb{1})$ is a diagonal matrix where $\mathbf{D}_{xx} = d(x)$. The adjacency matrix of a directed graph is asymmetric, from which we obtain a negative degree matrix $\mathbf{D}_- = \mathrm{diag}(\mathbf{A}^\top\mathbb{1})$ for the in-degrees, and the positive degree matrix $\mathbf{D}_+ = \mathrm{diag}(\mathbf{A}\mathbb{1})$ for the out-degrees.

The *linegraph* $L(G) = \langle V_L, E_L \rangle$ is constructed from an *underlying graph* G, such that each edge $xy \in G$ is represented by a vertex $\alpha \in V_L$. Two vertices $\alpha, \beta \in L(G)$ are adjacent if the underlying *edges* are also adjacent, meaning they have at least one end-vertex in common. Otherwise, the same notation as graphs applies to linegraphs. The *directed linegraph* is constructed similarly, except the vertices are joined only when their direction is coincident [31]. That is, there is a directed edge $\alpha\beta \in L(G)$ if there are two edges $xy, wz \in G, \alpha, \beta \in L(G)$ such that $xy = \alpha, wz = \beta, y = w$. If both $x = z$ and $y = w$, then we term the two edges *inverse edges* of one another. The linegraph may be said to represent the second-order or edge structure of G.

Graph signals are frequently structured on the vertices of the graph, but can also be structured on the edges [15]. For generality, the graph-structured signals are formulated as a mapping from the graph to the c-dimensional signals $f : G \to \mathbb{R}^{k \times c}$ where k can be n or m. The graph signals are therefore denoted $f(G), f(V)$ or $f(E)$, or simply f. We use $f(x) \in \mathbb{R}^c$ to represent the c-dimensional signal on vertex x.

2.2 Representing a Datacenter as a Graph

A graph can represent a datacenter if we assign each machine a vertex in the graph and draw an edge between a vertex pair when the two machines they represent are connected physically. The connections are directed; each pair of machines has two connections, one upstream, one downstream. Directed edges can represent the separate, directed connections, yielding a directed graph.

As mentioned above, graph convolution is more often defined on graph vertices. The diagnostic data from the 007 simulator (Section 3.1) is not structured on the vertices, however, but on the edges. We use a linegraph (Section 2.1) to represent the links as vertices. Specifically we construct a directed linegraph [31] from the underlying directed graph.

However, there is a problem arising from the nature of the underlying graph. For each connection from machine A to B, there is an inverse connection from B to A, and *vice versa*. We refer to both edges as the *inverse edges* of one another. A special consideration thus arises in the construction of the directed linegraph. The inverse edges become a pair of vertices in the directed linegraph, and strictly speaking, the two edges (now vertices) would be connected and become the in- and out-neighbors of one another (Section 2.1). Modeling the inverse edges as a member of both groups would be double counting however. A decision must therefore be made on how it is handled, most appropriately by reference to the domain. Traffic passing along an edge in the datacenter is unlikely to pass through its inverse. On these ground the two inverse edges could remained disconnected. We test including and excluding the inverse edge in this paper.

One disadvantage of linegraphs should be mentioned, noted elsewhere [15]. The linegraph's order is the size of its underlying graph. The linegraph's size increases at a greater rate. Were the graph to become too large, the linegraph could become unmanageable if it does not remain sparse.

2.3 The Directed Graph Spatial Convolution

A spatial convolution on the graph is a localized function over the signals of a vertex and its neighbors. The function's form between methods is highly variable. One spatially defined function is GraphSAGE's mean aggregator [11]:

$$f'(x) = \frac{\theta}{d(x)+1} \sum_{y \in \{x\} \cup \Gamma(x)} f(y), \tag{1}$$

where $\theta \in \mathbb{R}^c$ is the set of learned parameters.

Methods defined in such away do not account for the direction of signals in the graph. We propose the directed graph convolutional neural network (DGCNN) to explicitly account for the direction of neighbors:

$$f'(x) = \frac{\theta_0}{d_-(x)} \sum_{y \in \Gamma_-(x)} f(y) + \frac{\theta_1}{d_+(x)} \sum_{y \in \Gamma_+(x)} f(y) + \theta_2 f(x), \tag{2}$$

where $\theta_0, \theta_1, \theta_2 \in \mathbb{R}^c$ are the sets of learned parameters for the in- and out-neighbors and target vertex respectively. Defined over all vertices and multiple output channels, the equation is simplified to

$$f' = \hat{\mathbf{A}}^\top f\Theta_0 + \hat{\mathbf{A}} f\Theta_1 + f\Theta_2, \tag{3}$$

where $\hat{\mathbf{A}}$ is the row-normalized adjacency matrix and $\Theta_0, \Theta_1, \Theta_2 \in \mathbb{R}^{c \times d}$ are the sets of learned parameters, where d is the output's dimensionality.

In cases where there are inverse edges in the directed graph, they can be included in each neighborhood as undifferentiated members; or as a separately learned term:

$$f' = \hat{\mathbf{A}}^\top f\Theta_0 + \hat{\mathbf{A}} f\Theta_1 + f\Theta_2 + \mathbf{B} f\Theta_3, \tag{4}$$

where $\mathbf{B} \in \mathbb{R}^{n \times n}$ is a mask indexing the inverse edges in $\mathbf{A}$.

2.4 The DGCNN Architecture

In this section we describe the structure of the directed graph convolutional neural network (DGCNN) model, illustrated in Fig. 1. The input is passed through three sets of DGCNN layer, batch normalization and activation layer. The output is passed through a dense layer that maps each vertex's signals to a single sigmoid-activated output, the model's prediction. We decided on 3 convolutional layers and 10 output maps for each layer following experimentation: after 3 convolutional layers the performance improvement petered out. We found that batch normalization with a momentum of 0.99 stabilized learning. We used LeakyReLU rather than ReLU as an activation functions, since the latter led to dead neurons. The model handles the class imbalance by specially weighting the loss-function and biasing the last dense layer.

3 Experiment

The experiments below were programmed in Python 3.6.6. The models were trained in Python Tensorflow 2.4.1. The GraphSAGE model is the Spektral 1.0.4 implementation.

3.1 The Dataset

The dataset is generated by the 007 simulator developed by Arzani et al. [23]. The diagnostic system uses agents to record the paths of transmission errors to

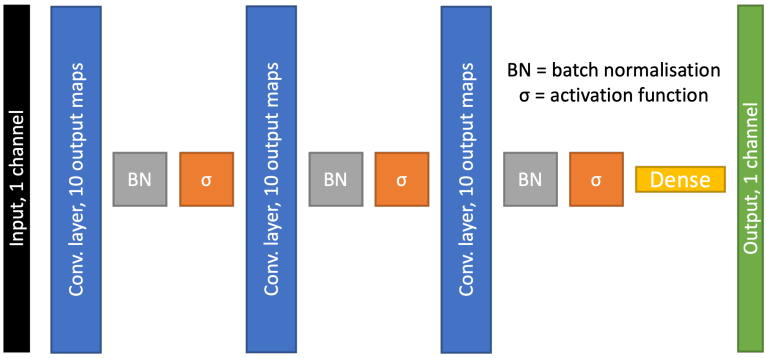

Figure 1: The structure of the DGCNN, where BN and σ acronymize the batch normalization layer and activation functions respectively. In the comparison models we have replaced only the convolutional layer given here.

locate the link faults responsible for the packet-drops, and assigns potentially culpable links a blame-score. The simulator uses a FAT topology for the datacenter, which we configured to have 10 T2 switches and two pods of 10 T1 switches and 10 top-of-rack (ToR) switches, each connecting to 24 Hosts, giving 540 machines and 1,440 links. We are not interested in the Host links since faults there can be trivially diagnosed. Rather we are interested in the type-1 (T1–ToR), type-2 (ToR–T1), type-3 (T1–T2) and type-4 (T2–T1) links.

In total we ran 2880 simulations, each thirty seconds long, representing 24 hours' worth of datacenter simulation data. At the end of each simulation, each link is assigned a blame-score, the model input. For each simulation between two and ten links are chosen to be faulty. The rate at which these faulty links would drop was decided by which of the four link-types it belonged to. The packet-drop probabilities of each link-type were sampled from the range $[0.01, 0.1]$. In order to add noise to the problem and render it more challenging, healthy links would occasionally cause packet-drops but at a much lower rate of 0.000001, which one would expect in a datacenter, as Arzani et al. observed. The difference in the ranges of the probabilities seems vast, but the faulty links cause wide-ranging collateral failures owing to the displacement of traffic. It is therefore difficult to identify the efficient cause of the collateral failures.

There is a high inherent degree of class imbalance in the dataset that is very difficult to remedy. The datacenter has 1,440 links and a number of failures k is uniformly distributed $k \sim U(2, 10)$ with an expectation $\mathbb{E}(k) = 2 + {}^{10-2}/_{2} = 6$. The expected class imbalance is therefore $\rho = {}^{E(k)}/_{1,440-E(k)} = 4.18 \times 10^{-3}$. It is imperative that we address this fact in training to mitigate against the model's skewing towards the negative samples. We describe how we address the imbalance in Section 3.3.

Table 1: The models we are evaluating in the experiments.

Name	Type	Directed	Inverse edges	Excluding neighbors
DGCNN	Spatial	✓	—	—
DGCNN-R	Spatial	✓	—	—
DGCNN/I	Spatial	✓	✓	—
DGCNN-out	Spatial	✓	—	✓
DGCNN-in	Spatial	✓	—	✓
UGCNN	Spatial	—	—	—
UGCNN/I	Spatial	—	✓	—
UGCNN+I	Spatial	—	✓	—
GraphSAGE	Spatial	—	—	—
P-F model	Spectral	✓	—	—
Fusion model	Spatial	✓	—	—
Dense model	—	—	—	—
RF	—	—-	—	—

3.2 Implementation and Comparisons

We compare the DGCNN with several modifications and other models (Table 1). In each case we replace only the convolutional layers of the model illustrated in Fig. 1. The **UGCNN** models use the same convolutional layer as the DGCNNs; the only difference is that the adjacency matrix is made symmetric:

$$A'_{ij} = \begin{cases} 1, & \text{if } A_{ij} = 1 \text{ or } A_{ji} = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

In the analysis, we use the differences in structure to elucidate how each aspect informs the model; but we are primarily concerned with performance of the DGCNN.

The variations are meant to evaluate empirically several decisions we have made. The residual connections in the **DGCNN-R** counteract any oversmoothing of the signal over the three layers that may undermine the model's performance [32]. The input of each layer is added to the layer's activated output. If the dimensionality of the two tensors differs, the signals are projected using a learned matrix. Hence, an input $\mathbf{h}'_{i-1} \in \mathbb{R}^{n \times c}$ to layer i is added to its output $\mathbf{h}_i \in \mathbb{R}^{n \times d}$ as $\mathbf{h}'_i = \mathbf{h}'_{i-1}\mathbf{W} + \mathbf{h}_i$ where $W \in \mathbb{R}^{c \times d}$ is a learned set of parameters.

We evaluated how the models performed with and without inverse edges. The inverse edges are modeled in **DGCNN/I** and **UGCNN/I** as a separate term of the convolution. In a further undirected variant, **UGCNN+I**, the inverse can be included in the vertices' neighborhoods. If the inverse edges are not relevant to the task on each vertex, inverse edges' signals' inclusion would not help the performance. Adding the inverse edge to a directed and undirected model moreover enables us to compare the study whether the inverse edge provides additional directed information indirectly. The inverse edge's existence depends on directionality in the graph. We speculate that it may therefore contain some discriminative information. We also experiment with the exclusion of whole sets of neighbors according to their incidence. **DGCNN-in** uses only the in-neighbors and **DGCNN-out** uses only the out-neighbors.

GraphSAGE [11] is an undirected spatial technique in the literature against which we compare our models, and the **P-F model** [8] and **Fusion** [14] models are respectively spectral and spatial directed approaches in the literature. A directed *spectral* approach would not fare well in locating several overlapping independent faults at the same time, because the transformation and spectral filtering would have to operate on the graph signals globally to find several independent, localized signals that have no spatial correlation. The dense model's performance tells us how the graph structure informs a model's solution, and the **RF** indicates how deep models compare to a conventional model.

3.3 Experimental Conditions

The training procedure is kept alike between the deeply learned models; the only aspect that varies is the architecture. Each model is trained for 50 epochs.

The training rate is initially 0.1 and annealed to 1×10^{-7} during training using cosine decay.

The loss-function is the binary cross-entropy of the model output and the ground-truth. However, as mentioned in Section 3.1, there is a high class imbalance that we must address. We have modified the loss-function in three ways to mitigate its deleterious effect on learning. Firstly we find the lowest blame-score of the positive samples. This score acts as a threshold on the cross-entropy losses. Namely we discount the loss of samples whose input falls below the threshold. The risk is that the model learns to ignore all samples below the threshold, hence a few below-threshold samples in the test data may be ignored.

Secondly, the loss on each vertex was weighted by the balanced odds-ratio of positive/faulty and negative/healthy links. The weight of a positive/faulty link is $^{(neg+pos)}/_{2 \cdot pos}$, and $^{(neg+pos)}/_{2 \cdot neg}$ for a negative/healthy link, where pos and neg are the numbers of positive and negative samples above the threshold.

Thirdly, we set the bias of the final layer such that the model output reflect the prior distribution of positive samples. The log-ratio of positive to negative samples was computed $b = \ln(^{pos}/_{neg_{\text{total}}})$, where neg_{total} is the number of negative samples ignoring the threshold. The sigmoid layer therefore gives $\sigma(b) = {}^{pos}/_{(pos+neg_{\text{total}})}$.

These conditions largely cannot apply to the RF. The RF uses 100 estimators, each estimator is constructed on a subsampling as large as 10% of the dataset. The classes are balanced according to their proportions in each set of bootstrapping samples, using a built-in parameter that balances the bootstrap samples of each estimator.

3.4 Metrics and Analyses

We compared the models' performances on the test sets across four measures: precision, recall, F_1-score, and McNemar's test. The predictions of two models can be compared directly using McNemar's test [33]. We use Edwards' correction [34]. First we compute two values: n_{sf}, which compares how many times the first model is right and the second model is wrong and *vice versa* n_{fs}. These numbers are additionally interesting in themselves, as they reveal to us the relative strengths of the models. McNemar's test statistic is computed as $\chi^2 = (|n_{sf} - n_{fs}| - 1)^2/(n_{sf} + n_{fs})$. The value χ^2 is approximated by a chi-squared distribution. We can therefore use a binomial test to evaluate the null hypothesis, that the performances of the two models are equal.

We compare the models with six further analyses. The first five concern the architectural features of the models exhibited by the comparison models (Table 1). Firstly we compare the neural-network models to a conventional machine-learning model to determine whether the task is helped by deeply learned models. Secondly we measure the benefit of including direction in our model by comparing our directed model to the undirected variants. Thirdly we measure the effect of excluding a whole set of neighbors—the in- or out-neighbors—from the convolution. Fourthly we measure the effect of including the inverse edge's signal on the efficacy of the directed and undirected models.

Table 2: The reference model for the McNemar's tests is the directed model with the residual connections because it had the best performance by F_1-score. The parameter counts of each model are given for the perspective they give on capacity, training and inference times, *etc.*

Model	No. Params.	F_1-score	Precision	Recall	n_{sf}	n_{fs}	χ^2	p-value
Directed model	731	$\mathbf{0.779\,85 \pm 0.007\,87}$	$0.722\,50 \pm 0.014\,07$	$0.848\,53 \pm 0.036\,82$	0.0	0.0	0.0	0.00000
Directed model with residual connection	941	$0.777\,61 \pm 0.012\,15$	$\mathbf{0.725\,77 \pm 0.012\,37}$	$0.839\,21 \pm 0.045\,52$	418.0	416.4	1.01	0.40969
Directed model including the inverse	941	$0.776\,28 \pm 0.010\,21$	$0.723\,81 \pm 0.014\,09$	$0.838\,70 \pm 0.042\,99$	530.8	516.8	0.57	0.51480
Directed model, out-neighbors only	521	$0.715\,14 \pm 0.051\,62$	$0.667\,50 \pm 0.072\,96$	$0.778\,80 \pm 0.081\,51$	1139.6	643.2	175.28	0.00000
Directed model, in-neighbors only	521	$0.741\,34 \pm 0.032\,82$	$0.708\,36 \pm 0.041\,89$	$0.792\,03 \pm 0.117\,19$	789.0	560.6	40.10	0.00000
Undirected model	731	$0.776\,75 \pm 0.016\,45$	$0.716\,55 \pm 0.032\,39$	$0.855\,14 \pm 0.079\,76$	549.2	513.8	2.30	0.31816
Undirected model including the inverse	941	$0.763\,41 \pm 0.015\,71$	$0.732\,82 \pm 0.024\,56$	$0.800\,67 \pm 0.063\,26$	667.4	612.2	3.30	0.28927
Undirected model with merged inverse	731	$0.745\,35 \pm 0.063\,47$	$0.676\,50 \pm 0.093\,06$	$0.842\,10 \pm 0.075\,11$	883.6	522.4	177.17	0.19262
GraphSAGE model	521	$0.705\,62 \pm 0.072\,61$	$0.613\,86 \pm 0.103\,34$	$0.839\,98 \pm 0.012\,48$	1369.6	543.6	467.21	0.16386
Perron-Frobenius model	521	$0.118\,00 \pm 0.054\,97$	$0.066\,62 \pm 0.034\,98$	$0.597\,47 \pm 0.044\,98$	34123.6	696.6	32091.83	0.00000
Spatial directed fusion model	1243	$0.115\,80 \pm 0.076\,09$	$0.065\,43 \pm 0.048\,22$	$0.637\,00 \pm 0.081\,47$	40394.0	828.6	37980.77	0.00000
Dense model	311	$0.743\,03 \pm 0.006\,03$	$0.640\,16 \pm 0.005\,32$	$\mathbf{0.885\,39 \pm 0.011\,67}$	975.6	514.4	144.34	0.00000
Random forest	100	$0.736\,98 \pm 0.004\,63$	$0.639\,52 \pm 0.005\,76$	$0.869\,54 \pm 0.007\,41$	1043.0	553.2	151.24	0.00000

Finally we compare our directed model with the three graph models from the literature: the GraphSAGE model, the Perron-Frobenius spectral model and the Fusion model.

4 Results

Generally the directed spatial models attain a better balance of precision and recall (Table 2). It is clear that the graph structure informs the model when compared with its absence, as in the Dense model and RF. There is also a clear trade-off between precision and recall. Wherever a model's precision is better than the DGCNNs', that model suffers a worse recall, such as the undirected model. The opposite is observed with the dense model.

With respect to the parameter counts, the picture is not complicated, but our interpretations can be supplemented with reference to McNemar's test. More parameters can increase the capacity of models, but it is not a straightforward relationship [35]. Certainly it takes more computational power.

A subtle yet notable aspect of McNemar's test statistics is the size of n_{sf} and n_{fs} across the models compared to the reference model. The chief difference between these numbers lies in samples that the directed model correctly identifies but the others do not. The two numbers are frequently of the same magnitude or within an order of it, ignoring the P-F model and Fusion model. This concerns not samples that neither model can correctly identify, rather samples that are mutually exclusive. In other words, it seems that if a model is to correctly identify one set of samples, it cannot correctly identify another set. It suggests that the task might be well learned by an ensemble approach.

Conventional machine learning versus deep learning By F_1-score the RF is outperformed by all deep models, with the exception of the literature models, GraphSAGE, P-F model and Fusion model. While the RF has the second-highest recall, the precision falls below the performance of even the dense

model. The deeply learned models offer a clear advantage over conventional machine learning in this scenario.

Directed versus undirected deeply learned models The DGCNN and DGCNN-R outperform the variants of the UGCNNs by F_1-score. Whereas the recall of the UGCNNs is higher than the DGCNN and DGCNN-R's, the precision is poorer. The directed models were however more stable than the undirected models. The difference in precision between the DGCNN and the UGCNN is nearly six percentage points, and the DGCNN-R widens the difference further to ten percentage points.

One must note that the undirected model's architecture is identical to the directed models'; the only difference lies with the adjacency matrix, as we described in Section 3.2. We may therefore confidently attribute the improvement of performance to the direction included in the model, although the difference between the best directed model and the best undirected model is not statistically significant.

Excluding neighbors Excluding neighbors of the DGCNN-in and DGCNN-out markedly undermined their performance. Excluding the out-neighbors in DGCNN-out appears to be more deleterious than excluding the in-neighbors in DGCNN-in, which suggests that the in-neighbors' signals may be marginally more informative than the out-neighbors'. This fact is perhaps owed to the strain effected on links earlier in the traffic flow rather than later. Their F_1-scores are not even better than the dense model's, which has no structural inductive biases at all.

Inverse edges The advantage that the inverse signals gives to a model is not clear. The DGCNN/I performed worse than the DGCNN, and likewise the UGCNN/I and UGCNN+I performed worse than the UGCNN. The UGCNN/I improves minimally on the UGCNN's stability. The results suggest that the undirected models are hitting an upper limit on performance. Directed information does not appear to be supplied to the model indirectly via the inverse edge. That the UGCNN+I performs worse than the UGCNN moreover suggests that the information on the inverse edge is not just useless, but muddies the interpretation of the more informative neighborhood signals. Curiously the DGCNN/I exhibited an inconsiderable improvement of stability over the DGCNN.

Spatial and spectral graph-based approaches in the literature Comparing our directed models to the models in the literature, it is clear that the two directed models, P-F model and Fusion model, perform badly. On the one hand, we believe that it is clear that the spectral direct model, P-F model, is not suitable to the task at hand, where we must detect several uncorrelated signals in the same input space. On the other hand, we consider the Fusion model's design to be responsible for the poor performance. The first-order neighbors in

the convolution are convolved irrespective of the direction of their incidence to each vertex. We believe that this weakens the capacity of the model.

GraphSAGE in contrast performs far better, though still not as well as the UGCNNs, which has the most comparable architecture. It has fewer parameters than the DGCNN and UGCNN, but it is doubtful that this alone accounts for the difference in performance between the two. The explanation is better explained by the way GraphSAGE levels the distinction between the target vertex and its neighbors, whereas the DGCNN and UGCNN model the neighbors separately. A similar smoothing effect occurs in the UGCNN+I. Overall it appears that GraphSAGE and the undirected approaches are limited in their performance.

Performance stability over link-types and failures in simulation The performance of the models as measured by the F_1-score varies very little with respect to the number of failures in a simulation (Fig. 2a). This is unsurprising, as the number varies only from 2 to 10 throughout the simulations. With respect to the different link-types, however, the difference in performance within models is stark (Fig. 2b). Further analysis shows that the models struggled to identify the type-3 and type-4 faults, the links that resided furthest from the hosts. We suspect that a low number of in-neighbors and out-neighbors of these links is responsible: type-3 links have 11 in-neighbors and 1 out-neighbor; type-4 links have 1 in-neighbor and 11 out-neighbors. Whereas type-1 links have 11 in-neighbors and 33 out-neighbors (type-2 contrariwise). Sufficient information for the type-3 and type-4 links might be lacking to draw conclusions about faults.

5 Conclusion

In this paper, we presented the directed graph convolutional neural network (DGCNN). Few models have been explicitly presented in recent years to operate on directed graphs. The DGCNN aggregates the signals of the in- and out-neighbors separately. We also propose using a directed linegraph to learn on domains that are ordinarily represented as directed graphs but whose signals occur on its directed edges. The directed linegraph leads to another issue, the inverse edge, which we propose to handle by including it in our definition of convolution. Additionally we proposed a way of mitigating the inherent class imbalance in the graph. With the convolution defined, we evaluated the performance of our model against variants and models found in the literature. We found that the DGCNN outperforms the other models we tested, indicating that including the direction improves a model's capacity to learn. The DGCNN offers a new formulation of modeling of domains represented as directed graphs. Unlike previous work, we account for direction when learning directed signals, especially edge-structured signals.

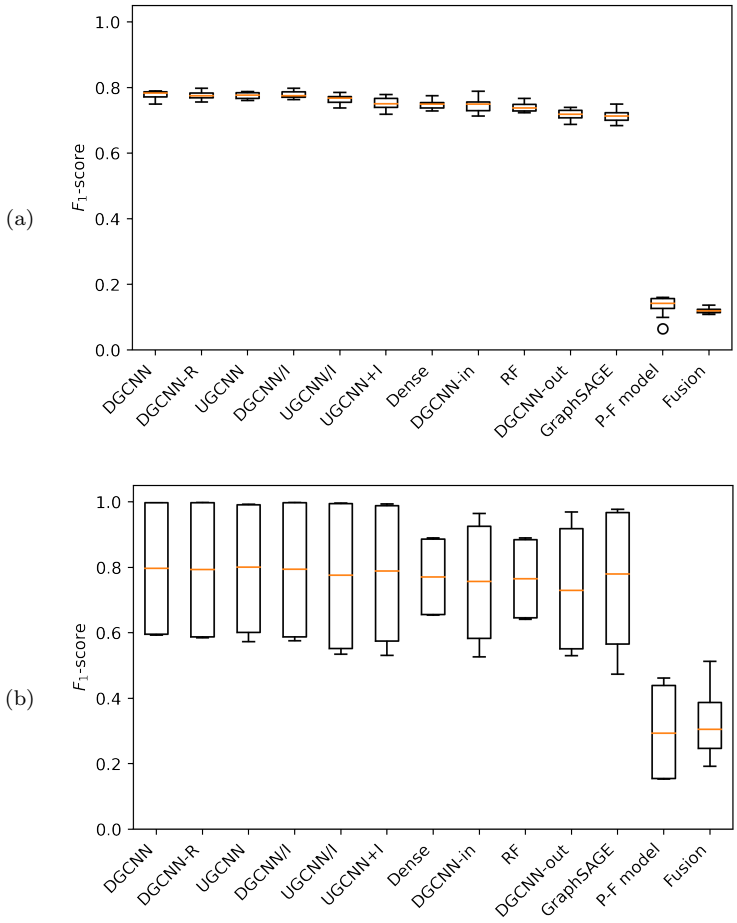

Figure 2: For (a) for each model on each fold we grouped the simulations by the number of failures and computed the F_1-scores. For (b) for each model on each fold we masked for each edge type and then computed the F_1-scores. In both cases we average the scores across the folds. The box-plot shows the average, first standard deviation and 25th and 75th percentiles of those F_1-scores across the folds. Circles are outlier samples.

References

[1] T. N. Kipf, M. Welling, Semi-Supervised Classification with Graph Convolutional Networks, in: ICLR, 2017.

[2] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, *et al.*, Convolutional Networks on Graphs for Learning Molecular Fingerprints, in: NeurIPS, 2015, pp. 2224–2232.

[3] Y. Li, R. Yu, C. Shahabi, *et al.*, Diffusion convolutional recurrent neural network: Data-driven traffic forecasting, in: ICLR, 2018.

[4] J. Bruna, W. Zarembda, A. Szlam, Y. Lecun, Spectral Networks and Locally Connected Networks on Graphs, in: ICLR, 2014.

[5] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering, in: NeurIPS, 2016, pp. 3844–3852.

[6] R. Levie, F. Monti, X. Bresson, M. M. Bronstein, CayleyNets: Graph Convolutional Neural Networks with Complex Rational Spectral Filters, IEEE Transactions on Signal Processing (2019).

[7] Y. Ma, J. Hao, Y. Yang, *et al.*, Spectral-based Graph Convolutional Network for Directed Graphs, 2019. `arXiv:1907.08990`.

[8] C. Li, X. Qin, X. Xu, *et al.*, Scalable Graph Convolutional Networks With Fast Localized Spectral Filter for Directed Graphs, IEEE Access 8 (2020) 105634–105644.

[9] M. Niepert, M. Ahmed, K. Kutzkov, Learning Convolutional Neural Networks for Graphs, in: ICML, 2016, pp. 2014–2023.

[10] F. Monti, D. Boscaini, J. Masci, *et al.*, Geometric deep learning on graphs and manifolds using mixture model CNNs, in: IEEE CVPR, 2017.

[11] W. L. Hamilton, Z. Ying, J. Leskovec, Inductive Representation Learning on Large Graphs, in: NeurIPS, Curran Associates, Inc., 2017, pp. 1024–1034.

[12] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How Powerful are Graph Neural Networks?, in: ICLR, 2019.

[13] P. Veličković, A. Casanova, P. Liò, *et al.*, Graph Attention Networks, in: ICLR, 2018.

[14] Z. Tong, Y. Liang, C. Sun, *et al.*, Directed Graph Convolutional Network, 2020. `arXiv:2004.13970`.

[15] S. Georgousis, M. P. Kenning, X. Xie, Graph Deep Learning: State of the Art and Challenges, IEEE Access 9 (2021) 22106–22140.

[16] C. Chen, W. Ye, Y. Zuo, *et al.*, Graph Networks as a Universal Machine Learning Framework for Molecules and Crystals, Chemistry of Materials 31 (2019) 3564–3572.

[17] Y. Zheng, Z. Li, J. Xin, G. Zhou, A Spatial-temporal Graph based Hybrid Infectious Disease Model with Application to COVID-19, in: ICPRAM, 2021.

[18] Z. Chen, J. Bruna, L. Li, Supervised community detection with line graph neural networks, in: ICLR, 2019.

[19] P. Gill, N. Jain, N. Nagappan, Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications, ACM SIGCOMM Computer Communication Review 41 (2011) 350.

[20] T. Pelkonen, S. Franklin, J. Teller, *et al.*, Gorilla: A Fast, Scalable, In-Memory Time Series Database, Proceedings of the VLDB Endowment 8 (2015) 1816–1827.

[21] A. Chircu, E. Sultanow, D. Baum, *et al.*, Visualization and Machine Learning for Data Center Management, in: INFORMATIK, 2019, pp. 23–35.

[22] C. Guo, L. Yuan, D. Xiang, *et al.*, Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis, SIGCOMM Computer Communication Review 45 (2015) 139–152.

[23] B. Arzani, S. Ciraci, L. Chamon, *et al.*, 007: Democratically Finding the Cause of Packet Drops, in: USENIX Symposium on Networked Systems Design and Implementation, 2018, pp. 419–435.

[24] Y. Zhang, Z. Ge, A. Greenberg, M. Roughan, Network Anomography, in: ACM SIGCOMM Conference on Internet Measurement, 2005.

[25] H. Ren, L. Nie, H. Gao, *et al.*, NetCruiser: Localize Network Failures by Learning from Latency Data, in: IEEE International Conference on Smart Internet of Things, 2020, pp. 23–30.

[26] D. Rafique, T. Szyrkowiec, H. Grieser, *et al.*, Cognitive Assurance Architecture for Optical Network Fault Management, Journal of Lightwave Technology 36 (2018) 1443–1450.

[27] W. Wang, Y. Sheng, J. Wang, *et al.*, HAST-IDS: Learning Hierarchical Spatial-Temporal Features Using Deep Neural Networks to Improve Intrusion Detection, IEEE Access 6 (2018) 1792–1806.

[28] L. Fang, X. Cheng, H. Wang, L. Yang, Mobile Demand Forecasting via Deep Graph-Sequence Spatiotemporal Modeling in Cellular Networks, IEEE Internet of Things Journal 5 (2018) 3091–3101.

[29] X. Wang, Z. Zhou, F. Xiao, *et al.*, Spatio-Temporal Analysis and Prediction of Cellular Traffic in Metropolis, IEEE Transactions on Mobile Computing 18 (2019) 2190–2202.

[30] D. Andreoletti, S. Troia, F. Musumeci, *et al.*, Network Traffic Prediction based on Diffusion Convolutional Recurrent Neural Networks, in: IEEE INFOCOM 2019, 2019, pp. 246–251.

[31] M. Aigner, On the linegraph of a directed graph, Mathematische Zeitschrift 102 (1967) 56–61.

[32] G. Li, M. Muller, A. Thabet, B. Ghanem, DeepGCNs: Can GCNs Go As Deep As CNNs?, in: ICCV, 2019, pp. 9266–9275.

[33] Q. McNemar, Note on the sampling error of the difference between correlated proportions or percentages, Psychometrika 12 (1947) 153–157.

[34] A. L. Edwards, Note on the "correction for continuity" in testing the significance of the difference between correlated proportions, Psychometrika 13 (1948) 185–187.

[35] B. Neyshabur, S. Bhojanapalli, D. McAllester, N. Srebro, Exploring Generalization in Deep Learning, in: NeurIPS, 2017.