A Deep Learning System for Service Fault Prediction on Optical Transport Networks

Pengyun Wang[*] Hanling Yi[*] Hong Cheng[†] Lujia Pan[*]

Abstract

As the society is increasingly characterized by an explosion of digital traffic, maintenance of the network infrastructure is becoming ever more important. Frequently, the maintenance requires a solution that does not only respond to faulty incidents but also predicts and prevents them from occurring. In this study, we design a deep learning system for online fault prediction for services on Optical Transport Networks (OTN), which is an efficient infrastructure to transport data for telecommunications and service provider networks. This system exploits heterogeneous network data, and realizes automatic offline training and daily online service fault prediction. We show that the combination of time series features and network topology features can provide a considerable improvement in prediction accuracy. In addition, we deal with label uncertainty using advances in deep learning techniques to make further improvement. The system has been deployed on three OTN networks, with each serving millions of users. Experimental evaluation on the deployed system on one OTN network shows that the proposed model achieves a "precision" of 85.1% and a "recall" of 73.2%.

1 Introduction

Dealing with an explosion of digital traffic driven by multimedia services, mobile applications, social media, VoIP, and cloud computing, Optical Transport Network (OTN) is an efficient infrastructure to transport data for telecommunications industry and service provider networks. It wraps each client payload transparently into a signal for transport across optical networks. Each client signal can contain different traffic types, including Ethernet, storage, digital video, as well as Synchronous Optical Networking/Synchronous Digital Hierarchy.

There are over one thousand OTN networks across the world. Take the OTN network studied in this work as an example, it serves 1661 client signals and can support millions of users accessing Internet, in addition to performing other tasks. The typical capacity of a client signal can be 10, 40, or 100 GB/s, with 100 GB/s being the mainstream nowadays. If the traffic in a 100 GB/s client signal is purely for mobile Internet service, it can support up to tens of thousands of users. We refer to a client signal as a *service* in the rest of the paper for simplicity.

Due to long travel distance and multiple stages involved in the transportation of a service, it is subject to fault, affecting all the digital traffic therein. A fault occurs when a service performance index drops below a configured threshold. It is recorded as an alarm event by the network management. Typically, thousands of incidents of service faults occur in OTN networks every day across the world, potentially affecting millions of users. Around 60% of these incidents are associated with optical channel deterioration or optical module deterioration. If they can be predicted and prevented beforehand, it can provide considerable benefits to both the network operators and users.

However, the current maintenance scheme is mostly responsive, with mitigation measures being deployed in response to the occurrence of faults; meanwhile, end users may experience service interruption throughout the time-consuming troubleshooting process, which may range from tens to hundreds of minutes. For this reason, there is a strong demand of a solution to predict and prevent faults from happening. This problem can be formulated as a binary classification problem on whether a service is going to fault in the near future. Although seemingly a simple problem, it is non-trivial because several challenges exist:

▷ **Heterogeneous data sources**. We collect both key performance index of services in the form of time series and network topology data. Time series data alone cannot provide enough discriminative features, as the network topology also provides useful information (in terms of the path a service traverses) for fault prediction. How to extract discriminative features from both spatial and temporal data and incorporate them into the classification model is a challenge.

▷ **Cost effectiveness**. To provide a cost-effective

[*]Noah Ark's Lab, Huawei Technologies (wangpengyun@huawei.com, yi.hanling@huawei.com, panlujia@huawei.com)

[†]Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong (hcheng@se.cuhk.edu.hk)

solution, the proposed system needs to be able to perform model training without requiring too many human efforts in labeling training instances. This is due to the consideration that different OTN networks may have different discriminative patterns and that the network dynamics may change the patterns over time. Therefore, an automatic instance creation and labeling method is needed in order to achieve a cost-effective prediction system.

▷ **Instance label uncertainty**. The third challenge is common for classification problems, but it is particularly severe when instances are labeled in an automatic instead of manual manner. Label uncertainty has two aspects: 1) instances may be labeled incorrectly, resulting in noisy label problems; and 2) some instances may be unlabeled. An effective semi-supervised learning technique is needed to deal with the label uncertainty.

Recognizing the needs and challenges, we exploit the availability of network data, advances in machine learning, and domain expertise, and develop a deep learning system for online service fault prediction. Specifically, the system performs online classification to predict whether each service is going to fault in the near future. Combining with fault localization methods, the prediction system can potentially reduce the number of incidents of service faults considerably. Contributions of this study are summarized as follows.

▷ **A deep learning based service fault prediction model**. We propose a deep learning based model that can efficiently work on spatial-temporal data. Deployed in a metropolitan OTN network, this model achieves a "precision" of 85.1% and "recall" of 73.2%[1], considerably outperforming alternatives such as Random Forest (RF). To the best of our knowledge, the proposed deep learning model is the first of its kind reporting the use of deep learning techniques for online fault prediction for services on OTN networks.

▷ **Effective feature extraction based on heterogeneous sources of data.** To fully exploit the spatial-temporal nature of the network data, we construct three categories of features: *path embedding features*, *historical KPI features* and *historical alarm features*. Specifically, network embedding technique is used to extract spatial features and Temporal Convolutional Network (TCN) is employed to extract temporal features. These features are complementary to each other in the prediction model.

▷ **Automatic instance labeling and semi-supervised learning.** Based on service alarm data, we propose a labeling scheme in a "conservative" way

[1]Here "precision" and "recall" refer to TRR and FPR defined in Section 5.1, respectively.

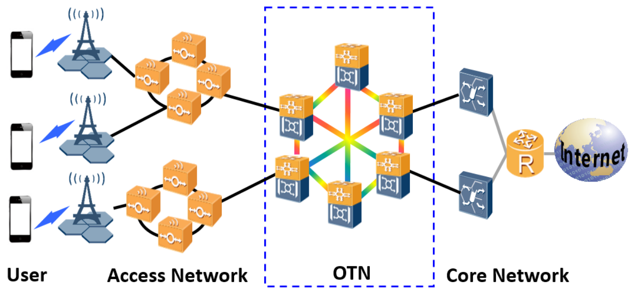

Figure 1: OTN in the context of mobile Internet service.

and create both labeled and unlabeled instances. To deal with label uncertainty, we use Neural Graph Machine (NGM) [3], a graph-based semi-supervised learning method, to facilitate and regularize the training of the deep learning model.

2 Data

Our study is based on a production OTN system of mobile network in a metropolitan area serving around 4.5 million users. We use this network to motivate the design of our solution, yet our methodology is applicable to other OTN systems for service fault prediction. In this section, we first provide a general overview of the OTN architecture, and then describe the data format for our analysis.

2.1 OTN Architecture An OTN is composed of a set of optical network elements (ONEs) connected by optical fibre links. It is an important transmission part of a mobile network or a fixed network. Figure 1 shows a simplified mobile network architecture, where the OTN is responsible for transporting signals between Access Network and Core Network, both of which are essential for providing Internet access for users in a mobile network. An OTN can be modeled as a directed graph, denoted as $G = (V, E)$, where V represents the set of ONEs and $E \subseteq V \times V$ corresponds to the optical fibre links.

A service refers to a continuous stream of digital traffic, which is added to the transport network at one node and dropped off at another to reach its destination. Typically there are thousands of services transporting on one OTN system. We use s to denote a service, and $\mathcal{S}$ to denote the set of services transporting on an OTN G. The sequence of nodes a service traverses is called its path, denoted as $(v_1, v_2, \ldots, v_n)$, where $v_i \in V$ for $i = 1, 2, \ldots, n$ and $(v_i, v_{i+1}) \in E$ for $i = 1, 2, \ldots, n-1$. We use $\mathcal{P}$ to denote the set of all paths in G, and $s(\mathcal{P})$ to denote the path of service s.

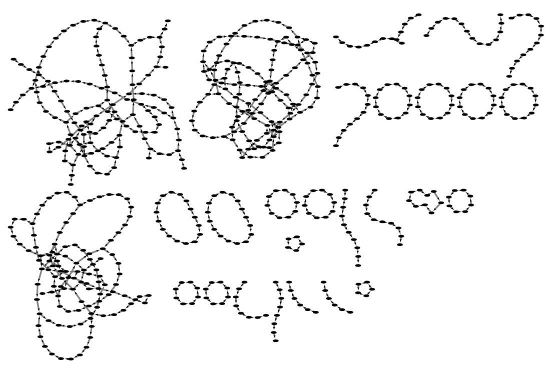

Figure 2: Topology of the studied OTN network.

2.2 Data Collection The data we collected can be classified into three categories: network configuration, key performance indicator (KPI) and alarm data. The network configuration information was obtained from a network management system. We collected the KPI and alarm data by probes in our OTN network from February 1 to April 25, 2018, and from June 15 to August 19, 2018, for a total of 149 days. The data from the first period is used for training our model and the data from the second period for testing. In the following, we describe these three categories of data and their statistics.

Network configuration. Network configuration specifies the path of each service, which allows us to establish the OTN topology. Although the network configuration may vary in the long run, it does not change often in the short term and therefore is considered stationary. Figure 2 displays the topology of the studied OTN network, in which there are a number of disconnected sets of nodes. Each set of nodes is responsible for connecting users in one part of the metropolitan area with the core network. There are 2030 nodes in the whole network, and 1661 services transporting on it. The number of distinct paths supporting these services is 853. On average, a path carries about two services. The average length of a path is about 9.9. All services share at least one node with other services. These statistics imply substantial path overlap among services.

KPI data. KPI data contains information measuring the system's performance. There are different kinds of KPI data, among which we select the *received optical power* (ROP) and *bit error ratio* (BER) for modeling after discussing with domain experts. KPI data are available at nodes where services are dropped off. Each service has separate KPI data even when multiple services are dropped off at the same node. KPI data are collected at a 15-minute interval. At time t, we denote the collected KPI data for service s as a two-dimensional vector measuring ROP and BER as follows:

$$x_t^s = [x_t^s(1), x_t^s(2)]. \tag{2.1}$$

The KPI data of a service s form a multivariate time series over time, and is denoted as $(x_1^s, x_2^s, \ldots, x_T^s)$ over time stamps $1, 2, \ldots, T$. For each service on each day, we collect 96 KPI data points, and in total we have $23,758,944$ KPI data points.

Alarm data. When the service quality declines below a predetermined threshold, an alarm is triggered. An alarm record contains the alarm occurrence time, duration and alarm type. Since the fault prediction for services is performed on a daily basis in this study, we use $a_i^s \in \{0, 1\}$ to indicate whether an alarm occurs on day i for service s. After preprocessing, there are a total of $1,262$ alarms recorded during the observed period. 96% of the days (out of 149 days) have alarms recorded, the average number of alarms in a day is 8.46, and the maximum number of alarms in a day is 46. Among all the services running in the network, 22% of them (out of 1661 services) have alarms, the average number of alarms for a service is 0.75, and the maximum number of alarms for a service is 18 for the whole period.

2.3 Instance Creation We create instances for each service on a daily basis. For example, instance s_i is created for service s on day i based on the network topology, the observed KPI data and alarm data as follows:

- Instance s_i takes $s(\mathcal{P})$, the path of service s, into consideration.
- Instance s_i considers the KPI time series of service s within a window of 7 days before day i, i.e.,

  $$\{x_t^s : \text{time } t \text{ falls within days } [i-7, i]\}. \tag{2.2}$$

  As a result, the observed KPI time series in instance s_i contains 672 data points in a 2-dimensional space. Note that if there are NaN values in the time series, s_i will not be included in our instance collection.
- Instance s_i also takes the sequence of historical alarm data of service s within the last 30 days before day i, i.e.,

  $$\{a_j^s : i - 30 \leq j \leq i - 1\}. \tag{2.3}$$

Using the above instance creation method, we created $200,731$ instances from all services, among which $118,525$ instances are created from the training period and $82,206$ from the testing period.

2.4 Instance Labeling For historical instances, we add labels to them according to the following criteria:

- If an alarm is triggered on day $i+1$ for service s, we label instance s_i as 1, i.e., service fault;
- If no alarm is triggered in the vicinity of day i for service s, we label instance s_i as 0, i.e., no service fault. In our experiments, we define the *vicinity* as 15 days before and after day i;
- All other instances not satisfying the above two conditions remain unlabeled.

According to the above criteria, we label a service as *faulty* in a "conservative" way, i.e., only an alarm is triggered immediately after day i, instance s_i is labeled as faulty. For an instance with no alarm triggered 15 days before and after day i, it is safe to conclude that the instance does not lead to any service fault. For other instances which have some alarms triggered in a window of 15 days but not on the immediate following day, it is hard to conclude whether they trigger any service fault or not. So we would rather leave them as unlabeled to reduce label uncertainty. These unlabeled instances can be incorporated in the learning process via semi-supervised learning techniques.

Based on the above labeling method, we report the number of instances in class 0, class 1 and unlabeled instances in Table 1. In particular, the ratio of class 0 to class 1 instances is about 210 : 1. Thus the class labels are highly imbalanced.

Table 1: The number and percentage of instances for different classes in the training and testing periods.

	Class 0	Class 1	Unlabeled
Training	$100,543$ (84.8%)	412 (0.3%)	$17,569$ (14.8%)
Testing	$68,716$ (83.6%)	392 (0.5%)	$13,097$ (15.9%)

3 Problem Formulation

Our study aims to perform fault prediction for services on OTN based on the collected data. It is formulated as a classification problem where the outcome variable is whether a service will have service fault in the near future, between day i and day $i+p$. We perform the prediction on a daily basis. On day i, the instance of service s, s_i is created based on the above-mentioned method. We aim to build a classifier denoted as a function f, which takes s_i as input and makes service fault prediction on the OTN.

4 Feature Extraction and Model Learning

4.1 Solution Overview To build a classifier for service fault prediction, we need to extract discriminative features from the collected data. To exploit the spatial-temporal nature of the network data, we define three types of features, denoted as z_1, z_2, z_3 respectively, that are complementary to each other. The first type of features, called *path embedding features*, is extracted from the path a service traverses by the embedding technique. The second type of features, called *historical KPI features*, is extracted from the KPI time series by a Temporal Convolutional Network (TCN) [1]. The third type of features, called *historical alarm features*, is extracted from the historical alarm data as in Eq. (2.3). Specifically, for an instance s_i, its historical alarm feature is the number of days with alarms in the previous 30 days:

$$z_3(s_i) = \sum_{i-30 \le j \le i-1} a_j^s. \tag{4.4}$$

This feature reflects how inclined a service is to fault in the recent history.

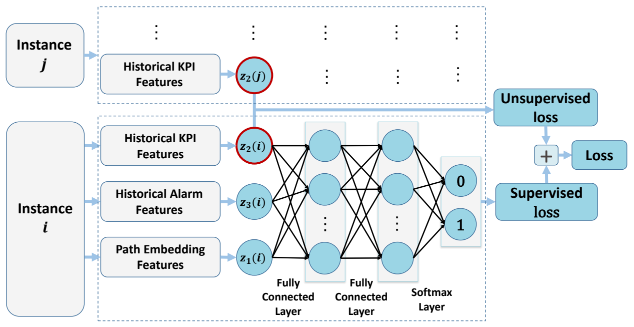

Figure 3: Solution overview.

Figure 3 gives an overview of our solution. During prediction, the concatenation of the extracted features z_1, z_2, z_3 are passed through a few fully connected layers to produce a score, as illustrated by instance i in the figure. In order to efficiently train the model, Neural Graph Machine (NGM) is employed to utilize both labeled and unlabeled instances. Labeled instances, such as instance i, is used to construct a supervised loss; while both labeled and unlabeled instances, such as instance i and j, are used to form an unsupervised loss, by ensuring "similar" instances resulting in similar historical KPI features z_2. Together, the supervised and unsupervised losses constitute the training target. To further improve the performance, snapshot ensemble is employed in making predictions. In the following, we introduce these components in details.

4.2 Path Embedding Path embedding feature represents the service paths in a low-dimensional space and captures the path similarity from the network topology perspective. Intuitively, if two paths overlap with each other significantly, their embeddings should be "close" in the representation space. It is constructed using a two-step approach. The first step is to generate node embeddings, denoted by $U = \{u_i, i \in V\}$. We use the spectral embedding technique to generate node embeddings. It is based on the spectral decomposition of the graph Laplacian, and each dimension of the embedding space corresponds to an eigenvector of the Laplacian matrix [5]. We take the first 96 eigenvectors to establish U. In the second step, the path embedding for instance s_i is computed by averaging the embeddings of nodes on the path of service s, i.e.,

$$z_1(s_i) = \frac{1}{|s(\mathcal{P})|} \sum_{j \in s(\mathcal{P})} u_j, \tag{4.5}$$

where $|s(\mathcal{P})|$ denotes the number of nodes that service s traverses. Note that other node embedding methods, such as Node2Vec [7], have been investigated, but they are less effective.

The path embedding for each instance has the same dimension as the node embedding. It is a useful feature in predicting the service faults from the network topology perspective. Applying a distance function on the path embeddings of different services can capture the similarity of the paths they traverse.

4.3 Temporal Convolutional Network TCN is a generic convolutional architecture that is designed for sequence modeling tasks. It is used to extract historical KPI features z_2 in this study. The reason to use TCN instead of LSTM [9] is that the former performs better at capturing longer effective memory [1], as demonstrated in a subsequent comparison analysis in Section 5.

Figure 4 depicts the structure of a TCN. The basic elements in a TCN are *residual blocks*, each of which is parameterized by the kernel size k, the dialation rate d and the number of output channels c. In a residual block, dialated causal convolution is used to extract features from the inputs. For example, the first residual block in Figure 4 takes the historical KPI time series data ($\in \mathbb{R}^{672 \times 2}$) as input, and subsequently applies two identical dialated causal convolution layers to generate the first-level hidden representation ($\in \mathbb{R}^{672 \times c}$, where $c = 64$ for the first residual block). The j-th ($j = \{1, 2, \ldots, 672\}$) row of the hidden representation is the dialated causal convolution results from c different filters, each with a kernel size k and dialation rate d.

A stacking of several such residual blocks forms a TCN. Usually, the dialation rate d increases exponentially with the depth of the network, i.e., $d = 2^{i-1}$ in the i-th ($i = \{1, \ldots, n\}$) residual block. By carefully

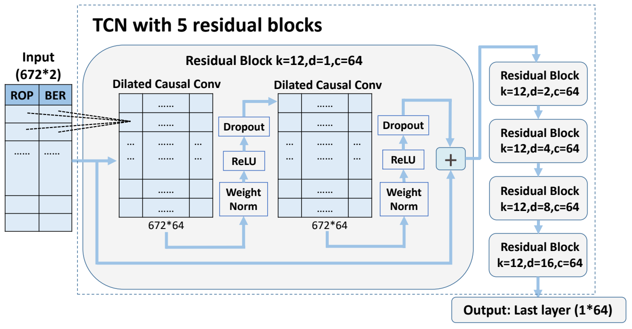
TCN with 5 residual blocks
Input (672*2)
ROP
BER
Residual Block k=12,d=1,c=64
Dilated Causal Conv
Dilated Causal Conv
672*64
672*64
Dropout
ReLU
Weight Norm
Dropout
ReLU
Weight Norm
+
Residual Block k=12,d=2,c=64
Residual Block k=12,d=4,c=64
Residual Block k=12,d=8,c=64
Residual Block k=12,d=16,c=64
Output: Last layer (1*64)

Figure 4: Overview of TCN.

designing the parameters in each residual block, TCN enables its outputs to represent features extracted from different ranges of inputs, i.e., achieve different receptive fields. In general, the receptive field of a TCN with n residual blocks and exponentially increasing dialation rate can be computed as in [4]:

$$F(n) = 1 + 2 * (k - 1) * (2^n - 1). \tag{4.6}$$

In our study, we stack 5 residual blocks in a TCN, i.e., $n = 5$. Each residual block has the kernel size $k = 12$, the number of channels $c = 64$, and the dialation rate at the i-th ($i = \{1, 2, 3, 4, 5\}$) residual block as $d = 2^{i-1}$. From Eq. (4.6), we know that this TCN has a receptive field of length 683, which is large enough to cover the historical KPI time series data. We take the last row of the hidden representation in the last residual block as the historical KPI features for the input instance.

4.4 Neural Graph Machine NGM is a graph-based semi-supervised learning technique that has been shown to outperform many existing methods on a wide range of tasks, with multiple forms of graph inputs and using different types of neural networks [3]. The main idea of NGM is to leverage the information encoded in the graph structure when training the neural network. In the following, we first introduce the graph we construct based on the instances, and then present the details of how the graph information is leveraged in NGM.

To exploit the spatial-temporal nature of the network data, we construct a fully connected graph with each instance being a node in the graph. Formally, we denote the graph as $\tilde{G} = (\tilde{V}, \tilde{E})$, where $\tilde{V}$ is the set of instances and $\tilde{E}$ is the set of edges that capture the spatial-temporal similarity among instances. Further, we denote the set of labeled and unlabeled nodes as $\tilde{V}_l$ and $\tilde{V}_u$, respectively. The weight on edge $(i, j) \in \tilde{E}$, denoted as $w_{i,j}$, is defined as the product of the spatial and temporal similarity between two instances i and j.

Specifically, $w_{i,j}$ is calculated as follows:

$$w_{i,j} = I(t(i) == t(j)) \times \frac{cos\,(z_1(i), z_1(j)) + 1}{2}, \tag{4.7}$$

where $I(\cdot)$ is an indicator function, $t(i)$ is the day on which instance i is collected, $z_1(i)$ is its path embedding, and $cos(z_1(i), z_1(j)) \in [-1, 1]$ computes the cosine similarity between $z_1(i)$ and $z_1(j)$. The first term represents temporal similarity, i.e., two instances are temporally similar if they are collected on the same day. Although it seems more reasonable to use a smooth function for measuring the temporal similarity, discontinuity in time series data caused by irregular network maintenance (human intervention) justifies the choice of the indicator function. The second term stands for the spatial similarity, and is computed as the shifted cosine similarity between path embedding features.

Based on the assumption that neighbouring nodes in the graph $\tilde{G}$ have similar historical KPI features, NGM incorporates the graph information by adding a regularization term in the loss function. Specifically, the loss function in NGM contains two terms: the supervised loss from the labeled instances and the unsupervised loss from the neighbouring instances (both labeled and unlabeled). Formally, the loss function in NGM can be expressed as:

$$L(\theta) = \sum_{i \in \tilde{V}_l} c(g_\theta(i), y_i) + \alpha \sum_{(i,j) \in \tilde{E}} w_{i,j} d(z_2(i), z_2(j)), \tag{4.8}$$

where $g_\theta(\cdot)$ denotes overall output of the deep learning algorithm parameterized by θ, $z_2(\cdot)$ denotes the historical KPI features extracted from TCN, y_i denotes the label of instance i, $c(\cdot, \cdot)$ denotes supervised loss function and $d(\cdot, \cdot)$ denotes unsupervised loss function. The parameter α controls the relative importance of supervised and unsupervised loss. With this new loss function in Eq. (4.8), both label information from labeled instances and graph information from neighbouring instances can be leveraged to update the neural network through back propagation. In our study, we set $\alpha = 0.1$, and use cross entropy and squared l_2 norm for supervised loss $c(\cdot, \cdot)$ and unsupervised loss $d(\cdot, \cdot)$, respectively.

4.5 Snapshot Ensemble To further improve the performance of deep learning algorithm, we implement snapshot ensemble [10], which is a technique that can ensemble multiple neural networks at no additional training cost.

In snapshot ensemble, the training process is split into M cycles, each of which starts with a large learning rate l_0. The learning rate is then annealed in each iteration according to a shifted cosine function as follows:

$$l(t) = \frac{l_0}{2}(1 + cos(\frac{\pi mod(t-1, \lceil N/M \rceil)}{\lceil N/M \rceil})), \tag{4.9}$$

where N is the total number of training iterations. At the end of each training cycle, the model is stored (i.e., take a snapshot). At the testing time, the predictions from $m \leq M$ snapshots are averaged to form the final output. Intuitively, the model will converge to a local minimum at the end of each cycle, and the large learning rate at the beginning of the cycle allows the model to escape from the current local minimum. Thus essentially snapshot ensemble improves the performance of neural network by exploiting the diversity in different local minima visited during each cycle. In our study, we set the initial learning rate $l_0 = 0.002$. For training, the Adam optimizer is used and the learning rate is annealed in each iteration according to Eq. (4.9). During the training process, we take $M = 4$ snapshots, and for testing we use the averaged output of $m = 4$ as the final result.

5 Evaluation

In this section, we evaluate our proposed method in service fault prediction. During the evaluation process, only instances from the testing period are used to compute the performance metrics.

5.1 Metrics In order to evaluate the effectiveness of the prediction model in real-world applications, we use several metrics similar to the ones used in disk failure prediction [16]. In principle, a desired model should be able to predict at least one of the instances collected within the last p days before the alarm to be positive; meanwhile, for the instances collected before the last p days, the model should not predict any of them to be positive.

To evaluate, we first split the testing period for each service into sub-periods in the following way: the first sub-period starts at the beginning of the testing period and ends on the day when the first alarm occurs; subsequently, the second sub-period starts on the day after the first period and ends on the day where the next alarm occurs; we repeat this process, and the last sub-period ends on the last day of the testing period. If a sub-period ends with an alarm, it is called a *faulty period*, otherwise it is called a *normal period*. As a result, in the testing period, we have 1612 normal periods and 605 faulty periods. The average length of normal periods and faulty periods are 53.38 and 12.86 days, respectively.

Then, we define fault prediction rate (FPR) and true risk rate (TRR) for evaluation. For each faulty period, a fault is correctly predicted (i.e., true positive) only when at least one of the instances collected within the last p days before the alarm is classified as positive. Otherwise, the fault is not correctly predicted (i.e., false negative). FPR is then defined as the fraction of faults that are successfully predicted to occur:

$$FPR = \frac{\#\text{true positives}}{\#\text{true positives} + \#\text{false negatives}}$$

Similarly, for each sub-period (both normal and faulty periods), a false positive occurs when any instance collected outside the last p days is predicted positive[2]. TRR measures the precision of fault prediction:

$$TRR = \frac{\#\text{true positives}}{\#\text{true positives} + \#\text{false positives}}$$

Note that there is an analogy between TRR (resp. FPR) and precision (resp. recall). There is usually a trade-off between TRR and FPR, which can be summarized using the TRR-FPR area under curve (TFAUC). In the following, we use TRR, FPR and TFAUC to evaluate the performance of the predictors. When computing the above metrics, the parameter p is set to 30 days, which is determined according to the domain knowledge that there might be a long deterioration period before a service fault happens. Intuitively, the smaller p is, the smaller TFAUC will be, as the window for prediction is shorter. In order to investigate the impact of parameter p, a sensitivity analysis is conducted and presented in subsection 5.3.

5.2 Results In this subsection, we first evaluate the classification performance by each category of features in service fault prediction. Then we report the overall performance of our proposed deep learning method, and show its superiority by comparing to other alternative classifiers.

Effects of features. To demonstrate the effect of each category of features, we employ Random Forest (RF) algorithm to train a classifier that takes a subset of $[z_1, z_2, z_3]$ as input. The reason to use RF instead of the proposed deep learning method to perform this task is two-fold: (a) RF is employed widespread, off-the-shelf and quick to yield results, and (b) it can provide a baseline to demonstrate the efficacy of the proposed deep learning method. When using a RF classifier, we manually extract features from each dimension of an instance's observed historical KPI time series data and concatenate them to construct z_1. The features we used include: (a) basic statistics, such as maximum, minimum, standard deviation, and slope; (b) fast Fourier transformation, and (c) wavelet transformation. These features are typically helpful in time series classification problems. To tackle the class imbalance issue in the training data set, we applied random downsampling and fixed the ratio between negative and positive instances in the training set to be 10:1. A RF classifier was built from the sample training set and subsequently employed to make predictions for the testing instances. For ease of comparison, we fix TRR at 0.85[3] and compare FPR, in addition to comparing TFAUC. To account for the randomness in the downsampling and training process, we run 10 trials for each experiment setup and report the mean and standard deviation for FPR and TFAUC, respectively.

The classification results are reported in Table 2. In the first column, the symbols in the parenthesis indicate how a classifier is trained. For example, RF(z_1, z_2) means that the classifier is trained with input features from z_1 and z_2. As shown in the first three rows of the table, we can observe that the inclusion of each additional category of features provides incremental benefit to the model performance. Specifically, if we compare RF(z_1, z_2, z_3) with RF(z_2), we can see that the RF classifier achieves a considerable improvement in FPR (5.2%) and TFAUC (4.3%) on average. This result verifies the effectiveness of each category of features proposed in this study.

Performance of the baseline. In order to establish a strong performance baseline, other classifiers have also been investigated, such as Support Vector Machine (SVM)[4], Logistic Regression (LR) and XGBoost, to conduct the experiment. The results are also reported in Table 2. We can observe that all of the three classifiers perform worse in terms of TFAUC, as compared to RF. In particular, on average XGBoost has a TFAUC of 0.835, which is the highest among the three classifiers, but still 1.6% lower than that of RF. Therefore, we consider the result of RF as our baseline for subsequent comparison. Note that deep learning based time series anomaly detection methods such as TCN and LSTM have also been investigated, but their outcomes are significantly below the above baselines and therefore they are not reported here.

Performance of the proposed method. In our method, batch gradient descent is employed to train the proposed deep learning model, and a random over-sampling scheme is used to generate the training batch. Specifically, in each iteration, 128 instances are sam-

[2]If p is larger than the length of the period, then there is no false positive.

[3]If we can not find a TRR that equals to 0.85, we set TRR to be the smallest number that is larger than 0.85.

[4]We use the RBF kernel for SVM.

Table 2: Results on feature effect and performance of different classifiers.

Algorithm	TRR	FPR	TFAUC
RF(z_2)	0.851	0.510 ± 0.063	0.808 ± 0.007
RF(z_1, z_2)	0.850	0.513 ± 0.082	0.836 ± 0.011
RF(z_1, z_2, z_3)	0.851	0.562 ± 0.086	0.851 ± 0.012
SVM(z_1, z_2, z_3)	0.865	0.074 ± 0.047	0.703 ± 0.010
LR(z_1, z_2, z_3)	0.851	0.527 ± 0.052	0.811 ± 0.014
XGBoost(z_1, z_2, z_3)	0.851	0.552 ± 0.081	0.835 ± 0.014
$DL_{LSTM}(\alpha = 0)$	0.851	0.432 ± 0.070	0.802 ± 0.016
$DL_{LSTM}(\alpha = 0.1)$	0.850	0.537 ± 0.032	0.834 ± 0.007
$DL_{TCN}(\alpha = 0)$	0.851	0.624 ± 0.023	0.858 ± 0.003
$DL_{TCN}(\alpha = 0.1)$	0.851	$\mathbf{0.732 \pm 0.015}$	$\mathbf{0.884 \pm 0.002}$

pled with the class ratio among positive, negative and unlabeled instances being fixed at $[0.1, 0.8, 0.1]$. This batch sampling scheme can effectively deal with learning problems associated with class imbalance. Recall that the Adam optimizer is used for training and the learning rate is annealed in each iteration according to Eq. (4.9) with an initial learning rate $l_0 = 0.002$.

Two experiments on our proposed method are conducted. In the first experiment, parameter α is set to 0. This indicates that we do not consider the unsupervised loss in the deep learning model. In the second experiment, α is set to 0.1 to incorporate the unsupervised loss based on the constructed instance graph. The results for both experiments are listed in the last two rows of Table 2. If we compare $DL_{TCN}(\alpha = 0)$ with RF(z_1, z_2, z_3), the former has a slightly better performance in TFAUC (0.7%), and a considerable improvement in FPR (6.2%). In addition, $DL_{TCN}(\alpha = 0.1)$ achieves improvement in each metric, compared with both $DL_{TCN}(\alpha = 0)$ and RF(z_1, z_2, z_3). In particular, $DL_{TCN}(\alpha = 0.1)$ increases FPR and TFAUC by 17% and 3.3%, respectively, in comparison to the RF baseline. This demonstrates the efficacy of the proposed method and the benefit of utilizing unlabeled data. Moreover, we note that the deep learning method has the smallest standard deviation in FPR and TFAUC among all the classifiers, which demonstrates its reliability and robustness.

Comparison between TCN and LSTM. It is of interest to conduct a comparison between TCN and LSTM to investigate which method can more effectively extract temporal features in our case. To make a fair comparison, we replace TCN in the proposed method with LSTM, and it is denoted as DL_{LSTM}. The number of hidden units in LSTM is set to be 64, which is the same as the number of channels in TCN. We stack multiple LSTM layers to conduct the experiment and find that 1 layer LSTM has the best performance. We apply the same experiment settings to DL_{LSTM}, and results are reported in Table 2. We can observe that the TFAUCs of DL_{LSTM} are around 5% less than the counterparts of DL_{TCN}, under settings with and without unsupervised loss. The potential reason for the less effectiveness of LSTM in our case is that, the length of the input time series (i.e., 672) is considerably long, which can be better handled by TCN as argued by Bai et al. [1].

5.3 Sensitivity Analysis

We investigate the impact of parameter p on the prediction performance. We report TFAUC for parameter p taking values in $[5, 10, 15, 20, 25, 30]$. This experiment is carried out for RF(z_1, z_2, z_3) and $DL_{TCN}(\alpha = 0.1)$. Figure 5 depicts the results. Overall speaking, the prediction performance increases as a function of parameter p, which is consistent with our expectation. In addition, it is worth noting that the proposed deep learning algorithm consistently outperforms RF algorithm across different predictive window sizes.

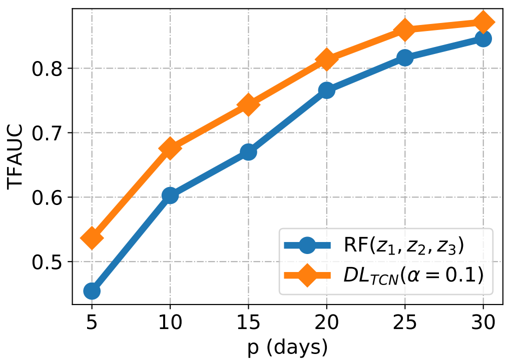

Figure 5: Sensitivity analysis for parameter p.

6 Related Work

Graph-based semi-supervised learning. Graph-based semi-supervised learning aims to leverage unlabeled data together with the graph structure to improve performance [2, 11, 18]. Depending on how the graph information is used, graph-based semi-supervised learning algorithms can be broadly categorized into convolution-based and regularization-based algorithms. The convolution-based algorithms, such as GCN [11], GraphSAGE [8] and GAT [15], try to learn node embedding by generalizing conventional convolution operator on arbitrarily structured graphs. The regularization-based algorithms incorporate the graph information in the model by explicitly adding a graph regularization term in the loss function [18]. Prominent examples include label propagation [20], manifold regularization [2] and planetoid [18]

The NGM [3] belongs to the regularization-based method and it can be viewed as a generalization of the aforementioned algorithms, in the sense that NGM can work with multiple neural network architectures (CNNs, RNNs) and on multiple forms of graph inputs (constructed or natural). In our proposed deep learning model, NGM works on a constructed graph to utilize the spatial-temporal similarity among training instances to perform semi-supervise learning, which we think is a novel approach to dealing with label uncertainty issues.
Machine learning methods for spatial-temporal data. In our problem, the data contain both topology (spatial) and time-series KPI (temporal) information. How to efficiently extract and utilize the spatial and temporal features is a key challenge. Various machine learning models have been proposed to deal with spatial-temporal data, including classical methods such as STARIMA [6], Gaussian Process [13], etc., and deep learning methods such as convLSTM [17], DCRNN [12], STGCN [19], etc. A detailed survey can be found in [14].

The existing deep-learning methods operate on the node level and work by implicitly capturing spatial features. In our case, however, we explicitly extract spatial features and concatenate them with temporal features to form the full feature vector, due to the difficulty of representing a service/path as a node in a constructed graph while maintaining the complex relations among service/path in the original graph. Experiment results demonstrate the effectiveness of such methods.

7 Conclusion

We design and implement a cost-effective deep learning system for service fault prediction on OTN. The system is based on the availability of network data, advances in machine learning techniques and expertise knowledge. It combines three categories of features (i.e., path embedding, historical KPI features and historical alarm features) to optimize the prediction accuracy. Furthermore, it exploits the advances in neural network techniques to account for label uncertainty to improve the accuracy. It has been deployed and tested on three OTN networks, with each serving millions of users. Evaluation shows that the proposed system can achieve a superior level of prediction performance.

References

[1] S. Bai, J. Z. Kolter, and V. Koltun, *An empirical evaluation of generic convolutional and recurrent networks for sequence modeling*, arXiv preprint arXiv:1803.01271, (2018).

[2] M. Belkin, P. Niyogi, and V. Sindhwani, *Manifold regularization: A geometric framework for learning from labeled and unlabeled examples*, Journal of machine learning research, 7 (2006), pp. 2399–2434.

[3] T. D. Bui, S. Ravi, and V. Ramavajjala, *Neural graph learning: Training neural networks using graphs*, in Proceedings of WSDM, 2018.

[4] Ceshine Lee, *Implementing Temporal Convolutional Networks*. Internet: `https://goo.gl/F3FSiL`, 2018.

[5] F. R. Chung and F. C. Graham, *Spectral graph theory*, no. 92, American Mathematical Soc., 1997.

[6] A. Cliff and J. K. Ord, *Space-time modelling with an application to regional forecasting*, Transactions of the Institute of British Geographers, (1975).

[7] A. Grover and J. Leskovec, *node2vec: Scalable feature learning for networks*, in SIGKDD, ACM, 2016, pp. 855–864.

[8] W. Hamilton, Z. Ying, and J. Leskovec, *Inductive representation learning on large graphs*, in NeurIPS, 2017, pp. 1024–1034.

[9] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural computation, 9 (1997).

[10] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, *Snapshot ensembles: Train 1, get m for free*, arXiv preprint arXiv:1704.00109, (2017).

[11] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, in ICLR, 2017.

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

[13] R. Senanayake, S. O'Callaghan, and F. Ramos, *Predicting spatio-temporal propagation of seasonal influenza using variational gaussian process regression*, in AAAI, 2016.

[14] X. Shi and D.-Y. Yeung, *Machine learning for spatiotemporal sequence forecasting: A survey*, arXiv preprint arXiv:1808.06865, (2018).

[15] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, *Graph attention networks*, arXiv preprint arXiv:1710.10903, 1 (2017).

[16] J. Xiao, Z. Xiong, S. Wu, Y. Yi, H. Jin, and K. Hu, *Disk failure prediction in data centers via online learning*, in ICPP, ACM, 2018.

[17] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, *Convolutional lstm network: A machine learning approach for precipitation nowcasting*, in NeurIPS, 2015.

[18] Z. Yang, W. W. Cohen, and R. Salakhutdinov, *Revisiting semi-supervised learning with graph embeddings*, arXiv preprint arXiv:1603.08861, (2016).

[19] B. Yu, H. Yin, and Z. Zhu, *Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting*, in IJCAI, 2018, pp. 3634–3640.

[20] X. Zhu, Z. Ghahramani, and J. D. Lafferty, *Semi-supervised learning using gaussian fields and harmonic functions*, in ICML, 2003.