NorBERT: NetwOrk Representations through BERT for Network Analysis & Management

Franck Le
IBM Research US
fle@us.ibm.com

Davis Wertheimer
IBM Research US
davis.wertheimer@ibm.com

Seraphin Calo
IBM Research US
scalo@us.ibm.com

Erich Nahum
IBM Research US
nahum@us.ibm.com

Abstract—Deep neural network models have been very successfully applied to Natural Language Processing (NLP) and Image based tasks. Their application to network analysis and management tasks is just recently being pursued. Our interest is in producing deep models that can be effectively generalized to perform well on multiple network tasks in different environments. A major challenge is that traditional deep models often rely on categorical features, but cannot handle unseen categorical values. One method for dealing with such problems is to learn contextual embeddings for categorical variables used by deep networks to improve their performance. In this paper, we adapt the NLP pre-training technique and associated deep model BERT to learn semantically meaningful numerical representations (embeddings) for Fully Qualified Domain Names (FQDNs) used in communication networks. We show through a series of experiments that such an approach can be used to generate models that maintain their effectiveness when applied to environments other than the one in which they were trained.

Index Terms—Network management, foundational models, contextual embeddings

I. INTRODUCTION

How to improve the generalization performance of deep models for network analysis and management tasks? As deep models are gradually being adopted for a wide range of network operations (e.g., packet classification [1]–[3], device classification [4]–[6], resource management [7], congestion control [8]–[11], routing [12]–[14], and scheduling policies identification [15]), this question becomes ever more salient. Although the generalization performance of deep models is a general problem, computer and telecommunication systems present unique challenges that exacerbate the issue. For example, local network configurations, geographic load balancing, or software updates can all cause deep models to overfit the training data, and perform poorly in the relevant outside environments.

Motivation: To illustrate these problems, we consider the task of detecting and classifying devices (e.g., manufacturer, operating system) present in a network from passive network monitoring. Because Internet of Things (IoT) devices are particularly vulnerable to attack, due to their poor security [16], network administrators may want to identify devices from specific manufacturers or types (e.g., cameras) to apply different policies (e.g., quarantine, network isolation) [4]. However, while connections to a server “server-54.us-east-1.update.xyz.com” may reveal the presence of an IoT device from the manufacturer xyz [5], that same device may instead connect to a different server “server-27.us-west-2.update.xyz.com” when deployed in a different enterprise network because of geographic load balancing. Network administrators would immediately recognize that the device is accessing the same service, but through a server in a different data center and location for better performance. However, to a deep neural network, the fully qualified domain name (FQDN) "server-27.us-west-2.update.xyz.com" is instead a new unseen value. Our deep device classifier may therefore perform poorly on the new enterprise network.

Similarly, devices running distinct software versions in different enterprise networks may cause deep models to perform well in the training environment, but poorly in new settings. As an example, devices of interest may connect to the server “fw-update.xyz.com” in the training setting, but devices from the same manufacturer in other enterprise networks may be running newer software versions, and instead connect to “fw-update2.xyz.com”. As in the prior example, the FQDN “fw-update2.xyz.com” is an entirely unseen value to our deep model, leading to poor performance in new settings.

To quantify the performance degradation, we train several Gated Recurrent Unit (GRU) models to predict the type (e.g., tv, refrigerator, hub, camera, doorbell) and manufacturer (e.g., Amazon, Apple, Belkin, Canary, D-link) of IoT devices, by analyzing their DNS traffic1. All the models exhibit poor generalization performance. For example, the device type classifier achieves a weighted average F1-score of 0.997 across all classes on the validation dataset, defined as 25% of the entire training set (traditional train-test split validation approach), but its performance drops to 0.592 when applied to a different publicly available network trace [4].

Research problem: Although the above examples demonstrate the problem in the context of device classification, the root cause is more general, and affects many other network analysis and management tasks. More specifically, traditional deep models’ approaches often rely on categorical features, but cannot handle unseen categorical values: performance degradation stems from an inability to learn the semantic meanings of new categorical values. Instead, if deep models were able to learn that the newly observed FQDN “server-27.us-west-2.update.xyz.com” is an example, devices may instead connect to a different server “server-27.us-west-2.update.xyz.com” when deployed in a different enterprise network because of geographic load balancing. Network administrators would immediately recognize that the device is accessing the same service, but through a server in a different data center and location for better performance. However, to a deep neural network, the fully qualified domain name (FQDN) "server-27.us-west-2.update.xyz.com" is instead a new unseen value. Our deep device classifier may therefore perform poorly on the new enterprise network.

Similarly, devices running distinct software versions in different enterprise networks may cause deep models to perform well in the training environment, but poorly in new settings. As an example, devices of interest may connect to the server “fw-update.xyz.com” in the training setting, but devices from the same manufacturer in other enterprise networks may be running newer software versions, and instead connect to “fw-update2.xyz.com”. As in the prior example, the FQDN “fw-update2.xyz.com” is an entirely unseen value to our deep model, leading to poor performance in new settings.

To quantify the performance degradation, we train several Gated Recurrent Unit (GRU) models to predict the type (e.g., tv, refrigerator, hub, camera, doorbell) and manufacturer (e.g., Amazon, Apple, Belkin, Canary, D-link) of IoT devices, by analyzing their DNS traffic1. All the models exhibit poor generalization performance. For example, the device type classifier achieves a weighted average F1-score of 0.997 across all classes on the validation dataset, defined as 25% of the entire training set (traditional train-test split validation approach), but its performance drops to 0.592 when applied to a different publicly available network trace [4].

Research problem: Although the above examples demonstrate the problem in the context of device classification, the root cause is more general, and affects many other network analysis and management tasks. More specifically, traditional deep models’ approaches often rely on categorical features, but cannot handle unseen categorical values: performance degradation stems from an inability to learn the semantic meanings of new categorical values. Instead, if deep models were able to learn that the newly observed FQDN “server-27.us-west-2.update.xyz.com” is an example, devices may instead connect to a different server “server-27.us-west-2.update.xyz.com” when deployed in a different enterprise network because of geographic load balancing. Network administrators would immediately recognize that the device is accessing the same service, but through a server in a different data center and location for better performance. However, to a deep neural network, the fully qualified domain name (FQDN) "server-27.us-west-2.update.xyz.com" is instead a new unseen value. Our deep device classifier may therefore perform poorly on the new enterprise network.

Similarly, devices running distinct software versions in different enterprise networks may cause deep models to perform well in the training environment, but poorly in new settings. As an example, devices of interest may connect to the server “fw-update.xyz.com” in the training setting, but devices from the same manufacturer in other enterprise networks may be running newer software versions, and instead connect to “fw-update2.xyz.com”. As in the prior example, the FQDN “fw-update2.xyz.com” is an entirely unseen value to our deep model, leading to poor performance in new settings.

To quantify the performance degradation, we train several Gated Recurrent Unit (GRU) models to predict the type (e.g., tv, refrigerator, hub, camera, doorbell) and manufacturer (e.g., Amazon, Apple, Belkin, Canary, D-link) of IoT devices, by analyzing their DNS traffic1. All the models exhibit poor generalization performance. For example, the device type classifier achieves a weighted average F1-score of 0.997 across all classes on the validation dataset, defined as 25% of the entire training set (traditional train-test split validation approach), but its performance drops to 0.592 when applied to a different publicly available network trace [4].

Research problem: Although the above examples demonstrate the problem in the context of device classification, the root cause is more general, and affects many other network analysis and management tasks. More specifically, traditional deep models’ approaches often rely on categorical features, but cannot handle unseen categorical values: performance degradation stems from an inability to learn the semantic meanings of new categorical values. Instead, if deep models were able to learn that the newly observed FQDN “server-27.us-west-2.update.xyz.com” is an example, devices may instead connect to a different server “server-27.us-west-2.update.xyz.com” when deployed in a different enterprise network because of geographic load balancing. Network administrators would immediately recognize that the device is accessing the same service, but through a server in a different data center and location for better performance. However, to a deep neural network, the fully qualified domain name (FQDN) "server-27.us-west-2.update.xyz.com" is instead a new unseen value. Our deep device classifier may therefore perform poorly on the new enterprise network.

Similarly, devices running distinct software versions in different enterprise networks may cause deep models to perform well in the training environment, but poorly in new settings. As an example, devices of interest may connect to the server “fw-update.xyz.com” in the training setting, but devices from the same manufacturer in other enterprise networks may be running newer software versions, and instead connect to “fw-update2.xyz.com”. As in the prior example, the FQDN “fw-update2.xyz.com” is an entirely unseen value to our deep model, leading to poor performance in new settings.

To quantify the performance degradation, we train several Gated Recurrent Unit (GRU) models to predict the type (e.g., tv, refrigerator, hub, camera, doorbell) and manufacturer (e.g., Amazon, Apple, Belkin, Canary, D-link) of IoT devices, by analyzing their DNS traffic1. All the models exhibit poor generalization performance. For example, the device type classifier achieves a weighted average F1-score of 0.997 across all classes on the validation dataset, defined as 25% of the entire training set (traditional train-test split validation approach), but its performance drops to 0.592 when applied to a different publicly available network trace [4].

Research problem: Although the above examples demonstrate the problem in the context of device classification, the root cause is more general, and affects many other network analysis and management tasks. More specifically, traditional deep models’ approaches often rely on categorical features, but cannot handle unseen categorical values: performance degradation stems from an inability to learn the semantic meanings of new categorical values. Instead, if deep models were able to learn that the newly observed FQDN “server-27.us-west-2.update.xyz.com” is an example, devices may instead connect to a different server “server-27.us-west-2.update.xyz.com” when deployed in a different enterprise network because of geographic load balancing. Network administrators would immediately recognize that the device is accessing the same service, but through a server in a different data center and location for better performance. However, to a deep neural network, the fully qualified domain name (FQDN) "server-27.us-west-2.update.xyz.com" is instead a new unseen value. Our deep device classifier may therefore perform poorly on the new enterprise network.
actually semantically similar to “server-54.us-east-1.update.xyz.com”, they may generalize better. Even in other tasks not relying on FQDNs, relevant features may still be categorical in general, and so the capability to learn semantic meaning of newly encountered values is essential to generalization performance. For example, for security vulnerability analysis, resource management (e.g., scaling of Virtualized Network Functions), or congestion control/routing, respectively, deep models may take into account the SSL cipher suite [17] of connections, the cloud instance types/flavors [18], and the QoS or Explicit Congestion Notification (ECN) codes [19] in IP packets, respectively, to determine the appropriate actions. But how can these models learn the semantic meanings of newly encountered values, not observed during training?

**Contributions:** In this paper, we first adapt a Natural Language Processing (NLP) pre-training technique and associated deep model called Bidirectional Encoder Representations from Transformers (BERT) [20], to learn semantically meaningful numerical representations, also called embeddings, for FQDNs. More specifically, we modify the tokenization step: BERT relies on WordPiece tokenization to identify basic word components – e.g., splitting “cars” into “car” and “s” – and determine the tokens for which to learn embeddings. Instead, we exploit the hierarchy of domains, which was designed to manage and resolve domains, to define the individual tokens.

Second, we demonstrate that applying the above algorithm allows embeddings to capture the semantic information behind FQDNs, and significantly improves deep models’ generalization performance. Continuing the previous example, the performance of the GRU-based IoT device type predictor improves from 0.592 to 0.965 when applied to the same independent dataset, but with embeddings learned through the proposed algorithm. We also compare performance when using context independent embeddings (e.g., GloVe [21]): with context independent embeddings, the performance of the downstream model on the independent dataset remains poor at 0.585. These experiments demonstrate the benefits of using BERT and deriving contextual embeddings for FQDNs.

Third, we further analyze the learned embeddings, and present (unsupervised) tasks network administrators could perform thanks to these novel embeddings (e.g., analysis of user behaviors). Finally, we discuss approaches for learning embeddings that capture semantic relationships for network categorical values, other than FQDNs; and present preliminary results. In particular, we found that the proposed approach can find meaningful relationships. For example, HTTP and HTTPS services are discovered to be semantically very similar; and strong TLS ciphersuites that rely on the same set of algorithms, and differ only in the keys’ lengths, are also discovered to be close. We conclude the paper with a discussion of future research directions.

**II. BACKGROUND: WORD EMBEDDINGS IN NLP**

This section provides a brief overview on the motivation, challenges, and techniques for creating word embeddings in NLP domains. Word embeddings are needed because deep neural networks cannot handle strings, only numerical vectors. As such, converting text to vectorial representations in high-dimensional latent spaces has been an active research area.

Although one hot encoding provides a simple numerical representation for categorical values, they do not capture any semantic information between words. Instead, every pair of words is equidistant in the latent space.

**Word2Vec:** In 2013, researchers developed Word2Vec [22], a popular technique to learn numerical representations for words that captures their semantic meaning via proximity. Two words that are close in meaning (e.g., “king” and “queen”) would have word embeddings (numerical vectors) that are close in distance (e.g., cosine similarity). To achieve this, Word2Vec relies on two neural network variants that calculate word embeddings based on the words’ context: Continuous Bag-of-Words (CBOW) predicts the current word based on the context; and Skip-gram instead predicts the closely related context words to an input word.

**GloVe:** In 2014, researchers presented Global Vectors for Word Representation (GloVe) [21], an extension of Word2Vec that capture global contextual information in a text corpus. More specifically, while Word2Vec ignores the frequency of co-occurrence of words (i.e., the number of times word \( j \) occurs in the context of word \( i \)), GloVe computes word embeddings taking co-occurrence into account. For this reason, GloVe embeddings are often considered a representation of the training corpus in lower dimensionality, and which reflects the word-word contextual relationships.

**BERT:** More recently, in 2018, researchers introduced Bidirectional Encoder Representations from Transformers (BERT) [20] to compute contextual embeddings. Word2Vec and GloVe compute context independent embeddings: given a word, its numerical representation is the same regardless of its position in a sentence, or the different meanings it may have. For example, in the sentence “Alice had a picnic by the river bank, and then went to the bank to open an account”, the two occurrences of the word “bank” have the same embedding. In contrast, with BERT, the two occurrences of the word “bank” have different embeddings. To achieve this, a tokenizer [23], [24] splits the text into words or subwords (e.g., “banks” would be decomposed into “bank” and “s”). Then, BERT is trained on two unsupervised tasks:

- **Masked LM:** A fraction of the words in the text are replaced with a \([\text{MASK}]\) token, and the model’s goal is to predict the original value of the masked words. However, compared to previous solutions which read the text input sequentially (e.g., left-to-right, or right-to-left), BERT applies the bidirectional training of a Transformer [25], an attention model [26], to language modelling. This allows the model to learn the context of a word based on all of its surroundings.

- **Next Sentence Prediction:** Given two sentences, the model is to predict whether the second sentence is the next sentence to the first one in the initial text, or if the second sentence has been randomly sampled.

BERT sets new state-of-the-art results for a wide range
of natural language processing tasks (e.g., text classification, question answering), and a large number of variants [27], [28] have since been proposed.

III. CONTEXTUAL EMBEDDINGS FOR DATA NETWORKS

Our goal consists of learning contextual embeddings (i.e., numerical representations that capture semantic relationships within a sequence) for categorical variables used by deep networks trained for network analysis and management tasks, to improve their performance.

A. What Categorical Variable?

In this paper, we focus on fully qualified domain names (FQDN), the complete, human-readable domain name (e.g., mascots.iitis.pl) for servers on the Internet. This is because FQDNs are semantically rich, and either explicitly, or implicitly, carry a lot of information. For example, in the FQDN “x.y.z”, the Top Level Domain (TLD) “z” distinguishes commercial enterprises (“.com”), government entities (“.gov”), educational institutions (“.edu”), and non-profit organizations (“.org”); or can also indicate the country where the domain is hosted: “.ca” for Canada, “.uk” the United Kingdom, etc. The Second Level Domain (SLD) “y” commonly indicates the organization that registered the domain name with a registrar. Subsequent domains may indicate services, department, or other structures in that organization.

Given their semantic richness, a large number of solutions rely on the analysis of FQDNs to detect botnets, phishing, spam, abnormal activities, and for classification [5], [29]–[34]. As such, we set out to investigate whether recent techniques developed in NLP can capture this information, and derive semantically meaningful embeddings for FQDNs.

B. How To Tokenize FQDNs?

Studies have revealed that tokenizers play a critical role in the semantic relations that NLP models learn from a corpus [35]. Because FQDNs have their own syntax, they demand a custom tokenization approach.

We propose to exploit the hierarchical structure of FQDNs. While FQDNs may simply appear as strings of numbers, letters and hyphens, with a maximal length of 255 characters, they actually embed a hierarchical structure, where the hierarchy levels of an FQDN are read from right to left, with full stops acting as delimiters between the different levels. As such, we truncate each FQDN to the k-th level, where k is a hyper-parameter that allows one to control the vocabulary size. More precisely, we replace every FQDN, FQDN, with the corresponding token: token = ".".join(FQDN.split(".”))[:min(k, len(FQDN.split(".”)))]

We leave the definition, and comparison, of more advanced tokenizers for FQDN (e.g., methods that may give larger weights to higher hierarchical levels) for future work.

C. How To Contextualize the Embeddings?

Neural language models have made great strides in language understanding through their ability to contextualize the tokens from a given input sequence. For example, in the sentence, “I visited the river bank, then went to the bank to make a withdrawal,” the token “bank” takes on two entirely separate meanings based on context: the first refers to a geographical feature, while the second to a financial institution. A straightforward mapping of single tokens to single embeddings will struggle to reflect this semantic dual nature. An encoder-based neural language model restores this ability, however, by remapping input embeddings to better reflect the context of the input sequence - e.g., the embedding for the first “bank” may get shifted closer to that for “shore”, while the second may move closer to “ATM”. By learning and remapping the input embeddings, the language model gains the ability to extract high-level semantic information from a given input sequence, which has proven useful for many downstream tasks.

In NorBERT, we propose a similar contextualization/remapping operation to capture the high-level semantics of network traffic sequences, using a small BERT-based model. Our training task is the standard Masked Language Modeling task, implemented as in Roberta [28]: given an input sequence, 15% of tokens are randomly chosen as training anchors. 80% of those anchors are replaced with a standard masking token, and the model must fill in the blanks from its learned vocabulary. A separate 10% of anchors are left unchanged, to encourage the model to preserve the meanings of unaltered non-anchor tokens, and the remaining 10% are replaced with a random token from the vocabulary, to encourage the model to contextualize all tokens in the sequence, rather than only those that are masked off. We use the Adam optimizer [36] and a linear annealing schedule for learning rate. Training sequence length ranges from 8 to 64, resampled every batch. Further hyperparameters and architecture details are provided in Table I.

IV. EXPERIMENTS

We conduct a number of experiments to evaluate the benefits of contextual embeddings for network analysis. We first present the datasets, then describe the network tasks, and finally discuss the results.

| Variable | Description | Value |
|----------|-------------|-------|
| D        | Dimension of embeddings | 128   |
| N        | Number of successive FQDNs in a sequence | [8,16,32,64] |
| k        | Truncation level in FQDN hierarchy (Sec. III-B) | 3     |
| h        | Number of BERT self-attention heads | 8     |
| b        | BERT batch size | 32    |
| l        | Number of attention layers | 4     |
| lr       | Learning rate | 0.001 |
| w        | Weight decay | 0.01  |

TABLE I
SUMMARY OF BERT HYPER-PARAMETERS AND VARIABLES
monitoring. More specifically, after observing device type (e.g., camera, doorbell, hub), and (2) their manufacturer and type of the IoT devices that originated them. We label the sequences with the instance as a sequence of successive DNS queries that a device queried, can a model correctly predict its device type and manufacturer? We trained successive DNS queries through GloVe (Section II).

The results are summarized in Table III. First, we observe that with randomly initialized embeddings, the performance of the models — both for predicting the device type, and the manufacturer — suffer significant drops from F-1 scores larger than 0.996 on the training dataset, to scores lower than 0.588 on the independent validation dataset. Those numbers confirm the model generalization problem for network analysis tasks (Section I). Second, we note that context independent embedding learnt through GloVe may improve the performance. For example, the model to predict the device’s manufacturer increased its F-1 score on the independent validation dataset from 0.588 to 0.726 thanks to context independent embedding embeddings, and their contextualizations to be processed by neural networks, we compute embeddings using three approaches. However, before describing each of them in detail, we present the training corpus, and the common preprocessing steps.

We preprocess each dataset by applying Zeek [37], a network analysis framework that parses packet captures and extracts fields of interest. We focus on the DNS traffic, and define an instance as a sequence of N successive FQDNs queried by a single device. We label the sequences with the type and manufacturer of the IoT devices that originated them.

### Summary of classifier hyper-parameters and variables

| Variable | Description | Value |
|----------|-------------|-------|
| D        | Dimension of embeddings | 128   |
| N        | Number of successive FQDNs in a sequence | 32    |
| k        | Truncation level in FQDN hierarchy (Sec. III-B) | 3     |
| h        | GRU Number of features in the hidden state | 64    |
| b        | GRU Batch size | 1024  |
| l        | GRU Number of recurrent layers | 2     |
| l_w      | Learning rate | 0.001 |

**TABLE II**

### A. Datasets

The datasets consist of packet capture (pcap) from three independent environments:

- **Smart Lab - UNSW**: A lab with 21 IoT devices, including cameras, switches and triggers, hubs, air quality sensors, electronics, healthcare devices, and light bulbs was assembled [4], and all traffic was collected, for two weeks in 2016, using `tcpdump`. The devices are labeled with the manufacturers and device types (e.g., Amazon Echo, Netatmo Welcome, etc.).

- **Smart Lab - Private**: Instrumented very similarly to the previous one, a private lab comprises 65 IoT devices from 31 vendors (e.g., Amazon, Apple), and 24 different types (e.g., TV, bulbs, camera, doorbell, hub, refrigerator, thermostat, etc.). The devices are also labeled with their manufacturer, and type. We focus on their traffic captured over 31 days in 2017.

- **Large enterprise network**: Traffic from an enterprise network located in North America, and which comprises slightly more than 3,000 hosts including servers, laptops, and phones, are captured at the border routers with the Internet for several 24 hour periods. The devices are not labeled.

We preprocess each dataset by applying Zeek [37], a network analysis framework that parses packet captures and extracts fields of interest. We focus on the DNS traffic, and define an instance as a sequence of N successive FQDNs queried by each unique device, from all three datasets (Section IV-A). Learning embeddings does not require labels, and as such, the trace from the large enterprise network can also be exploited. Then, we preprocess the training corpus by filtering out domains that are of little value. For example, we exclude FQDNs that end with "in-addr.arpa", which are mainly used for reverse DNS. This is similar to removing stop words in natural language processing. In addition, to reduce the vocabulary size, and allow a fair comparison across the methods, we truncate the FQDNs to the same level (Section III-B) with k set to 3. Finally, we implement and compare the following approaches for computing embeddings:

- **Random**: A simple and common approach consists in initializing the embedding to random values, which are then adjusted through backpropagation.

- **GloVe**: We learn context independent embeddings for the FQDNs through GloVe (Section II).

- **NorBERT**: We train an FQDN embedding contextualization model using the procedure outlined in Section III-C.

### C. Results

The results are summarized in Table III. First, we observe that with randomly initialized embeddings, the performance of the models — both for predicting the device type, and the manufacturer — suffer significant drops from F-1 scores larger than 0.996 on the training dataset, to scores lower than 0.588 on the independent validation dataset. Those numbers confirm the model generalization problem for network analysis tasks (Section I). Second, we note that context independent embedding learnt through GloVe may improve the performance. For example, the model to predict the device’s manufacturer increased its F-1 score on the independent validation dataset from 0.588 to 0.726 thanks to context independent embeddings. Most importantly, contextual embeddings computed through the proposed NorBERT approach allow models to outperform other approaches. The average weighted F-1 scores of the models to predict device type and manufacturer reach 0.965 and 0.906, respectively. These large improvements demonstrate the benefits of contextual embeddings, and their ability to increase the models’ generalization performance.

---

\(^{2}\)Methods have recently been developed for this problem (e.g., [39]–[42]), but we leave the identification of out of distribution instances for future work.
V. FURTHER ANALYSIS OF EMBEDDINGS

In this section, we further analyze the learned embeddings. For example, for a given FQDN, we can retrieve the nearest neighbor. In NLP, nearest neighbor search has been an important building block for real world applications such as text search and recommender systems. We show how such analysis in the context of FQDNs can reveal new insights, and enable new unsupervised analysis tasks.

A. Nearest Neighbor

To verify the quality of word embeddings, NLP studies often look at words and their most similar counterparts. For example, the word “France” should intuitively be similar to “Spain”, or some other countries. We first conduct a similar analysis, using the learned embeddings from our NorBERT model. We focus on the packet capture from the large enterprise network. Given a list of FQDNs, we retrieve their nearest neighbor using the cosine similarity to compute the distance between any pair of embeddings.

| FQDN               | Nearest Neighbor |
|--------------------|-----------------|
| www.theatlantic.com | www.wired.com    |
| www.theguardian.com | www.wired.com    |
| getpocket.com      | www.newyorker.com|
| www.gq.com         | www.buzzfeednews.com|
| www.fastcompany.com | www.wired.com    |
| www.harpersbazaar.com | www.wired.com    |
| www.theringer.com  | lifehacker.com   |
| www.wired.com      | www.nytimes.com  |
| fivethirtyeight.com | www.vox.com      |
| www.politico.com   | www.wired.com    |
| ctdl.windowsupdate.com | www.bing.com   |
| activity.windows.com | edge.microsoft.com |
| content.office.net | time.windows.com  |
| dscg.akamaiedge.net | dscj.akamaiedge.net |
| nvidia.github.io   | download.nvidia.com |
| desktop-#####.#####.com | win-#####.#####.com |
| laptop-#####.#####.com | laptop-#####.#####.com |

Table IV presents an excerpt of the results. We observe that the first set of FQDNs and their nearest neighbors do appear similar: They are American or English websites with articles in politics, foreign affairs, business, culture, or technology. Similarly, the next sets of FQDNs may have different second level domains (SLD) (e.g., windows, bing, windowsupdate), or third level domain, but they all appear to be owned by the same organization. For the last set of FQDNs, they all point to computers in the enterprise network. The second level domain has been anonymized with a fixed number of characters “#”. Network administrators in that enterprise assign a default hostname to computers starting with either “desktop-”, “laptop-”, or “win-” and followed by a unique identifier, also anonymized. However, for many such computers, their nearest neighbor seems to be another computer from the same domain. These results confirm the embeddings capture semantic relationships between FQDNs, and the latest observation suggests that computers in that enterprise may form clusters, and could point to outliers.

B. Unsupervised Learning

Insights from the previous analysis (Section V-A) suggest that FQDNs with similar semantics may cluster together, and can help unearth patterns. As such, we explore and visualize the data through projection: we project the high dimensionality embeddings (128 dimensions) into a two-dimensional plane using t-distributed Stochastic Neighbor Embedding (t-
SNE) [43], a nonlinear dimensionality reduction algorithm. Figure 1 shows the result. A point in the graph represents a single FQDN. We expect users’ computers to have similar embeddings. Any user’s computer that differs significantly may indicate an unexpected behavior (e.g., misconfiguration).

To detect any such endpoint, we distinguish FQDNs starting with the prefix “laptop-”, “desktop-”, and “win-” respectively, and ending with the second level domain of that enterprise with a different color, and the label 1, 2, and 3, respectively. We observe that the FQDNs starting with the prefix “laptop-”, “desktop-”, and “win-”, and ending with the second level domain of that enterprise, actually form a compact cluster with no outstanding outlier – a desirable property for network administrators. In contrast, some embeddings, in the upper left corner of the figure appear to protrude from the main cluster. Table V presents an excerpt of those FQDNs. Their names indicate a majority of them correspond to mirror servers, and repositories. Intuitively, those servers behave differently from users’ computers. These results confirm that exploration of FQDN embeddings can reveal new patterns and insights.

In addition to visual inspection, a more systematic approach to explore the data and discover patterns could consist in applying hierarchical clustering. However, due to space limitations, we leave this for future work.

C. From Token to Sequence

A major characteristic of contextual embeddings lies in their ability to derive numerical representations whose values can change based on the context. As such, analyzing sequences of FQDNs, rather than standalone FQDNs, would further exploit their strength. For example, one can passively monitor the DNS queries submitted by each host, create sequences of N FQDNs, and compute their embeddings. Then, analysis of the sequence embeddings, either through projection and visual inspection, or through hierarchical clustering, could reveal abnormal user behaviors.

To illustrate this, we consider a different 24 hour trace from the large enterprise network than the one that was used in Section IV-B. We define a sequence as the list of N successive FQDNs queried by a single host. This sequence is fed into NorBERT to compute the contextualized embedding for each FQDN. We then derive the sequence embedding as the mean of the N contextual embeddings. Finally, we again project the sequence embeddings into a two-dimensional plane using t-distributed Stochastic Neighbor Embedding (t-SNE).

Figure 2 shows the result. Each dot represents the embedding of a sequence of N FQDNs queried by a single host. If the hostname of that endpoint includes specific sub-strings indicating the host is a desktop or a laptop, the dot is marked with a value of 0 or 1, respectively. All other hosts are marked with a value of 2. Network administrators can then look more closely at the outliers. An outlier indicates a sequence of FQDN that is considerably different from those typically observed by other members of a class (e.g., laptop, desktop). Note that compared to the uncontextualized FQDN embeddings in Fig. 1, the contextualized host sequence embeddings are far more distinctively clustered and semantically rich, indicating that NorBERT truly is contextualizing meaningfully, and performing non-trivial network traffic analysis, despite the lack of any human-annotated labels or guidance during the training process.

VI. GOING BEYOND FQDN

This section discusses approaches to, and presents preliminary results on, learning contextual embeddings for categorical variables other than FQDN. While previous sections have shown how embeddings for FQDN can be computed, and then used for different network analysis tasks, one may question: Can we compute embeddings for other network categorical variables? For example, as the destination port represents the server application, destination port 80 (HTTP [44]) may be
considered semantically closer to port 5684 (CoAP [45]) than port 22 (SSH [46]). Similarly, as the ciphersuite field in the TLS handshake [47] represents the set of algorithms used to secure connections, some ciphersuites may be considered semantically closer (e.g., strong versus weak). Can we learn embeddings for network categorical variables that capture such information?

One major challenge to learning embeddings consists in defining the context, and the tokens. For example, network traffic is often viewed as a sequence of bytes. In such a case, what would the delimiters and tokens be? In natural language processing, white space and punctuation are commonly used as delimiters to extract tokens. What are the equivalent of white space and punctuation for network data?

To address those challenges, we propose to consider network data at a higher level than sequences of bytes. We argue that bytes are in reality mainly the implementation of protocols, and the exchange of information, between clients and servers. As such, network data can be treated as sequences of (protocol field, protocol value) pairs. As illustrated in Table VI, network traffic analyzers (e.g., [48]) rely on protocol parsers to extract protocol fields and values, and present to users who can then analyze the data more easily. A token can then be either a protocol field, or protocol value; and the equivalent of a sentence from natural language processing can be a sequence of tokens as depicted in Table VII.

Experiments: To evaluate what information network embeddings might learn using such token definition, and what context might capture, we focus on the “Smart Lab - UNSW” trace, and extract a number of protocol fields (e.g., ip.src_host, ip.dst_host, udp.srcport, udp.dstport, tcp.srcport, tcp.dstport, icmp.type, dns.qry.name, http.server, http.request.method, http.request.uri, http.host, ssl.handshake.ciphersuite, http.user_agent, etc.) using tshark. We create sequences of N tokens, and learn contextual embeddings through the procedure described in Section III-C.

Preliminary results: Similarly to the analysis in Section V-A, in order to verify the quality of the learned embeddings, given a token, we retrieve its nearest neighbor. For example, considering the token “80”, the closest neighbor is “443”. Destination port 80 represents HTTP, while destination port 443 represents HTTPS. Those results are therefore according to intuition. Next, considering the token “49199” which is the code for the ciphersuite “ECDHE + RSA authentication AES-128 GCM SHA-256”, its closest neighbor is token “49200” which is the code for ciphersuite “ECDHE + RSA authentication AES-256 GCM SHA-384”. Those two codes represent the same algorithms, and differ only in the keys’ lengths. As such, the proximity of these codes is again according to intuition. In summary, the preliminary results are promising, indicating that the embeddings can learn meaningful relationships for different network categorical variables, though downstream evaluation tasks are left to future work.

VIII. ACKNOWLEDGEMENT

We thank Mudhakar Srivatsa, Raghu Ganti, and the MASCOTS reviewers for their insightful feedback.

REFERENCES

[1] S. Rezaei and X. Liu, “Deep Learning for Encrypted Traffic Classification: An Overview,” IEEE Communications Magazine, vol. 57, no. 5, pp. 76–81, 2019.
[2] M. Lotfollahi, M. Jafari Siavoshani, R. Shirali Hossein Zade, and M. Saberian, “Deep packet: A novel approach for encrypted traffic classification using deep learning,” Soft Computing, vol. 24, no. 3, pp. 1999–2012, 2020.
[3] E. Liang, H. Zhu, X. Jin, and I. Stoica, “Neural Packet Classification,” in Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM, 2019.
[4] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijewardena, A. Vishwanath, and V. Sivaraman, “Characterizing and Classifying IoT Traffic in Smart Cities and Campuses,” in IEEE Infocom Workshop Smart Cities and Urban Computing, 2017.
