Learning IP Network Representations

Mingda Li
University of California
Los Angeles, CA
lmd1993@g.ucla.edu

Cristian Lumezanu
NEC Laboratories America
Princeton, NJ
lume@nec-labs.com

Bo Zong
NEC Laboratories America
Princeton, NJ
bzong@nec-labs.com

Haifeng Chen
NEC Laboratories America
Princeton, NJ
haifeng@nec-labs.com

This is a slightly revised version of the paper "Deep Learning IP Network Representations", initially presented at the SIGCOMM'18 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks (Big-DAMA).

ABSTRACT

We present DIP, a deep learning based framework to learn structural properties of the Internet, such as node clustering or distance between nodes. Existing embedding-based approaches use linear algorithms on a single source of data, such as latency or hop count information, to approximate the position of a node in the Internet. In contrast, DIP computes low-dimensional representations of nodes that preserve structural properties and non-linear relationships across multiple, heterogeneous sources of structural information, such as IP, routing, and distance information. Using a large real-world data set, we show that DIP learns representations that preserve the real-world clustering of the associated nodes and predicts distance between them more than 30% better than a mean-based approach. Furthermore, DIP accurately imputes hop count distance to unknown hosts (*i.e.*, not used in training) given only their IP addresses and routable prefixes. Our framework is extensible to new data sources and applicable to a wide range of problems in network monitoring and security.

CCS CONCEPTS

• **Networks → Network structure**; *Network security*; • **Computing methodologies → Machine learning**;

KEYWORDS

deep learning, network, structure, neural networks, embedding

1 INTRODUCTION

The ability to map, analyze, and understand the structure of the Internet helps network management and operations by revealing opportunities for improvement or potential design flaws. For example, accurately predicting the closest server is critical in peer selection and load balancing [15]. Knowing how remote IPs are clustered can help diagnose anomalous events such as spoofing attacks [10]. A holistic view of the network and its structure is essential towards achieving the vision of self-driving networks [9].

Most previous attempts to uncover the Internet structure relying on active probing from multiple vantage points using tools such as *traceroute* and *ping* [14, 21]. Such techniques provide fine-grained introspection (*i.e.*, can measure specific properties in specific parts of the network, such as the latency of a path) but pose a significant cost in terms of network overhead.

In contrast, embedding-based approaches use fewer, strategic measurements [4, 6] or passive observations on network traffic [8] to learn vector representations for the network end-hosts in a low-dimensional space. The representations approximate the positions of hosts in the Internet and are used to recover structural network properties, such as distance between nodes or clustering of nodes. However, the complexity of the Internet and the sparse input data make it difficult to compute accurate representations. Oftentimes, embedding approaches rely on additional data sources, which cannot be easily used in the embedding process, to refine and tune the final embeddings. For example, several embedding methods build representations based on distance-based metrics, such as latency or hop count, and then refine (or even replace) the final representations using additional probes or static information such as AS membership or routing information [4, 7].

The emergence of deep learning as a powerful tool to extract hidden features in data calls for revisiting the problem of learning network representations through embedding. In particular, deep learning techniques provide two key benefits. First, they allow multiple heterogeneous sources of information as input, thereby identifying more accurately the relationships between multiple sources of data that jointly contribute to a specific structural property [11, 17, 23]. Second, deep neuron networks are extensible and can easily incorporate additional sources of information by attaching more neurons, network layers, or network branches [23]. One can start with a model trained on the original components and retrain it using only the newly added parts or data sources [5]. This makes it easier for network operators to deploy, apply, or update neural network based models.

We propose DIP, a deep learning based framework to learn the structure of the Internet. DIP is a ten-layer neural network[1] that computes a low-dimensional vector representation for any node[2] in the Internet *given only its IP address and routable prefix*. DIP

[1] To avoid confusion and unless explicitly stated otherwise, we use *network* or *Internet* to refer to the physical IP network and *neural network* to refer to the neural network we design to learn the structural properties of the physical network

[2] We use *node* or *(end-)host* to denote any computer connected to the Internet and assigned an IP address.

preserves both local and global structure: clustered nodes have similar representations and the distance between two representations approximates the hop count between the associated nodes.

We train our neural network using three heterogeneous data sets: hop count distances between Internet nodes, the 32 bits IP address and inter-domain routable prefix information for each node. A key insight to train DIP is to *first compute representations based on the IP and routing information*, thereby recovering structural information hidden in the IP values, and refine them using a distance-based optimization. As the size of the routable prefix varies by IP, we first normalize the IP and prefix data by representing an IPv4 address on 64, rather than, 32 bits. To capture structural information encoded in the IP address value, we feed the eight bytes of the normalized IP sequentially at each of the first eight layers of the neural network. We then use the last two layers to get the hop count matrix and optimize the embedding distance prediction. With a trained DIP, we can estimate distance between *any* two Internet hosts as long as we have their IPs, even if they are not part of the training data.

Results on large real-world data sets of hop counts between thousands of IP addresses and 95 geographically distributed servers show that we can predict hop count distance between known hosts (*i.e.*, whose IP address were used in the training) with an absolute error of around 2 hops and over 30% better than a mean-based method. We infer the distance between unknown IPs (*i.e.*, not appearing in training data) with a small loss in accuracy compared to known IPs. In addition, the representations learned by DIP preserve the real-world clustering of the associated hosts. The accuracy of our model increases when we increase the training data set.

While our results are preliminary, they offer us a glimpse of the power of deep learning in recovering structural properties of the Internet from sparse data. DIP is the first framework that can estimate accurately the distance to any Internet host given only its IP address and routable prefix without any distance data.

2 BACKGROUND AND RELATED WORK

What is structure? Many properties can make up the structure of the Internet: connectivity between IPs, routers, or networks; distance-based metrics such as hop count or latency; similarity-based metrics such as the set of one's neighbors in the connectivity graph; path-based properties such as the sequence of routers on a path. Here, we focus on two specific properties that define both the local and the global structure of a network: clustering of end-hosts and hop count based distance between end-hosts.

Network coordinate systems learn vector representations for participating nodes such that the position of the node in the embedding approximates its position in the Internet. Most coordinate systems build embeddings using a single source of structural data: latency measurements among nodes or to predetermined landmark servers [3, 4, 18, 19, 24] or hop count information from passive traffic observations [6, 7]. Latency and hop count data is often sparse and cannot always be accurately embedded in metric spaces. To overcome these issues, several approaches use out-of-band information, such as location [4] or routing [7] data, or perform active measurements [6] to impute the missing data and detect clusters or distances. Unlike them, we propose to train our embedding jointly using distances, routing information and host IP values, thereby learning hidden structural features encoded in a node's IPv4 address. With a trained model, we are able to embed and find the hop count to any IP, without the participation of its host.

Deep neural networks consist of multiple layers of interconnected neurons [13]. A neuron aggregates multiple input values using local weights and biases, applies an activation function, and produces one or more numerical values as output. Given a training task, one can define an objective function to evaluate the output of the entire neural network, *e.g.*, prediction error. Using gradient-based back-propagation algorithms to optimize the objective function [12], neural networks automatically tune the weights and biases of each neuron to achieve a better performance.

3 LEARNING NETWORK REPRESENTATIONS

DIP learns an embedding model that accurately reflects the structure of the Internet, *i.e.*, preserves node clustering and distances between nodes. The goal of learning is to minimize the prediction error for the distance between any two nodes. The learned model is defined by the structure of the neural network and the final values for the weights and biases of each neuron. Next, we describe the data used in learning and how we construct the neural network.

3.1 Data sources

IP addresses and routing information. The IP address of a host provides a coarse indication of the location of the host in the Internet. To make routing scalable and fast, IP addresses are assigned hierarchically and divided into a network (or routable) part and a host (or local) part. The routable part, usually expressed by an integer representing the number of bits (also called prefix), tells routers how to route the packet through the core of the Internet towards the destination network. Intuitively, IPs with the same routable prefix share a path towards them through the Internet core and are more likely to be close to each other.

Hop counts. The hop count between two hosts represents the number of routers on the default path between hosts. We use hop count, rather than latency, to measure the distance between two hosts, as it can be easily extracted from the TTL value of a network packet [10], without active measurements. In Section 5, we discuss how to extend the model using latency measurements. Our hop count matrix is asymmetric and very sparse; it does not contain hop counts between all IPs.

3.2 IP transformation

The key idea of our work is to use both local (IPs and routing information) and global (hop counts) structural information to guide the embedding of network nodes. By utilizing deep learning for embedding, we can identify and use hidden features encoded in the IP address of a given node. We perform several transformations on the input, guided by observations on real network data.

IP normalization. Because the routable information is tied to an IP address, we combine the IP and prefix values when feeding them to the neural network. To keep the size of the input constant and independent on the prefix size, we generate a *normalized IP address* for each regular IP. The process of normalization is depicted in Figure 2. We divide each IP into the network and the host parts. We pad the end of the network part and the beginning of the host

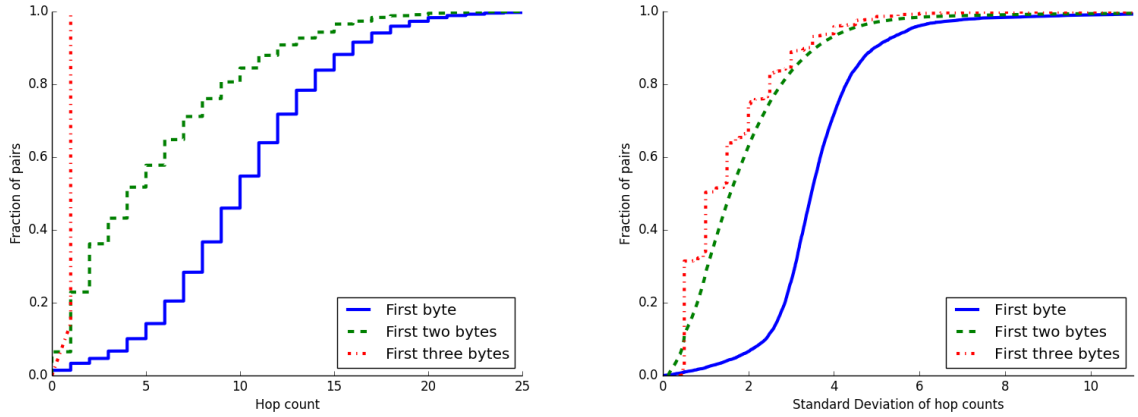

Figure 1: Cumulative distribution of (left) hop counts between pairs of host-server IPs that share the first, first two, or first three bytes, and (right) standard deviation of hop count distribution among groups of IPs sharing the first, first two, or first three bytes. The more similar two IPs are, the closer they are and the more similar their distances to the same third IP are.

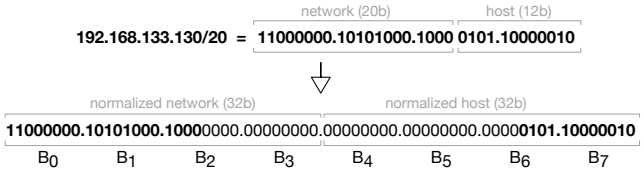

Figure 2: Generating a normalized IP address for 192.168.133.130/20.

part with zero to obtain two four-byte values. We concatenate the values and get the eight bytes normalized IP. Further, for easier processing, we represent each byte of the input in one-hot vector format (256 dimensions), *e.g.*, a one and the rest are 0s, where the 1's position is the value of the byte (0 to 255).

Sequential feeding. IP addresses are assigned hierarchically and encode structural information of the network. To better understand how the hierarchical assignment affects node clustering, we perform two experiments on a data set of hop counts between 95 geographically distributed servers and ten million IP addresses of end hosts. Section 4.1 describes the data in more detail.

First, we group all pairs of host-server IPs according to whether they share (within the pair) the first byte, first two bytes, or first three bytes. We show the all-to-all hop counts between pairs in each of the three groups in Figure 1(left). The more similar two IP addresses are, the closer they are in terms of number of hops. Second, we group separately hosts and servers according to whether they share the first one, two, or three bytes and generate the hop count distribution for each pair of host-server groups that share the same prefix. We present the standard deviation for each pair in Figure 1(right). The smaller the standard deviation is, the more similar the distances are. This means that the more similar two IPs are, the more likely they have the same hop count to another node.

As shown in Figure 1, an IP address can help learn node representations that capture the network structure. The more bytes of an IP address we know, the better we can constrain the representation we assign to it. In addition, the more significant bytes of an IP address have a higher influence on the position of the associated host relative to other hosts. Therefore, the key is to capture the sequential correlation among the bytes of an IP address.

3.3 Network construction

Driven by the insight gained in the previous section, we develop DIP, a deep neural network that computes vector representations of network hosts based on their IP addresses and the hop counts to other hosts. The design of DIP, depicted in figure 3, is similar to that of a recurrent neural network [16], where new data is processed in the context provided by previous data (*e.g.*, like processing natural language). We explain the details below. Even though the figure and our explanation refer to the input as one-hot vectors (*e.g.*, a normalized IP is represented as a vector of size 8x256=2,048), in reality the inputs are matrices (*i.e.*, the number of IP addresses times 2,048). Because our hop count data (see Section 4.1) is between separate end-hosts (sources) and servers (destinations) and because distances in the Internet are not always symmetric, we choose to feed the source and destination IPs separately in the neural network.

Intermediate IP representation. As mentioned earlier, to get the most out of the format and value of an IP address towards building a representative embedding for its host, we should treat each byte separately. The more significant bytes can provide a context for how to interpret the less significant bytes. Thus, we choose to input each byte of the normalized IP (a 256-dimension one-hot vector $B_{256\times1}^{i\in\{0,\ldots,7\}}$) separately at each layer of the network. The input of layer i is the concatenation of byte i with the output of the previous layer (except for the first layer). This

$$Input = \begin{cases} i = 0 & B_{256\times1}^{i=0} \\ i \in \{1,\ldots,7\} & concat(f_{d\times1}^{i-1}, B_{256\times1}^{i}) \end{cases} \quad (1)$$

where d is the dimension of the final IP representation and *concat* represents the vector concatenation operation.

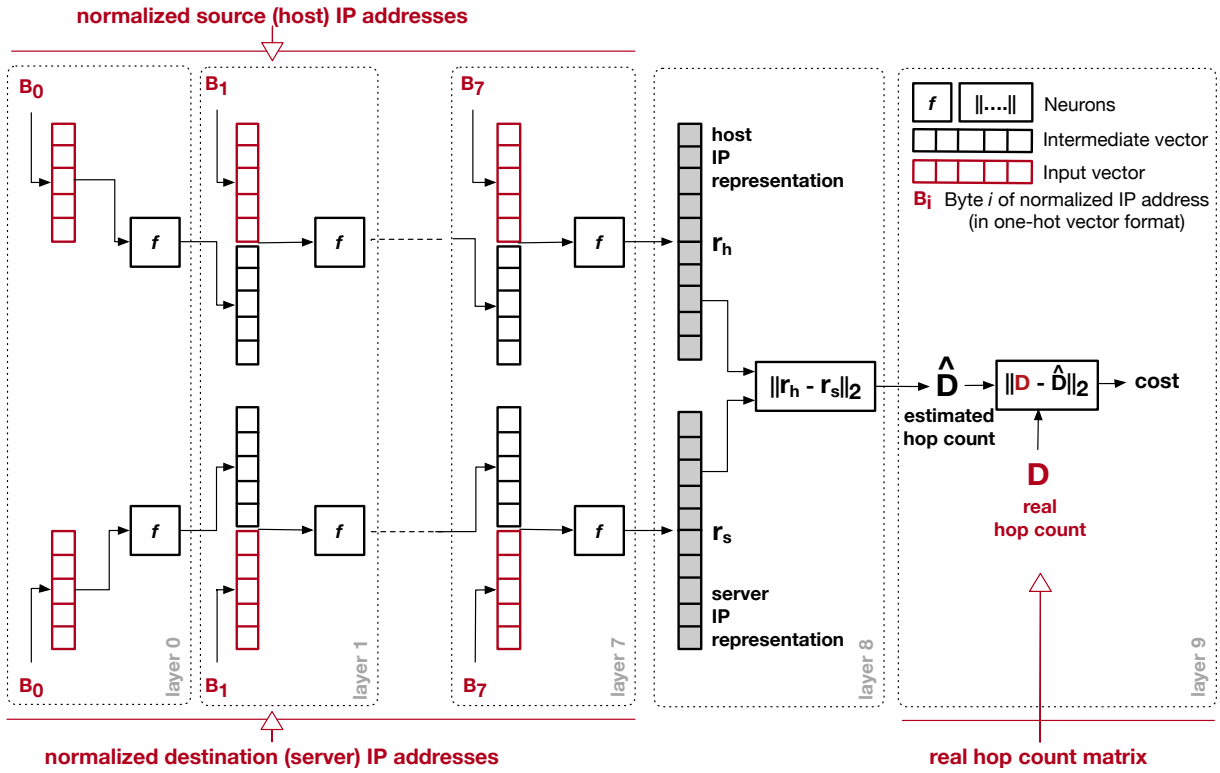

Figure 3: The neural network used for training our embedding model. The first eight layers receive the normalized IP addresses as input and compute the IP representations. The ninth layer estimates the hop count between two IP addresses and the tenth layer measures the model error. Elements in red are input. For simplicity we depict the input as one-dimensional vectors (one normalized IP); in reality, all inputs are matrices.

At each layer, the activation function f is given by:

$$f^i = \begin{cases} i = 0 & softsign(w^{i=0}_{d\times 256} \times B^{i=0}_{256\times 1} + b^{i=0}_{d\times 1}) \\ i \in \{1,\ldots,7\} & softsign(w^{i\in\{1,\ldots,7\}}_{d\times(256+d)} \times concat(B^{i\in\{1,\ldots,7\}}_{256\times 1}, f^{i-1}_{d\times 1}) + b^{i\in\{1,\ldots,7\}}_{d\times 1}) \end{cases} \quad (2)$$

where $w^{i\in\{0,\ldots,7\}}_{d\times(256+d)}$ are weights and $b^{i\in\{0,\ldots,7\}}_{d\times 1}$) are biases; the *softsign* function is $f(x) = \frac{1}{1+|x|}$. Initially, we assign random values to all weights and zeros to all biases. We employ *softsign* as the activation function for the ease of training, as *softsign* is more robust to saturation compared to other popular activation functions, such as *sigmoid* and *tanh*.

Intermediate distance estimation. We use the first eight layers of the neural network to process each of the eight bytes of the input normalized IP address. The output of the eighth layer is the intermediate vector representation for each IP address in the input data. We then use the last two layers to estimate how good the representation is. First we compute the estimated hop counts given by the current representation using an Euclidean distance. Given two matrices $H_{h\times d}$ and $S_{s\times d}$ storing the intermediate representations for the h hosts and s servers separately, the estimated distance matrix is:

$$Dist_{h\times s} = Euclidean(H_{h\times d}, S_{s\times d}) \quad (3)$$

Error reduction. Finally, we compare the estimated hop counts with the real hop counts matrix $D_{h\times s}$ to compute the cost as the mean difference of hop-counts. As the real hop count matrix is sparse, we compare only the valid entries:

$$Cost = \frac{\sum_{i=1}^{h}\sum_{j=1}^{s} W^{(i,j)}(||r^{H_{i\in\{1,..,h\}}}_{d\times 1} - r^{S_{j\in\{1,..,s\}}}_{d\times 1}|| - D^{(i,j)})}{count\ of\ non-zero\ D^{(i,j)}} \quad (4)$$

$D^{(i,j)}$ represents the value of the element at i^{th} row and j^{th} column in matrix D. $r^{H_{i\in\{1,..,h\}}}_{d\times 1}$ and $r^{S_{j\in\{1,..,s\}}}_{d\times 1}$ are rows in the matrices $H_{h\times d}$ and $S_{s\times d}$, and correspond to the representation of a host or server in the embedding space. W is a binary (0-1) matrix whose elements are defined as:

$$W^{(i,j)} = \begin{cases} 0 & D^{(i,j)} == 0 \\ 1 & D^{(i,j)} \neq 0 \end{cases} \quad (5)$$

To minimize the cost, we utilize the Adam algorithm, a gradient descent based back-propagation method [12], which is able to automatically tune the learning rate during the training process.

4 EVALUATION

4.1 Data and methodology

We use a large data set of network hop counts from the Ark project [1]. The data contains hop count information from 95 geographically distributed servers to ten million IP addresses that cover all routable prefixes in the Internet. We use data collected by Ark during Jun 2015. For each IP in the data, we look up the routable prefix and normalize it using the steps in Section 3.2. Due to the cost of monitoring a large number of IPs, not all servers have hop counts for all ten million IPs. Our hop count matrix is incomplete and contains valid entries for only 29% of the pairs. We extract IP prefix information from Routeviews data [20] and use a default value of 24 for missing prefixes.

We build a prototype for DIP using TensorFlow. We train the neural network using several smaller data sets obtained by randomly sampling 1,000, 10,000, and 100,000 IPs from the original data and keeping only the hop counts to them. Sampling increases the sparsity of the data: less than 15% of the entries in the smaller data sets are valid. We also vary the number of servers and the dimensionality of the embedding space. Intuitively, having fewer IPs or servers may not provide sufficient constraints to learn accurate representations and lead to an underfit model. Increasing the number of dimensions can reveal more hidden features, invisible at lower dimensions, but may lead to overfitting. Each training session has 2,500 iterations, *i.e.*, passes through the neural network to update the weights and biases. We use a GPU server with four 3.5GHz quad-core Intel Xeon processors and 128GB of RAM. We generate testing sets by randomly sampling the original data and preserving the previously trained parameters for embedding arbitrary IP via its address (*e.g.*, the weights and biases of each byte/layer).

4.2 Embedding accuracy

Clustering. First, we assess how well DIP preserves the clustering of hosts in the original IP space. For this, we group all IPs first according to their routable prefix and then at random. For each cluster we compute an embedding similarity metric, defined as the ratio between the average distance between all pairs of IP representations in the cluster and the maximum distance across all clusters. The lower the similarity value, the closer to each other the IPs of a cluster are in the embedding space. Figure 4(left) shows the similarity distribution for prefix-based and random clusters. Each IP representations is a 140-dimensional vector and computed after training the network using 10,000 IP addresses and 95 servers. Our embedding preserves the clustering of the original IP space well.

Distance prediction. To assess the quality of distance prediction, we first look at previous embedding mechanisms. Network coordinate approaches [4, 18] are not directly comparable as they embed latencies between strategically chosen pairs of nodes, while we rely on hop count information from passively observed traffic. Eriksson *et al.* [7] propose a matrix factorization based algorithm to predict hop count information but first build baseline representations of the monitoring servers using an all-to-all hop count matrix. We lack complete hop count information among servers and build our embedding directly from incomplete server-to-host distances. Therefore, we compare against a *mean estimation approach*, where we predict a host-to-server distance as the mean of all valid distances to the same server.

	DIP		Mean	
	known IPs	**new IPs**	**known IPs**	**new IPs**
Number of IPs				
1,000	2.16 (1.86)	2.89 (2.38)	2.99 (2.44)	3.27 (2.51)
10,000	2.15 (1.79)	2.68 (2.34)	3.00 (2.40)	3.04 (2.43)
100,000	2.06 (1.76)	2.29 (2.00)	2.98 (2.40)	2.97 (2.40)
Number of servers				
12	2.79 (2.25)	3.03 (2.47)	3.21 (2.54)	3.28 (2.62)
24	2.39(2.14)	2.60 (2.25)	2.90 (2.36)	2.97 (2.42)
48	2.34 (2.05)	2.75 (2.27)	2.99 (2.39)	3.02 (2.40)
95	2.15 (1.79)	2.68 (2.34)	3.00 (2.40)	3.04 (2.43)
Embedding dimension				
110	2.28 (1.93)	2.80 (2.39)	3.00 (2.42)	3.04 (2.43)
140	2.15 (1.79)	2.68 (2.34)	3.00 (2.42)	3.04 (2.43)
170	2.19 (1.86)	2.72 (2.33)	3.00 (2.42)	3.04 (2.43)

Table 1: Absolute mean error (standard deviation between brackets) of distance prediction of DIP and *mean*, for both known and new IPs, when varying the number of IPs, the number of servers and the embedding dimension. The default values are 10,000 IPs, 95 servers, and 140 dimensions.

We look at how well our embedding estimates the hop count value between a host and a server. We first consider only the IP addresses used in the training process (*i.e.*, *known IPs*). For this, we train a model using 90% of all host-server pairs and use the remaining 10% for testing. Figure 4(right) compares the absolute error between estimated and real hop count for DIP and *mean* for the 10,000 IPs data set on 140 dimensions. DIP predicts distances with a mean absolute error of around two hops (23% mean relative error) and reduces the error of the *mean* estimation by almost 30%. Table 1 presents the average absolute error and standard deviation for hop count estimation for embeddings trained with different number of hosts, servers, or dimensions. As expected, increasing the the number of IPs, servers, or the dimensionality reduces the absolute error. We also trained models with parameters outside the ranges presented in the table but found no improvement.

New IPs. An important feature of DIP is its ability to impute hop count values to arbitrary nodes based on their IP address. *New IPs* are IP addresses not used in the training process and that DIP has never seen before. Figure 4 (right) and Table 1 show that DIP approximates distance to new IPs with high accuracy. The distance prediction error is only around half a hop more than that for known IPs. To the best of our knowledge, DIP is the first framework to predict hop counts to arbitrary hosts based only on the value of their IP address and routable prefix and without any other domain knowledge.

5 DISCUSSION

We discuss several future applications and directions of using deep learning to understand and capture the Internet structure.

Extensions. An important benefit of using neural networks for learning the structure of the Internet is that they can be extended

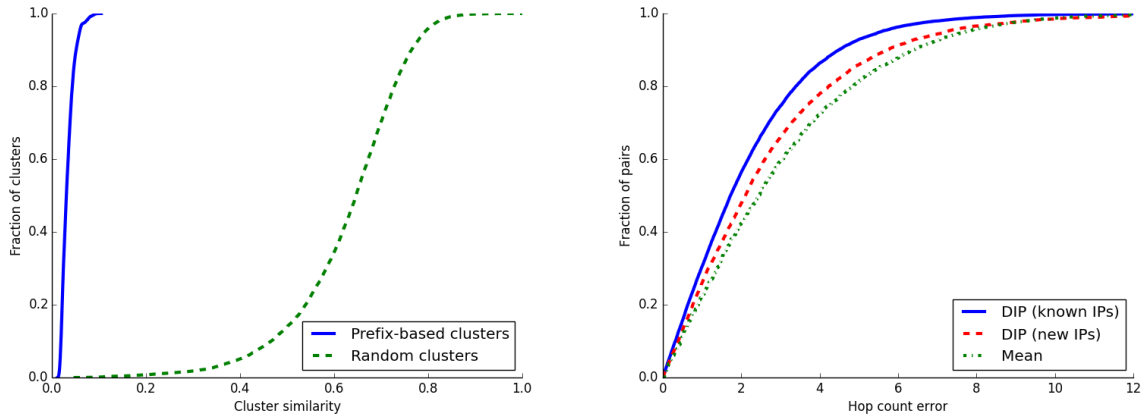

Figure 4: (left) Cumulative distribution of cluster similarity, computed using IP vector representations, for prefix-based and end-host random clusters; (right) Cumulative distributions of absolute distance estimation errors for DIP and *mean*. DIP representations preserve real-world prefix-based clustering and predict distances accurately.

easily for other data sources. Similarly to previous embedding approaches [4, 18], we could use latency measurements instead of, or in addition to, hop counts. This would require simply changing the cost estimation part of the neural network (last two layers). While gathering latency measurements is expensive as it introduces traffic into the network, the ability of our approach to work with sparse data can limit the cost necessary to obtain the measurements.

Furthermore, AS membership information could help find more accurate representations as many ASes cover limited areas in the network and provide a coarse indication of locality [7]. To add AS membership information, we could either extend the shape of our input vectors (by adding two bytes for AS number) or adapt the cost estimation layers to use AS data in estimating error, similarly to Eriksson *et al.* [7].

Applications. Building a model that accurately predicts structural properties of the Internet has several applications. Knowing the distance to remote IPs can help selecting a load balancing server or an overlay peer more efficiently and without having to perform expensive measurements. Understanding how nodes are clustered can make the transmission of video or large files faster by using close-by CDN nodes. DIP can be a passive defense mechanism against IP spoofing attacks, where malicious users change the source IP of attack packets to avoid identification and subvert authentication. By comparing the predicted distance according to the spoofed source IP to the real distance (extracted from a packet's TTL field), one could verify whether the packet is spoofed or not [10].

Limitations and future work. Our current approach uses structural information embedded in the value of IP addresses, routing data, and distances between nodes, but does not consider the actual physical links between nodes on the Internet (*i.e.*, the Internet physical topology). Adding topology would further constrain the embedding, since it is well known that the Internet is not a metric space and latency or hop count distances cannot always be embedded in metric spaces [4]. We plan to extend our framework using graph embedding algorithms to take advantage of physical topology information [22].

While our preliminary experiments focused on accuracy, the performance of building an embedding model is equally important. Training a model with 100,000 addresses and 95 servers on our 16-core GPU server takes a few hours, indicating that we may need to train models incrementally when resources are constrained [2]. For example, in a live deployment, we envision reconstructing our model every few days to capture the changes in topology triggered by the dynamic Internet. We are currently studying ways to incrementally add or update models without rebuilding from scratch.

Because our data is sparse, not even the best embedding may be able to recover all structural properties. While we show that our results are reasonably accurate, even when we have less than 15% of all distances available, getting more data is clearly helpful [7]. We plan to use active monitoring techniques (*e.g.*, *traceroute*) to collect more information for the training phase. Knowing the IPs and connectivity of routers in the network would make the training data set richer and constrain the representation of end-hosts further.

6 CONCLUSIONS

We used deep learning to learn vector representations for nodes in the Internet based on their IP address, routing information, and a sparse hop count distance matrix. Deep learning helps uncover hidden features in the input data and recover structural properties of the Internet, such as node clusters or distances between nodes. Our experiments on a large real-world data set show that our embeddings can recover most distances, even to arbitrary hosts, with two hops absolute error, even when the training data is sparse.

REFERENCES

[1] Ark [n. d.]. Ark IPv4 Routed Topology Dataset. http://www.caida.org/data/active/ipv4_routed_24_topology_dataset.xml. ([n. d.]).
[2] Lorenzo Bruzzone and D Fernandez Prieto. 1999. An incremental-learning neural network for the classification of remote-sensing images. *Pattern Recognition Letters* (1999).
[3] Manuel Costa, Miguel Castro, Antony Rowstron, and Peter Key. 2004. PIC: Practical Internet Coordinates for Distance Estimation. In *ICDCS*.
[4] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. 2004. Vivaldi: a decentralized network coordinate system. In *SIGCOMM*.

[5] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. 2010. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research* (2010).

[6] Brian Eriksson, Paul Barford, and Robert Nowak. 2008. Network Discovery from Passive Measurements. In *ACM Sigcomm*.

[7] Brian Eriksson, Paul Barford, and Robert Nowak. 2009. Estimating Hop Distance Between Arbitrary Host Pairs. In *IEEE Infocom*.

[8] Brian Eriksson, Paul Barford, Robert Nowak, and Mark Crovella. 2007. Learning Network Structure from Passive Measurements. In *IMC*.

[9] Nick Feamster and Jennifer Rexford. 2017. Why (and How) Networks Should Run Themselves. *arXiv preprint arXiv:1710.11583* (2017).

[10] Cheng Jin, Haining Wang, and Kang G. Shin. 2003. Hop-count filtering: An effective defense against spoofed DDoS traffic. In *CCS*.

[11] Andrej Karpathy and Li Fei-Fei. 2015. Deep visual-semantic alignments for generating image descriptions. In *CVPR*.

[12] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[13] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* (2015).

[14] K. Levchenko, A. Dhamdhere, B. Huffaker, k. claffy, M. Allman, and V. Paxson. 2017. PacketLab: A Universal Measurement Endpoint Interface. In *Internet Measurement Conference (IMC)*.

[15] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *ACM Sigcomm*.

[16] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Interspeech*.

[17] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. 2013. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168* (2013).

[18] T. S. Eugene Ng and Hui Zhang. 2002. Predicting Internet Network Distance with Coordinates-Based Approaches. In *INFOCOM*.

[19] Pyxida [n. d.]. Pyxida. http://pyxida.sourceforge.net/. ([n. d.]).

[20] RouteViews. [n. d.]. http://www.routeviews.org. ([n. d.]).

[21] Neil Spring, Ratul Mahajan, and Thomas Anderson. 2003. Quantifying the Causes of Path Inflation. In *ACM Sigcomm*.

[22] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural Deep Network Embedding. In *KDD*.

[23] Liwei Wang, Yin Li, Jing Huang, and Svetlana Lazebnik. 2018. Learning two-branch neural networks for image-text matching tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2018).

[24] Xiaohan Zhao, Alessandra Sala, Haitao Zheng, and Ben Y Zhao. 2011. Efficient shortest paths on massive social graphs. In *CollaborateCom*.