NeuRoute: Predictive Dynamic Routing for Software-Defined Networks

Abdelhadi Azzouni[1], Raouf Boutaba[2], and Guy Pujolle[1]

[1]LIP6 / UPMC; Paris, France {abdelhadi.azzouni,guy.pujolle}@lip6.fr
[2]University of Waterloo; Waterloo, ON, Canada rboutaba@uwaterloo.ca

***Abstract*—This paper introduces NeuRoute, a dynamic routing framework for Software Defined Networks (SDN) entirely based on machine learning, specifically, Neural Networks. Current SDN/OpenFlow controllers use a default routing based on Dijkstra's algorithm for shortest paths, and provide APIs to develop custom routing applications. NeuRoute is a controller-agnostic dynamic routing framework that (i) predicts traffic matrix in real time, (ii) uses a neural network to learn traffic characteristics and (iii) generates forwarding rules accordingly to optimize the network throughput. NeuRoute achieves the same results as the most efficient dynamic routing heuristic but in much less execution time.**

keywords - Routing, Machine Learning, Neural Networks, Software Defined Networking, Self Organizing Networks.

I. Introduction

The modern Internet is experiencing an explosion of the Machine-to-Machine (M2M) communications and Internet-of-Things (IoT) applications, in addition to other bandwidth intensive applications such as voice over IP (VoIP), video conferencing and video streaming services. Thus leading to a high pressure on carrier operators to increase their network capacity in order to support all these applications with an acceptable Quality of Service (QoS). The common practice to ensure a good QoS so far is to over-provision network resources. Operators over-provision a network so that capacity is based on peak traffic load estimates. Although this approach is simple for networks with predictable peak loads, it is not economically justified in the long-term.

In addition, most ISP networks today use Shortest Path First (SPF) routing algorithms, namely the Open Shortest Path First (OSPF) [1]. OSPF routes packets statically by assigning weights to links hence the routing tables are recalculated only when a topology change occurs. OSPF is a best effort routing protocol, meaning that when a packet experiences congestion, the routing subsystem cannot send it through an alternate path, thus failing to provide desired QoS during congestion even when the total traffic load is not particularly high.

Although OSPF has a QoS extension [2] that dynamically changes link weights based on measured traffic, it is still not implemented in the Internet for two major reasons. First, changing the cost of a link in one part of the network may cause a lot of routing updates and in turn negatively affect traffic in a completely different part of the network. This can be disruptive to many (or all) traffic flows. Another problem concerns routing loops that may occur before the routing protocol converges. Therefore, in networks with distributed control plane, changing the link cost is considered just as disruptive as link-failures. On the other hand, without the possibility to differentiate between traffic flows more granularly (not only based on destination IP address), dynamic routing cannot positively contribute to load balancing [3].

The dynamic routing problem, also known as QoS routing or concurrent flow routing, is a case of Multi-commodity flow problem where flows are packets or traffic flows and the goal is to maximize the total network flow while respecting routing constraints such as load balancing the total network traffic or minimizing the traffic delay. Due to their high computational complexity, multi-commodity flow algorithms are rarely implemented in practice.

There are many variants of the dynamic routing problem including the maximum throughput dynamic routing, the maximum throughput minimum cost dynamic routing and the maximum throughput minimum cost multicast dynamic routing. In this work, we focus on the maximum throughput minimum cost unicast dynamic routing where given a traffic demand matrix, the objective is to maximize the total throughput of the network while minimizing the cost of routing the total traffic knowing that each flow can be routed through only one end-to-end path. We present NeuRoute, a Neural Network based hyperheuristic that is capable of computing dynamic paths in real time. NeuRoute learns from a dynamic routing algorithm then imitates it achieving the same results but in only 25% of its execution time. The basic motivation behind NeuRoute is that dynamic routing using traditional algorithmic solutions is not practical due to their high computational complexity. That is, at every execution round the routing algorithm uses measured link loads as input and performs a graph search to find the near optimal paths.

The main contributions of this paper are summarized as follows: (i) We introduce for the first time an integral routing system based on machine learning and detail its architecture, (ii) we detail the design of the neural network responsible for matching traffic demands to routing paths and (iii) we evaluate our proposal against an efficient dynamic routing heuristic and show our solution's superiority.

The remainder of this paper is organized as follows: Section II formally states the dynamic routing problem and discusses its most prominent heuristic solutions. Section III details NeuRoute design. In section IV, we evaluate NeuRoute on real

978-3-901882-98-2 © 2017 IFIP

world network data and topology. We discuss related work in section V and we conclude the paper in section VI

II. The Dynamic Routing Problem

In this section, we formulate the maximum throughput minimum cost dynamic routing problem (MT-MC-DRP) as a linear program, and then prove its NP-completeness. The problem is equivalent to the known Unsplittable Constrained Multicommodity Max-Flow-Min-Cost problem. We want to find routes for multiple unicast flows which maximize the aggregate flow in a graph, while minimizing the routing-cost. By focusing on unsplittable multicommodity flow we exclude multipath routing where a flow can be split and routed through multiple end-to-end paths.

We consider a software-defined network $G(V, L)$, where V is the set of SDN-enabled switch nodes, and L is the set of links that connect the switches where each link $l_{i,j}$ has a capacity $C(l)$. Each unicast flow f has source and destination nodes denoted s_f and d_f respectively, a requested traffic rate R^f and a minimum necessary traffic rate N^f. Let $r^f_{in}(v)$ and $r^f_{out}(v)$ denote the aggregate flow rate into/out of node v due to flow f, respectively. The traffic rate related to flow f and flowing through link l is denoted by $r^f(l)$. Each link has a routing cost denoted by $\Theta(l)$ that can represent any linear function of the traffic flowing on it , i.e., delay, jitter, congestion probability or reliability. We define an Admissible Routing as an assignment of flows to the links in G, such that no capacity constraints are violated, and flow-conservation applies at every node. The MT-MC-DRP problem can be stated as follows: Does there exist an admissible routing for the flows, where each flow receives its requested rate R^f while the total routing cost is minimized?

A. MT-MC-DRP As Two Linear Problems

We formulate MT-MC-DRP as a succession of two linear problems (LPs): A Constrained-Maximum-Flow LP (CMaxF-LP) and a Constrained-Minimum-Cost LP (CMinC-LP).

1) CMaxF-LP:

$$maximize(\sum_{f \in F} r^f_{in}(d_f)) \tag{1}$$

subject to:

$$r^f(l) \geq 0 \qquad \forall f \in F, \forall l \in L^f \tag{2}$$

$$r^f(l) \leq C(l) \quad \forall f \in F, \forall l \in L^f \tag{3}$$

$$\sum_{f \in F} r^f(l) \leq C(l) \qquad \forall l \in L \tag{4}$$

$$r^f_{in}(v) = r^f_{out}(v) \qquad \forall f \in F, \forall v \in V^f - \{s_f, d_f\} \tag{5}$$

$$r^f_{in}(s_f) = 0 \qquad \forall f \in F \tag{6}$$

$$r^f_{out}(d_f) = 0 \qquad \forall f \in F \tag{7}$$

$$r^f_{out}(s_f) \leq R^f \qquad \forall f \in F \tag{8}$$

$$r^f_{out}(s_f) \geq N^f \qquad \forall f \in F \tag{9}$$

2) CMinC-LP:

$$minimize(\sum_{f \in F} \sum_{l \in L} r^f(l) \times \Theta(l)) \tag{10}$$

subject to:

$$r^f_{out}(s_f) = \Pi^f + / - \epsilon \qquad \forall f \in F, \forall l \in L^f \tag{11}$$

$$\sum_{f \in F} r^f(l) \leq C(l) \qquad \forall l \in L \tag{12}$$

$$r^f_{in}(v) = r^f_{out}(v) \qquad \forall f \in F, \forall v \in V' \tag{13}$$

$$r^f_{in}(s_f) = 0 \qquad \forall f \in F \tag{14}$$

$$r^f_{out}(d_f) = 0 \qquad \forall f \in F \tag{15}$$

Theorem. The Maximum Throughput Minimum Cost Dynamic Routing Problem as presented above is NP-hard.

Proof. refer to [5] [9] □

B. Heuristic Solution for The MT-MC-DRP

Due to its NP-completeness, an exact solution for the MT-MC-DRP as defined above is not practical to be implemented in the network controller. It is more practical to design an approximate but fast solution. Therefore, a major research effort was put into designing efficient fully polynomial-time approximation schemes (FPTAS) for multicommodity flow problems including max flow min cost multicommodity problem. A fully polynomial-time approximation scheme for a flow maximization problem is an algorithm that, given an accuracy parameter $\epsilon > 0$, computes, in polynomial time in the size of the input and $1/\epsilon$, a solution with an objective value within a factor of $(1 - \epsilon)$ of the optimal one [6]. The multicommodity problem literature has a rich body of work providing FPTASes. In this work, we use the novel method proposed in [6] as a baseline heuristic to solve the MT-MC-DRP. We also refer to the same paper for more literature on other existing heuristics.

III. System Design

As shown in figure 1, NeuRoute is designed as an integral routing application for the SDN controller. NeuRoute is composed of three key components: a Traffic Matrix Estimator (TME), a Traffic Matrix Predictor (TMP) and a Traffic Routing Unit (TRU). In this paper focus on and detail the TRU but also describe briefly the two other components for the sake of completeness.

A. Traffic Matrix Estimator

As mentioned earlier,detailed design of the traffic matrix (TM) estimator is out of the scope of this paper. Here we only motivate the need for a traffic matrix estimator and define its interfaces with the rest of NeuRoute components.

A network TM presents the traffic volume between all pairs of origin-destination (OD) nodes of the network at a certain time t. The nodes in a traffic matrix can be Points-of-Presence (PoPs), switches, routers or links. In OpenFlow SDNs, the controller leverages packet_in messages to build a

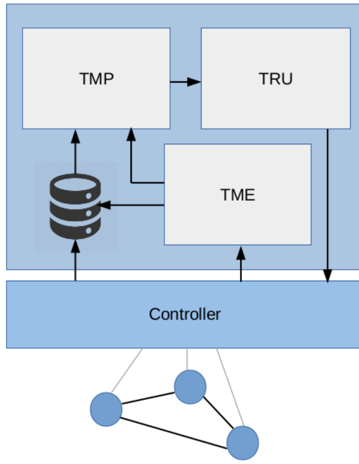


Fig. 1: NeuRoute architecture

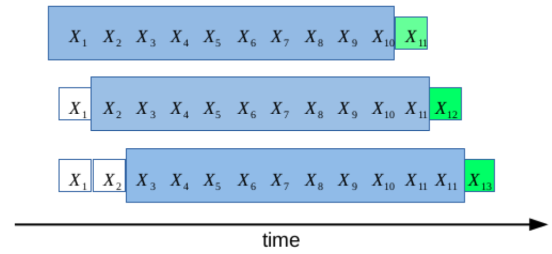


Fig. 2: Traffic Matrix Prediction Over Time

global view of the network. When a new flow arrives to a switch, it is matched against forwarding rules to determine a forwarding path for it. If the flow does not match any rule, the switch forwards the first packet or only the packet header to the controller. In addition, the controller can query switches for packet counts that track the number of packets and bytes handled by the switch. However, the number of packet_in and the number of controller queries, necessary for a near real-time measurement, increases rapidly with a large number of switches and flows, making this measurement mechanism not practical. Also, there is a chance that by the time the controller receives the message, the values of the counters become out of date and do not reflect the near real-time state of the switch anymore. These and a number of other issues listed in [12] call for an efficient measurement mechanism to capture traffic matrix in near real-time. In its current implementation, NeuRoute uses a variant of a recent proposal called openMeasure [13] to estimate traffic matrix.

B. Traffic Matrix Predictor

Network Traffic Matrix prediction refers to the problem of estimating future network traffic from the past and current network traffic data. Internet traffic is known to be self-similar enabling it to be predictable with high accuracy [10]. NeuRoute's Traffic Matrix Predictor (TMP) uses a Long Short Term Memory Recurrent Neural Network (LSTM-RNN) described in [7]. Figure 2 shows the sliding prediction window where at each time instant t, the TMP takes a fixed size set of achieved traffic matrices as input and outputs the traffic matrix of time instant $t+1$

Prediction using NNs involves two phases: a) the training phase and b) the test (prediction) phase. During the training phase, the NN is supervised to learn from the data by presenting the training data at the input layer and dynamically adjusting the parameters of the NN to achieve the desired output value for the input set. The most commonly used learning algorithm to train NNs is called the backpropagation algorithm. The underlying idea is to propagate the error backward, from the output to the input, where the weights are changed continuously until the output error falls below a preset value. In this way, the NN learns correlated patterns between input sets and the corresponding target values. The prediction phase represents the testing of the NN. A new unseen input is presented to the NN and the output is calculated, thereby predicting the outcome of new input data.

C. Traffic Routing Unit

The core component of the NeuRoute system is the Traffic Routing Unit (TRU) which is responsible of selecting optimal routes based on the predicted traffic matrix. TRU is based on the supervised learning approach where an agent is trained to infer a function from labeled training data. It consists of a Deep Feed Forward Neural Network that learns to match traffic demands to routing paths by observing the output of a heuristic, that we call the Baseline Heuristic (BH). In this paper we present our experimentations with a BH that is built following the algorithm discussed in section II-B.

To bootstrap, only the TME is activated to continuously provide the BH with timely estimated traffic matrices. Copies of these estimated traffic matrices are stored to be used later on by the TMP and the TRU. NeuRoute collects the output of the BH for a period of time that can be configured based on the desired performance. Once enough BH-output data is gathered, NeuRoute's components, TMP and TRU, are fired up. The TMP uses the stored history of estimated traffic matrices to predict the future traffic matrix, continuously as detailed in [7]. On the other hand, the TRU takes the BH output data and the stored history of estimated traffic matrices along with corresponding Network States (NSs) as input to train its routing neural network. Each tuple (NS+traffic matrix, BH output) constitutes one learning sample for the TRU. NS at a time instant t (or NS_t) is the set of all links available capacities and links costs at time instant t (links costs usually do not change frequently). Once the learning phase is done (within a few seconds to a few minutes depending on the volume of data and desired performance), the trained model is fired up to route new traffic flows. The reason why we predict the traffic matrix is that the real-time measurement of traffic matrix is not practical and by the time the controller gets the measured information, the flows to be routed are already on their way on the existing paths, before even the controller computes the new paths. In the following, we detail the design elements and the design challenges of TRU.

1) Deep Feed Forward Neural Networks: Deep neural networks are currently the most successful machine learning technique for solving a variety of tasks including language translation, image classification and image generation. TRU is similar to an image classifier that has a set of images in input and tries to find a function that matches these images to a set of classes. In the routing case, the traffic matrices are the images and the routing paths represent the output classes. The deep neural network used in TRU is presented in figure 3. It takes a traffic matrix and an NS instance as input and matches them to a set of paths as output.

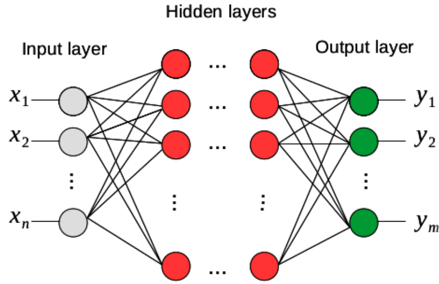


Fig. 3: Deep Feed Forward Neural Network

In a deep feed forward network, the information flows only forward through the network from the input nodes, through the hidden nodes to the output nodes, with no cycles or loops. Each node has an activation function which acts like a threshold for the node to fire up: A node n produces a value for its output nodes only if the weighted sum of the input values of n is equal or exceeds the threshold. Each edge has a weight and permits transfer of value from node to node.

Learning Algorithm. We use the Backpropagation learning algorithm that was first introduced in the 70s and now is the most widely used algorithm for supervised learning in deep feed-forward networks. The goal is to make the network learn some target function, in our case, matching traffic matrices to routing paths. The basic idea of the algorithm is to look for the minimum of the error function in weight space by repeatedly applying the chain rule to compute the influence of each weight in the network with respect to the error function: The output values of the network are compared with the learning sample (correct answer) to compute the value of the error function. The calculated error is then fed back through the network and used to adjust the weights of each connection in order to reduce the value of the error function by some small amount. After repeating this process for a sufficiently large number of training cycles, the network will usually converge to some state where the error is small enough. In other words, we say that the network has learned the target function to some extend. We refer to [14] for more details about the algorithm.

Optimization Algorithm. In this work, we use Adam (short for Adaptive Moment Estimation) optimizer, one of the most adopted optimization algorithms among deep learning practitioners for applications in computer vision and natural language processing. Adam optimizer is an improvement of the gradient descent algorithm that can yield quicker convergence in training deep networks [15].

Learning Rate. The learning rate determines how quickly or how slowly we want the network weights to be updated (by the backpropagation algorithm). In other words, how quickly or how slowly we want the network to forget learned features and learn new ones. Picking a learning rate is problem dependent since the optimum learning rate can differ based on a number of parameters including epoch size, number of learning iterations, number of hidden layers and/or neurons and number and format of the inputs. Trial and error is often used in order to determine the ideal learning condition for each problem studied. We describe our empirical approach for choosing the learning rate in the implementation section IV.

2) Input Pre-Processing and Normalization: The input (NS+traffic matrix) are merged into one single vector of numbers then normalized by dividing all numbers by the greatest number. The result is a vector of numbers ranging between 0 and 1. This normalization is a good practice that can make training faster and reduce the chance of getting stuck in local optima [8].

3) Routing Over Time: At each time instant t, the TRU's trained model takes predicted traffic matrix of time instant $t+1$ (TM_{t+1}) and corresponding NS as input. The model function is applied and the output is a set of path probabilities where the highest value indicates the best routing path. TRU then sends the chosen path to the controller in order to be installed in switches as flow rules. By the time $t+1$, when the flows arrive, the forwarding rules are already installed which minimizes considerably the network delay.

Matching traffic matrices and network states to routing paths is similar to classifying a stream of frames in a video, witch is not a common and well studied problem since the usual image classification is applied to individual images. Besides tweaking the neural network architecture and parameters to obtain a high classification performance, there are two unique challenges that arise in our problem:

- The runtime performance of the trained model is critical and needs to be optimized to perform continuous routing over time. We achieve high performance by keeping the predicted traffic matrices in memory before feeding them to the LRU's neural network.
- Unlike images and videos, there is no camera bias in traffic matrices (Camera bias refers to the fact that in many images and videos, the object of interest often occupies the center region), hence it is not possible to work around resolutions to optimize training time as it was done in [11].

IV. Implementation and Evaluation

We implemented NeuRoute as a routing application on top of POX controller [18]. The TRU's neural network is implemented using Keras library [19] on top of Google's TensorFlow machine learning framework [20]. We have chosen the GÉANT network topology for our testbed as GÉANT's traffic

matrices are already available online [21]. We implemented the GÉANT topology (shown in figure 4) as an SDN network using Mininet [22] setting link capacities at 10Mbps. We use link delay as the cost function with 2ms delay per link.

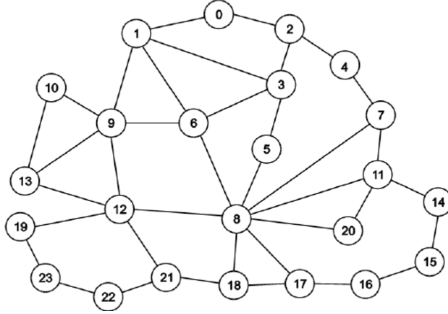
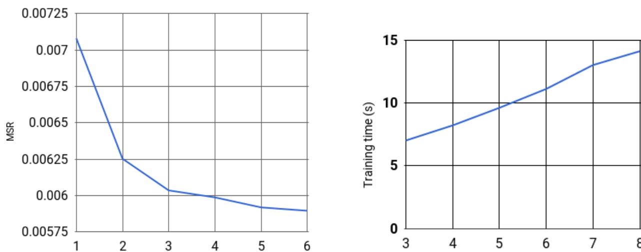


Fig. 4: GÉANT2 Network Topology [23]

Data generation. In order to generate the learning data, we applied the BH on the testbed described above with GÉANT's traffic matrices as input. We obtained a data set of 10000 samples (traffic matrix+network state, near optimal path) that we split to training data set of 7000 samples and test data set of 3000 samples.

The neural network architecture. Determining the neural network architecture is problem dependent, hence we adopted an empirical approach to determine the number of hidden layers and the size of each hidden layer. We measured the training time and the learning performance (GÉANT traffic matrices + related network states as input and the results of the BH as output) for different numbers of hidden layers and different hidden layer sizes. This allowed us to pick an optimal number of hidden layers of 6 with 100 nodes per hidden layer. Note that we choose the architecture parameters based on the measured learning performance, and we stop experimenting when the training time becomes too long.

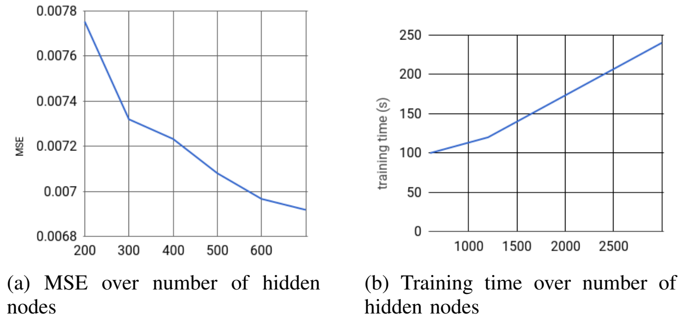


(a) MSE over number of hidden layers

(b) Training time over number of hidden layers

Fig. 5: Picking the number of hidden layers

Figure 5a depicts the measured Mean Squared Error (MSE) over different numbers of hidden layers. The MSE diminishes at high numbers of hidden layers (deep network) but figure 5b shows that the deeper is the network the longer it takes to train it. To select a good compromise, we fix the training time to 2 minutes. This training time corresponds to a depth of 6 hidden layers.



(a) MSE over number of hidden nodes

(b) Training time over number of hidden nodes

Fig. 6: Picking the number of hidden nodes

Similarly, figure 6a shows that the MSE diminishes at higher network sizes but the training time goes up as figure 6b shows. We fix again the training time to 2 minutes and obtain the corresponding hidden nodes number of 600, or 100 nodes per hidden layer. Note that a 2 minutes training time is not too long but is chosen proportionally to the size of the data set. Larger data sets may take hours or days to train.

Data preparation. We prepared the input data as follows: we split the total learning data into batches of size 100 each. Each input sample is a vector of size $506 + 38 = 544$, 506 being the size of a vector representing one traffic matrix of 23 nodes (23*22) and 38 being the number of links in the GÉANT topology, which is equal to the size of one network state vector. The output vector is of size $23 * 22 * 5$ with 23*22 being the number of origin-destination (OD) pairs and we arbitrarily fix the number of possible paths per OD pair to 6.

The learning rate. Like the neural network's architecture, the learning rate is problem dependent. Our approach is to start with a high value and go down to lower values, recording the learning performance and training time for every learning rate value.

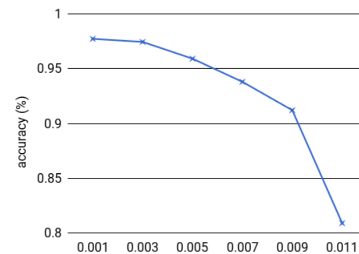


Fig. 7: Accuracy over different learning rate values

Figure 7 depicts the MSE variation over different learning rate values. The training time does not change for different learning rates (5s per epoch).

The overfitting problem. Overfitting is a serious problem that occurs when training a neural network on limited data. It happens when a model learns the detail and noise in the training data to the extent that it negatively impacts its performance on new data. This means that the noise or random fluctuations in the training data is picked up and learned as

features by the model. The problem is that these features do not apply to new data and negatively impact the model's ability to generalize. Various methods have been proposed to avoid or reduce overfitting, including stopping the training as soon as performance on a validation set starts to get worse, introducing weight penalties of various kinds such as L1 and L2 regularization and Dropout [24]. In this work, we chose to use the Dropout technique due to its simplicity and effectiveness [24]. Dropout is a technique that addresses both these issues. Dropout is a technique where randomly selected neurons are are dropped-out (or ignored) during training and their contribution to the activation of the downstream neurons is temporally removed. Also, the weight updates are not applied to the dropped-out neurons on the backward pass. The effect on the network is that it becomes less sensitive to the specific weights of neurons which results in a better generalization [24] [25].

Evaluation of TRU. Finally, we applied the trained model on the test data and recorded the accuracy (number of correctly chosen paths from the test set) over number of training epochs in figure 8. One epoch is a one complete training pass over the whole training data set where each epoch takes roughly 2s to complete. Figure 8 shows that the model picks the near optimal path learned from the BH with an estimated error of less than 0.05% when trained well (3min of training is enough to reach this error rate). Furthermore, the trained model executes and finds the near optimal path in 30ms compared to the BH execution time of 120ms.

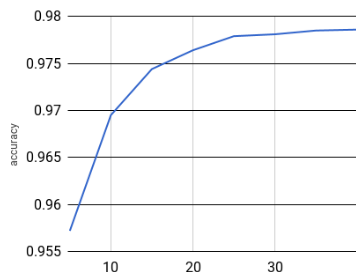


Fig. 8: MSE over number of training epochs

V. Related Work

The authors of paper [17] propose a machine learning meta-layer composed of multiple modules. Each module works only for one OD pair. The proposed scheme is however not practical since the number of OD pairs (hence the number of neural networks associated) explodes in large networks. Knowing that each neural network is trained separately and each trained model operates separately, this approach does not capture the relations between ODs requests that arrive at the same time. It is also much more complicated to implement and computationally expensive than our approach.

VI. Conclusion

In this paper, we introduced NeuRoute, a machine learning based dynamic routing framework for SDN. NeuRoute learns a routing algorithm and imitates it with higher performance. We implemented NeuRoute as a routing application on top of Pox Controller and performed proof of concept experiments that showed our solution's superiority compared to an efficient dynamic routing heuristic. Experiments on larger data sets are being conducted and will be presented in a future work along with more details about the system.

References

[1] Moy, John. "OSPF version 2." (1997).
[2] IETF. "QoS Routing Mechanisms and OSPF Extensions", RFC 2676, Aug. 1999.
[3] Tomovic, Slavica, et al. "A new approach to dynamic routing in SDN networks." Electrotechnical Conference (MELECON), 2016 18th Mediterranean. IEEE, 2016.
[4] Szymanski, Ted H. "Max-flow min-cost routing in a future-Internet with improved QoS guarantees." IEEE Transactions on Communications 61.4 (2013): 1485-1497.
[5] Hall, Alex, Steffen Hippler, and Martin Skutella. "Multicommodity flows over time: Efficient algorithms and complexity." Theoretical Computer Science 379.3 (2007): 387-404.
[6] Madry, Aleksander. "Faster approximation schemes for fractional multi-commodity flow problems via dynamic graph algorithms." Proceedings of the forty-second ACM symposium on Theory of computing. ACM, 2010.
[7] Azzouni, Abdelhadi, and Guy Pujolle. "A Long Short-Term Memory Recurrent Neural Network Framework for Network Traffic Matrix Prediction." arXiv preprint arXiv:1705.05690 (2017).
[8] Sola, J., and J. Sevilla. "Importance of input data normalization for the application of neural networks to complex industrial problems." IEEE Transactions on Nuclear Science 44.3 (1997): 1464-1468.
[9] Szymanski, Ted H. "Max-flow min-cost routing in a future-Internet with improved QoS guarantees." IEEE Transactions on Communications 61.4 (2013): 1485-1497.
[10] Mansfield, Glenn, T. K. Roy, and Norio Shiratori. "Self-similar and fractal nature of Internet traffic data." Information Networking, 2001. Proceedings. 15th International Conference on. IEEE, 2001.
[11] Karpathy, Andrej, et al. "Large-scale video classification with convolutional neural networks." Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. 2014.
[12] Suh, Junho, et al. "Opensample: A low-latency, sampling-based measurement platform for commodity sdn." Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on. IEEE, 2014.
[13] Liu, Chang, AMehdi Malboubi, and Chen-Nee Chuah. "OpenMeasure: Adaptive flow measurement & inference with online learning in SDN." Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on. IEEE, 2016.
[14] Demuth, Howard B., et al. Neural network design. Martin Hagan, 2014.
[15] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
[16] Kohavi, Ron. "A study of cross-validation and bootstrap for accuracy estimation and model selection." Ijcai. Vol. 14. No. 2. 1995.
[17] Yanjun, Li, Li Xiaobo, and Yoshie Osamu. "Traffic engineering framework with machine learning based meta-layer in software-defined networks." Network Infrastructure and Digital Content (IC-NIDC), 2014 4th IEEE International Conference on. IEEE, 2014.
[18] The POX controller. https://github.com/noxrepo/pox.
[19] Keras Documentation. https://keras.io/
[20] Google TensorFlow. https://www.tensorflow.org/
[21] https://goo.gl/JD6t78
[22] http://mininet.org/
[23] Barreto, Fernando, Emlio CG Wille, and Luiz Nacamura Jr. "Fast emergency paths schema to overcome transient link failures in ospf routing." arXiv preprint arXiv:1204.2465 (2012).
[24] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." Journal of machine learning research 15.1 (2014): 1929-1958.
[25] Jason Brownlee. Dropout Regularization in Deep Learning Models With Keras. http://machinelearningmastery.com/dropout-regularization- deep-learning-models-keras/