Learning To Route with Deep RL

Asaf Valadarsky, Michael Schapira, Dafna Shahaf
The Hebrew University of Jerusalem
{firstname.lastname}@mail.huji.ac.il

Aviv Tamar
UC Berkeley
avivt@berkeley.edu

Abstract

We investigate a novel and important application domain for deep RL: network routing. The question of whether/when traditional network protocol design, which relies on the application of algorithmic insights by human experts, can be replaced by a data-driven approach has received much attention recently. We explore this question in the context of the, arguably, most fundamental networking task: routing. Can ideas and techniques from machine learning be leveraged to automatically generate "good" routing configurations? We observe that the routing domain poses significant challenges for data-driven network protocol design and report on preliminary results regarding the power of data-driven routing. Our results suggest that applying deep reinforcement learning to this context yields high performance and is thus a promising direction for further research. We outline a research agenda for data-driven routing.

1 Introduction

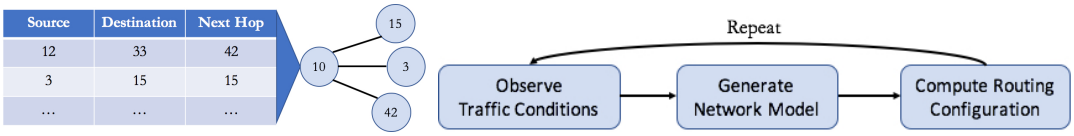
A fundamental challenge in networking is routing – dictating how traffic originated at one communication end-point within the network, and destined for to another end-point, should traverse the physical (network) graph inter-connecting these two. A routing configuration specifies, for every network node (e.g., router, switch), v, a "mapping" from each source and destination of traffic (and possibly additional metadata) to one of v's neighbors, representing v's "next hop node" en route to that traffic destination (i.e., the neighbor to whom traffic in this class should be forwarded by v). An example of such a mapping can be found in Figure 1a.

Why apply machine learning to routing? Routing is, arguably, the most fundamental networking task and, consequently, has been extensively researched in a broad variety of contexts. Traditionally, the optimization of routing with respect to the prevailing traffic demands contends with uncertainty about future traffic conditions in one of two manners: (1) optimizing routing configurations with respect to previously observed traffic conditions, with the hope that these configurations will also fare well with respect to future conditions, or (2) optimizing routing configurations with respect to *a range* of feasible traffic scenarios, with the aim of providing high performance across the entire range [16, 10, 29, 7]. We provide a sketch of the decision making process when computing routing configurations in Figure 1b.

Unfortunately, in general, routing configurations optimized with respect to specific traffic conditions can fail miserably in achieving good performance even under not-too-different traffic conditions. In addition, *simultaneously* optimizing the worst-case performance with respect to a broad range of considered traffic scenarios might come at the cost of being far from the achievable optimum for the actual traffic conditions.

Intuitively, ML suggests a third option: leveraging information about *past* traffic conditions to learn good routing configurations for *future* conditions. While the exact traffic demands are unknown to the decision maker in advance, a realistic assumption is that the history of traffic demands contains *some* information regarding future traffic patterns (e.g., changes in traffic across times of day, the

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.



(a) *A snippet of a network and routing configuration of node 10.*

(b) *A sketch of the decision making process for computing routing configurations.*

Figure 1: *A sketch showing a network and part of its routing configuration (Figure 1a) and how this configuration changes over time (Figure 1b).*

skewness of traffic, whether certain end-hosts communicate often, etc.). Hence, a natural approach is to continuously observe traffic demands and adapt routing with respect to (implicit or explicit) *predictions* about the future.

We explore how ML ideas can be applied to the routing domain. In particular, our focus is on deep learning [19], which, over the past decade, has dominated a broad range of areas, including computer vision [28], machine translation [46], and more recently, reinforcement learning and control [39, 31]. Our premise is that deep learning could be used for learning better routing strategies than those generated by conventional methods.

Intradomain routing as a case study. We initiate the study of ML-guided routing by examining the classical environment of intradomain traffic engineering [15, 16, 17, 24, 52, 29, 10]–the optimization of routing within a single self-administered network. We leave the investigation of data-driven routing in other contexts to future research (Section 7).

We present a model for data-driven (intradomain) routing that builds on the rich body of literature on this topic [15, 16, 17, 24, 52, 29, 10] and on (multicommodity [44, 23, 7, 10, 29, 16, 5]) flow optimization. We investigate, within this model, the application of different deep learning paradigms and machinery to the challenge of computing routing configurations. In our investigation of the application of deep learning to routing, we grapple with two main questions: (1) How should routing be formulated as an ML problem? and (2) What are suitable representations for learning in this domain? We next expand on each of these challenges, which also pertain to data-driven routing in other contexts.

Learning the next input or routing configurations directly? One natural approach to ML-based routing is the following: observe past traffic demands, apply machine learning to *explicitly* predict the upcoming traffic demands, and optimize routing with respect to the predicted demands. In ML terms, this is a *supervised learning* task [40]. We evaluate several supervised learning schemes for predicting traffic demands. Our preliminary results are discouraging, indicating that supervised learning might be ineffective if the traffic conditions do not exhibit very high regularity. We next turn our attention to a different approach: applying *reinforcement learning* [47]. Now, instead of attempting to explicitly learn traffic demands and optimizing routing with respect to our predictions, the goal is to directly learn a good mapping from the observed history of traffic demands to routing configurations. Our preliminary results suggest that this approach is more promising, yet realizing it requires care, as discussed next.

What should the output of the learning scheme be? We observe that the intradomain routing context poses significant challenges to the application of reinforcement learning. A key challenge is that the natural "output" of a (general) routing scheme is a collection of rules specifying how traffic is forwarded from each source to each destination, whose naive representation involves a very large set of parameters (as opposed to, e.g., selecting a single action from a fairly small set [33, 34]). Our initial results indicate that this can render learning slow and ineffective. We hence devise methods for constraining the size of the output without losing "too much" in terms of expressiveness of routing configurations. We leverage ideas from the literature on hop-by-hop traffic engineering [36, 52, 16] to efficiently learn (via reinforcement learning) experimentally good routing configurations. Our preliminary findings suggest that our approach constitutes a promising direction for improving upon today's intradomain routing schemes.

Outlining a research agenda for data-driven routing. We believe that our investigation thus far only scratched the surface of the potential of data-driven routing and leave the reader with many interesting research questions, including (1) extending our approach to other routing contexts, (2)

examining other performance metrics, (3) identifying better supervised learning approaches to traffic-demand estimation, (4) scaling reinforcement learning in this context, and beyond. We discuss this research agenda in Section 7.

2 Data-Driven Routing Model

Informally, in our framework, a decision maker (network operator / automated system) repeatedly selects routing configurations for the network. Traffic conditions might vary and routing decisions are oblivious to future traffic demands. Our focus is on the conventional optimization objective of minimizing link over-utilization (a.k.a. minimizing congestion) from traffic engineering literature [16, 7, 10, 29]. We formalize our framework below.

Network. We model the network as a capacitated directed graph $G = (V, E, c)$, where V and E are the vertex and edge sets, respectively, and $c : E \to \mathbb{R}^+$ assigns a capacity to each edge. Let n denote the number of vertices in V and $\Gamma(v)$ denote vertex v's neighboring vertices in G.

Routing. A *routing strategy* $\mathcal{R}$ for the network specifies, for each source vertex s and destination vertex t how traffic from s to t that traverses v is split between v's neighbors. Thus, a routing strategy specifies, for each vertex v and source-destination pair (s, t) a mapping from v's neighbors to values in the interval $[0, 1]$, $\mathcal{R}_{v,(s,t)} : \Gamma(v) \to [0, 1]$, such that $\mathcal{R}_{v,(s,t)}(u)$ is the fraction of traffic from s to t traversing v that v forwards to its neighbor u. We require that for every $s, t \in V$ and $v \neq t$, $\sum_{u\in\Gamma(v)} \mathcal{R}_{v,(s,t)}(u) = 1$ (no traffic is blackholed at a non-destination), and also for every $s, t \in V$, $\sum_{u\in\Gamma(v)} \mathcal{R}_{t,(s,t)}(u) = 0$ (all traffic to a destination is absorbed at that destination).

Induced flows of traffic. A *demand matrix* (DM) D is a $n \times n$ matrix whose (i, j)'th entry $D_{i,j}$ specifies the traffic demand between source i and destination j. Observe that any demand matrix D and a routing strategy $\mathcal{R}$ induce a flow of traffic in the network, as explained next; Traffic from every source s to destination t is split amongst s's neighbors according to $\mathcal{R}_{s,(s,t)}$. Similarly, traffic from s to t traversing a neighbor of s, v, is split amongst v's neighbors according to $\mathcal{R}_{v,(s,t)}$, etc.

How good is a traffic flow? We adopt the classical objective function of minimizing link (over)utilization (also termed congestion [16]) from traffic engineering literature [16, 10, 7, 29], though our formulation can easily be extended to other multicommodity-flow-based objective functions. The link utilization under a specific multicommodity flow f is $max_{e\in E} \frac{f_e}{c(e)}$, where f_e is the total amount of flow traversing edge e under flow f.

We point out that for any *given* demand matrix, computing a multicommodity flow f that minimizes link utilization can be executed in a computationally-efficient manner via linear programming [23, 7, 16]. Our focus, in contrast, is on the realistic scenario where the DM is not known beforehand.

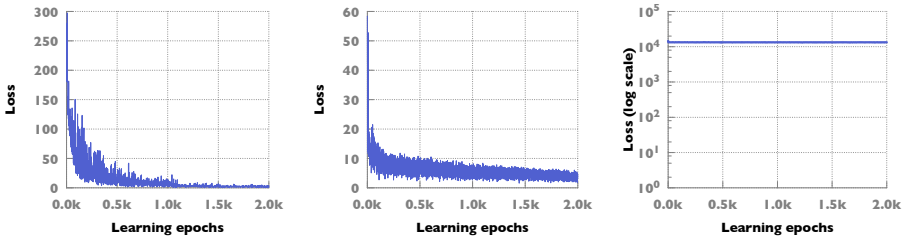
Routing future traffic demands. In our framework, time is divided into consecutive intervals, called "epochs", of length δ_t (δ_t is determined by the network operator). At the beginning of each epoch t, the routing strategy $\mathcal{R}^{(t)}$ for that epoch is decided. $\mathcal{R}^{(t)}$ can depend only on the history of *past* traffic patterns and routing strategies (i.e., from epochs $1, \ldots, t-1$).

We make two simplifying assumptions: (1) that the demand matrix is fixed within each time epoch, and (2) that demand matrices can be inferred after the fact (e.g., via network measurements). We leave the investigation of data-driven routing under more complex traffic patterns (e.g., capturing IP flows entering and leaving within each epoch) and of information-constrained routing decisions (e.g., only relying on partial information about the traffic demands) to the future. As discussed below, studying this simple setting already gives rise to interesting insights.

After selecting the routing strategy $\mathcal{R}^{(t)}$ for epoch t, the demand matrix for epoch t, and the associated cost, in terms of maximum link-utilization, are revealed. The objective of the decision maker is to select routing strategies in a manner that consistently results in low link over-utilization.

3 What to Learn?

Our underlying assumption is the existence of some *regularity* in the DMs, and the purpose of the investigation below is exploring ideas for how such regularity can be inferred/leveraged to optimize routing with respect to future DMs. We consider two different manifestations of regularity—



(a) *Cyclic gravity DM sequences (sparsity $p = 0.3$)*

(b) *Averaged bimodal DM sequences (sparsity $p = 1$)*

(c) *Randomly drawn gravity DM sequences (sparsity $p = 0.3$)*

Figure 2: *Representative Results for Supervised Learning (using* NAR-NN *with $k = 10$ and $q = 5$)*

embedding *deterministic* regularity into the DM sequence and drawing DMs from a *fixed probability* distribution—and two high-level learning approaches—supervised learning and reinforcement learning.

3.1 Supervised Learning Approach

Our goal is to extract "good" routing strategies from past observations. Since for a given demand matrix (DM), an optimal routing strategy is efficiently computable, a natural approach is to repeatedly try to predict (i.e., learn) the next DM and then compute an optimal routing strategy for that DM. In machine learning terms, this translates to a supervised learning problem.

Supervised learning. A supervised learning task involves a sample space $\mathcal{X}$ and a labeling space $\mathcal{Y}$. An algorithm $\mathcal{A}$ for the task is a function mapping values in $\mathcal{X}$ to labels in $\mathcal{Y}$. Given a set of samples and their true labels $\{(x_i, y_i)\} \in \mathcal{X} \times \mathcal{Y}$, the goal is to identify a mapping that produces correct labels for new samples, drawn from the same distribution as the data. How good/bad a mapping fares is quantified in terms of a loss function $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. Intuitively, for any pair of labels (y_1, y_2), $\mathcal{L}(y_1, y_2)$ represents the cost of predicting, for a given sample, the label y_2 instead of the correct label y_1. See [40] for a detailed exposition of supervised learning.

We consider the following supervised learning approach to routing: the learning algorithm observes the history of DMs up to the current epoch, and predicts the DM for the upcoming epoch. This prediction is then used to generate an optimal routing strategy with respect to the predicted DM. When is employing this scheme a good idea? To answer this question, we evaluate several supervised learning schemes for predicting the next DM on different traffic patterns.

Generating DM sequences for our experiments. We next discuss how traffic patterns are generated in our experiments. We consider two standard schemes for generating DMs: the (deterministic) gravity model [41] and the (probabilistic) bimodal model [35]. Intuitively, the former captures scenarios in which communication between end-points is proportional to their outgoing bandwidths and the latter captures scenarios in which communication end-points are divided into small flows (mice) and large flows (elephants). We also consider "*sparsifications*" of gravity/bimodal DMs generated by selecting, uniformly at random, a p-fraction of the communicating pairs, for some choice of $p \in [0, 1]$, and removing the traffic demands of all other pairs from consideration. We refer to p as the *sparsity* of the DM.

Our experiments require generating *sequences* of DMs, specifying a DM for each time epoch. We examine two classes of DM sequences:

Class I: DM sequences in which the next DM is *deterministically* derived from past DMs. One example for such a DM sequence is "a cycle of DMs", in which the DM in each epoch belongs to a fixed set of q DMs, $D^{(0)}, \ldots, D^{(q-1)}$ such that if $D^{(j)}$ is the DM in epoch $t-1$ then $D^{(j+1 \ modulo \ q)}$ is the DM in epoch t. $D^{(0)}, \ldots, D^{(q-1)}$ in our experiments are sparsified (for varied values of p) gravity/bimodal DMs (for varied values for parameters of the bimodal model). Cycles of DMs might capture, e.g., the scenario that the traffic demands at a certain time of day are rather similar across days. See discussion of such temporal consistencies in ISP networks in [17]. Another example of a DM sequence that, though more artificial, also exhibits high regularity (and so is interesting to study)

is when each DM is the average of the previous q DMs (for some fixed $q > 0$). Our experiments evaluate supervised learning schemes on DM cycles of sizes $q = 5, 10, 15, 20$, and DM sequences in which each DM is the average over the previous $q = 5, 10, 15, 20$ DMs.

Class II: DM sequences in which each DM is independent of the previous DMs. The DM for each epoch is now drawn independently from a fixed probability distribution over DMs, namely sparsified gravity/bimodal DMs. We point out that such traffic patterns are commonly used in evaluations of data center architectures and protocols [25, 4, 20, 53] as traffic in data centers is often viewed as highly skewed and unpredictable [21, 18].

Supervised learning schemes. Following the recent successes of deep neural networks (DNNs) [28, 39, 42]), we evaluate 3 different DNN architectures. The input to all three architectures is the k most recent observed DMs and the output is a DM. We examine different values of k (5, 10, and 20). We use the Frobenius (or l_2) norm [22] to quantify the quality of an output with respect to the actual DM. The three architectures differ in the structure of the neural network interconnecting the input layer (representing k-long histories of DMs) and output layer (representing the next DM). We evaluate (1) *FCN*, a 3-layered fully-connected network, (2) *CNN*, a 4-layered convolutional neural-network [30], and (3) *NAR-NN*, a nonlinear auto-regressive model [11], realized via a 4-layered neural network that, for input demand matrices $D^{(1)}, \ldots, D^{(k)}$, learns a k-vector $\alpha = (\alpha_1, \ldots, \alpha_k)$ and an $n \times n$ matrix β, and outputs the DM $\sum_i \alpha_i D^{(i)} + \beta$.

Evaluation framework. We experiment with gravity and bimodal DMs of various sizes (9×9, 12×12, 23×23, 30×30, 50×50, and 100×100) under various choices of sparsity levels $p = 0.3, 0.6, 0.9, 1$ and of values of per vertex outgoing bandwidths (ranging from 10's of MB to 10's of GB). We consider various DM sequence lengths for training and testing the model (ranging from a few 10's to few 100's of DMs). We generate, for each choice of parameter assignment to q, p, k and sequence length, a training set of 10 DM sequences and a test set of 3 DM sequences. We define a *learning epoch* as a full traversal of the training set. We train each neural network for 2000 learning epochs.

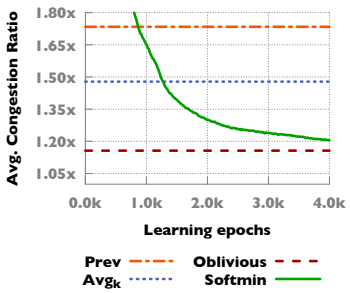
Results. Our experimental results (for the test DM sequences) show that for DM sequences that exhibit deterministic regularity, namely, cycles of DMs and "averaged DMs", only the *NAR-NN* performs fairly well and only for specific relations between the examined history (k) and the size of the cycle / number of DMs averaged over (q). Specifically, when $q \leq k$, *NAR-NN* well-approximates the next DM for cycles of DMs, and performs well on averaged DMs. *NAR-NN* continues to perform reasonably well on averaged DMs when $q > k$, but fails on cycles of DMs for $q > k$. All 3 architectures failed (quite miserably) to approximate the next DM for randomly generated DMs (which is not surprising, as there are no temporal correlations between DMs in the sequence).

We present representative results for *NAR-NN* on a network G with 30 vertices. We plot the loss, in terms of distance of the predicted DM from the actual DM (y-axis), over the number of learning epochs (x-axis). Figure 2b and Figure 2a show that the model succeeds in learning the next DM when using the averaged and cyclic DM sequence generation. Figure 2c demonstrates failure in learning the next DM when drawn from a probability distribution.

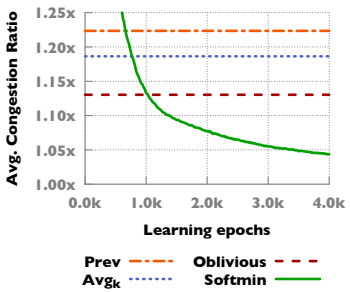
3.2 Reinforcement Learning Approach

We next investigate the application reinforcement learning to routing-strategy selection. Instead of attempting to learn the next DM and optimizing the routing strategy with respect to that DM, the goal now is to directly learn a good mapping from observed DMs to routing strategies.

Reinforcement learning. In the reinforcement learning framework, an agent repeatedly interacts with an environment. Time is divided into discrete time slots $t = 1, 2, 3, \ldots$. At the beginning of each time slot t, the agent observes the current *state* s_{t-1} of the environment and selects an action a_t from a fixed set of actions. Once the agent chooses action a_t, the state of the environment changes to s_t and the agent receives a *reward* r_t (a numerical value) signifying how good/bad the action he took was. The goal of the agent is to learn a mapping π from the set of possible states $\mathcal{S}$ to the space of actions $\mathcal{A}$ (i.e., $\pi : \mathcal{S} \rightarrow \mathcal{A}$) that fares well with respect to the objective of maximizing the *expected discounted reward* $\mathbb{E}[\sum_t \gamma^t r_t]$ for a predetermined $\gamma > 0$, called the *discount factor*. See [47] for a detailed exposition of reinforcement learning.



(a) *Congestion ratio for sparse ($p = 0.3$) gravity DM sequences*



(b) *Congestion ratio for non-sparse ($p = 1.0$) bimodal DM sequences with 40% elephant flows*

Figure 3: *Representative Results for $softmin$-Routing*

Routing via reinforcement learning. Routing-strategy selection can be easily cast as a reinforcement learning task as follows. At the beginning of each time epoch t, the operator/system (agent) decides on a routing strategy $\mathcal{R}^{(t)}$ for that epoch based on the routing strategies and DMs in the most recent k time epochs, which constitute the observed state of the environment at that point. Then, the state changes as the DM for epoch t, $D^{(t)}$, is revealed and the reward $r^{(t)} = -\frac{u^{(t)}}{OPT^{(t)}}$ is received, where $u^{(t)}$ is the max-link-utilization under $\mathcal{R}^{(t)}$ for $D^{(t)}$ and $OPT^{(t)}$ is the optimal max-link-utilization with respect to $D^{(t)}$ ($r^{(t)}$ thus captures the ratio between achieved performance and optimal performance). The goal is to learn a mapping from k-long histories of DMs to routing strategies that maximizes the expected discounted reward, as formulated above. We explore the power of this approach in the following sections.

4 Representing the Output

We observe that, in contrast to other recent applications of ML to networking [33, 34], learning routing strategies involves generating neural networks with very large output layers (containing, e.g., thousands of output nodes even for a communication network of but tens of vertices). Consider, e.g., the representation of a routing strategy described in Section 2. This representation involves $|V|^2 \cdot |E|$ variables, where $|V|$ and $|E|$ are the sizes of the network graph's vertex set and edge set, respectively. We show below that even for constrained routing strategies of much smaller sizes, a ("vanilla") reinforcement learning approach for predicting the complete routing strategy fails to attain good performance within reasonable time.

We restrict our attention to *destination-based* routing strategies, i.e., routing strategies in which the splitting ratios at each vertex u with respect to any destination d are the same across all possible sources s. Observe that any such routing strategy $\mathcal{RS}$ can be represented by $|V| \cdot |E|$ values (i.e., $|V|$ times smaller than unconstrained routing strategies). We use a state-of-the art continuous-control reinforcement-learning algorithm, TRPO [42] (also used in [33]), applied to a 3-layered fully-connected neural network, to learn the mapping π from k-long histories of DMs to a routing strategy $\mathcal{RS}$. The real-valued outputs generated by the deep neural network are turned into per vertex traffic-splitting ratios by applying, for each vertex in the communication network u, the softmax function [9] to the outputs corresponding to u's outgoing edges.

Evaluation. We adapt the open-source implementation of TRPO [42] provided by OpenAI [13] to the task of learning routing-strategies. We begin our evaluation with a seemingly easy target: learning the splitting ratios for a 12-vertices, 32-edges network (taken from [26]), and (sparsified) gravity DMs. We train a 3-layered fully-connected network over 7 sequences of gravity DMs of length 60 and evaluate (test) the neural network on 3 such sequences. We repeat this process for sparsity levels $0.3, 0.6$, and 0.9. We use $k = 10$ (the length of the history of past DMs received as input). We compute the optimal congestion using the CPLEX [2] LP solver.

The training phase involves generating, from every sequence of DMs of length 60, 50 sequences of 10 consecutive DMs (representing ten-long histories of DMs), by grouping DMs $1 - 10$, $2 - 11$,

etc. Training the neural network on each of these "histories of DMs" involves evaluating the neural network 30 times in parallel (and so $1,500$ iterations per DM sequence and $10,500$ overall). We refer to one execution of this process as a "learning epoch". Our results (omitted due to space constraints) suggest that this approach leads to slow and ineffective learning; e.g., even after more than 700 learning epochs, the produced routing strategies were still over 9x away from the optimum, in terms of max-link-utilization. As shown below, routing strategies that fare significantly better can be generated much quicker. We hypothesize that the large number of output parameters renders efficient learning very challenging. We thus seek a class of routing policies that can be more concisely represented yet is still rich enough to attain high performance.

5 Learning Softmin Routing

We explore the following approach: instead of learning splitting ratios directly, learn *per-edge weights*, and then use these weights to generate a routing strategy. Under this approach, the output of the neural network is of size $|E|$ (as opposed to $|V|^2 \times |E|$ and $|V| \times |E|$ for unrestricted and destination-based routing policies, respectively).

Generating forwarding rules from link weights is a classical approach to routing [16, 52, 36]. We resort to the following approach: The $softmin_\gamma$ value for a vector of r coordinates $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_r)$, for $\gamma > 0$, is the vector (also of r coordinates) $softmin_\gamma(\alpha)_i = \frac{e^{-\gamma\alpha_i}}{\sum_{i=1}^{r} e^{-\gamma\alpha_i}}, \quad i \in 1, \ldots, r.$ Observe that $softmin(\alpha)$ can be regarded as a probability distribution (as the sum of all coordinates necessarily equals 1).

Consider a specific assignment of per-edge weights $w = \{w_e\}_{e \in E}$, a specific edge $(u, v) \in E$, and a specific traffic destination d. Observe that, when viewing weights as distances, w determines the length of the shortest path from vertex u to vertex d that goes through u's immediate neighbor v. Let $SP_w(v, u, d)$ denote this length. Given a set of such per-neighbor distances for a vertex u, the $softmin$ function can be applied to generate a probability distribution across these neighbors, which can be interpreted as u's splitting ratios for traffic destined for d. We refer to this scheme as "*softmin-routing*". The higher the choice of γ to plug into the softmin function the closer the resulting routing scheme is to shortest path routing. We set $\gamma = 2$ in our experiments.

Our reinforcement learning scheme maps k-long histories of DMs to per-edge link weights. The reward is computed by turning these weights into traffic splitting ratios and computing the max-link-utilization of the resulting routing strategy with respect to the next DM. We realize this learning scheme via a 3 layers fully-connected network.

We benchmark our results against three alternative non-ML-based approaches to computing routing strategies: (1) $Prev$: optimizing softmin routing with respect to the most recent DM, (2) Avg_k, optimizing softmin routing with respect to the k most recent DMs, and (3) $Oblivious$, the optimal oblivious routing scheme [7] (which does not depend on the history of DMs at all). [1]

Evaluation. We consider a communication network with 12 vertices and 32 edges. We use the adaptation of [13] discussed in Section 4 to train a 3-layered fully-connected neural network to generate the weights for softmin routing. We train the neural network on 7 sequences of gravity and bimodal DMs of length 60 each, and tested on 3 such sequences. For gravity DM, the above process is repeated per sparsity levels $0.3, 0.6$, and 0.9. For bimodal DMs the sparsity level is $p = 1$ and the percentage of large (elephant) flows amongst the communicating pairs is varied: $20/40/60\%$ of all pairs. We set $k = 10$ and compute the optimal flow via the CPLEX [2] LP solver.

We show in Figure 3a representative results for gravity DMs and in Figure 3b representative results for bimodal based DMs. The figures plot the ratio between the performance of the resulting routing strategies (for the test DM sequences), in terms of averaged max-link-utilization (congestion), and the optimum congestion. Interestingly, oblivious routing outperforms the other two baselines. Observe that $softmin$-routing gets very close to oblivious routing's performance for gravity DM sequences

[1] Observe that both $Prev$ and Avg_k optimize softmin routing, as opposed to applying the optimal multicommodity flow computed for the input DM(s) to route the next DM. The reason is that the latter option is not well defined (and, in particular, some of the end-points communicating in the next DM might not communicate at all in the input DM). We point out that in our experimentation softmin routing is consistently within at most 5% of the optimum traffic flow with respect to any input DM and so very closely approximates this strategy.

(and could perhaps outperform it with more training), and significantly outperforms oblivious routing for bimodal DM sequences.

6 Related Work

Traffic engineering Traffic engineering (TE) is the optimization of routing withing a single, self-administered network. TE is fundamental to networking, and hence vastly researched. Results on TE range from routing in legacy networks [16, 10] to datacenter networks [3] and backbone networks [24]. Softmin routing is inspired by the literature on TE via hop-by-hop routing in IP networks e.g., PEFT [52] and HALO [36]. We find softmin routing especially convenient to use as it involves fairly simple splitting traffic across next-hops, while still achieving high performance.

Reinforcement learning. Machine learning via deep-neural networks has proven extremely useful in executing many different tasks: machine translation [8], image recognition [28], and more. Specifically, reinforcement learning has been applied to playing computer games [38] and beating world-champions in strategic board games [45], robotics [27], 3D-locomotion tasks [42], and beyond. The development and optimization of reinforcement learning algorithms is thus the subject of much attention. Our algorithms rely on utilizing TRPO [42]. We leave the evaluation of other reinforcement-learning algorithms [37, 43, 51] to future research.

ML applications to networking. Machine learning has been applied to various networking contexts including congestion control [50, 12], network bottleneck detection [48], and optimizing datacetner power consumption [1], resource allocation [33], and bitrate selection for video streaming [34]. Q-routing [32] applies Q-learning [49] to the network routing context. Under Q-routing [32], each router individually learns a mapping from packet headers to outgoing ports. This involves routers constantly exchanging information, at per packet resolution, about their latencies with respect to different destinations. Operating at per packet level, and in a decentralized fashion, poses significant challenges in terms of scalability and communication overhead.

7 Conclusion

We initiated the study of data-driven routing. Our preliminary results from experimentation with deep reinforcement learning show that extracting information from the history of traffic scenarios to generate good routing with respect to future traffic scenarios is an interesting approach. We view our results as a first step towards realizing a much broad research agenda.

Other routing domains. Optimizing routing is a keystone of networking research, investigated in a broad variety of contexts, including legacy IP networks [16], data centers [3], private backbone networks [24], overlay networks [6], publish-subscribe networks [14], and more. Applying a data-driven routing approach to other settings is an important research direction.

Other objective functions. Our focus in this study was on the classical objective of minimizing max-link-utiliziation. Examining other well-studied multicommodity-flow-based objectives, e.g., maximizing overall goodput, is of great interest, as is investigating performance metrics that relate to latency, flow-completion-time, etc.

Traffic-demands prediction. Our preliminary results suggest that well-predicting traffic conditions can, in general, be very challenging. This motivates further research on supervised learning approaches to this challenge.

Better ML-guided routing. Our investigation of the application of ML to routing is only a first step in this direction. Important questions remain regarding (1) the scalability of ML approaches in this context, (2) the environments in which ML-guided routing outperforms traditional routing, and the causes for this, and (3) the "right" choice of the duration of the time epoch be to strike the right balance between routing stability and reactiveness to traffic changes.

Better experimental and empirical evaluations. Our experiments involved fairly small networks and synthetically-generated traffic demands. Evaluating routing solutions in more realistic scenarios is important.

References

[1] DeepMind AI Reduces Google Data Centre Cooling Bill by 40%. https://goo.gl/QTdU2T.

[2] IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html.

[3] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, 2010.

[4] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). *ACM SIGCOMM*, 2010.

[5] M. Allalouf and Y. Shavitt. Maximum flow routing with weighted max-min fairness. In *Quality of Service in the Emerging Networking Panorama*.

[6] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. 2001.

[7] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Racke. Optimal oblivious routing in polynomial time. STOC, 2003.

[8] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv*, 2014.

[9] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2006.

[10] M. Chiesa, G. Rétvári, and M. Schapira. Lying your way to better traffic engineering. CoNEXT, 2016.

[11] T. W. S. Chow and C. T. Leung. Nonlinear autoregressive integrated neural network model for short-term load forecasting. *IEE Proceedings - Generation, Transmission and Distribution*, 1996.

[12] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira. PCC: Re-architecting congestion control for consistent high performance. NSDI, 2015.

[13] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. ICML, 2016.

[14] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 2003.

[15] B. Fortz, J. Rexford, and M. Thorup. Traffic engineering with traditional ip routing protocols. *Comm. Mag.*, 2002.

[16] B. Fortz and M. Thorup. Increasing internet capacity using local search. *Computational Optimization and Applications*, 2004.

[17] B. Fortz and M. Thorup. Optimizing ospf/is-is weights in a changing world. *IEEE J.Sel. A. Commun.*, 2006.

[18] M. Ghodbadi, R. Mahajan, A. Phanishayee, H. Rastegarfar, P.-A. Blanche, M. Glick, D. Kilper, J. Kulkarni, G. Ranade, and N. Devanur. ProjecToR: Agile Reconfigurable Datacenter Interconnect. *SIGCOMM*, 2016.

[19] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[20] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall. Augmenting data center networks with multi-gigabit wireless links. SIGCOMM, 2011.

[21] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, and A. Tanwer. Firefly: A reconfigurable wireless data center fabric using free-space optics. *SIGCOMM*, 2014.

[22] K. Janocha and W. M. Czarnecki. On loss functions for deep neural networks in classification. *CoRR*, 2017.

[23] W. S. Jewell. *Multi-commodity Network Solutions*. 1966.

[24] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the tightrope: Responsive yet stable traffic engineering. 2005.

[25] S. Kassing, A. Valadarsky, G. Shahaf, M. Schapira, and A. Singla. Augmenting data center networks with multi-gigabit wireless links. SIGCOMM, 2011.

[26] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 2011.

[27] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 2013.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[29] P. Kumar, Y. Yuan, C. Yu, N. Foster, R. D. Kleinberg, and R. Soulé. Kulfi: Robust traffic engineering using semi-oblivious routing. *CoRR*, 2016.

[30] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[31] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[32] M. Majer, C. Bobda, A. Ahmadinia, and J. Teich. Packet routing in dynamically changing networks on chip. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 3 - Volume 04*, IPDPS '05, pages 154.2–, Washington, DC, USA, 2005. IEEE Computer Society.

[33] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. Resource management with deep reinforcement learning. HotNets, 2016.

[34] H. Mao, R. Netravali, and M. Alizadeh. Neural adaptive bitrate streaming with pensive. SIGCOMM, 2017.

[35] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation: Existing techniques and new directions. 2002.

[36] N. Michael and A. Tang. Halo: Hop-by-hop adaptive link-state optimal routing. *IEEE/ACM Transactions on Networking*, 2015.

[37] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. ICML, 2016.

[38] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, 2013.

[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[40] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012.

[41] M. Roughan, A. Greenberg, C. Kalmanek, M. Rumsewicz, J. Yates, and Y. Zhang. Experience in measuring backbone traffic variability: Models, metrics, measurements and meaning. IMW, 2002.

[42] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. ICML, 2015.

[43] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *ArXiv e-prints*, 2017.

[44] F. Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *J. ACM*, 1990.

[45] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 2016.

[46] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[47] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[48] H. Wang and B. Li. Lube: Mitigating bottlenecks in wide area data analytics. HotCloud, 2017.

[49] C. J. Watkins and P. Dayan. Q-learning. *Machine Learning*, 1992.

[50] K. Winstein and H. Balakrishnan. Tcp ex machina: Computer-generated congestion control. SIGCOMM, 2013.

[51] Y. Wu, E. Mansimov, S. Liao, R. B. Grosse, and J. Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *ArXiv e-prints*, 2017.

[52] D. Xu, M. Chiang, and J. Rexford. Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering. INFOCOM, 2008.

[53] X. Zhou, Z. Zhang, Y. Zhu, Y. Li, S. Kumar, A. Vahdat, B. Y. Zhao, and H. Zheng. Mirror mirror on the ceiling: Flexible wireless links for data centers. SIGCOMM, 2012.