Demystifying Deep Learning in Networking[*]

Ying Zheng, Ziyu Liu, Xinyu You,
Yuedong Xu
School of Information Science and Technology
Fudan University, Shanghai, China
ydxu@fudan.edu.cn

Junchen Jiang
Department of Computer Science
University of Chicago
Chicago, Illinois
junchenj@uchicago.edu

ABSTRACT

We are witnessing a surge of efforts in networking community to develop deep neural networks (DNNs) based approaches to networking problems. Most results so far have been remarkably promising, which is arguably surprising given how intensively these problems have been studied before. Despite these promises, there has not been much systematic work to understand the inner workings of these DNNs trained in networking settings, their generalizability in different workloads, and their potential synergy with domain-specific knowledge. The problem of model opacity would eventually impede the adoption of DNN-based solutions in practice. This position paper marks the first attempt to shed light on the *interpretability* of DNNs used in networking problems. Inspired by recent research in ML towards interpretable ML models, we call upon this community to similarly develop techniques and leverage domain-specific insights to demystify the DNNs trained in networking settings, and ultimately unleash the potential of DNNs in an explainable and reliable way.

CCS CONCEPTS

• **Networks → Cloud computing**; **Network management**;

KEYWORDS

Neural networks, Resource allocation, Interpretability

ACM Reference Format:
Ying Zheng, Ziyu Liu, Xinyu You, Yuedong Xu and Junchen Jiang. 2018. Demystifying Deep Learning in Networking. In *APNet '18: 2nd Asia-Pacific Workshop on Networking, August 2–3, 2018, Beijing, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3232565.3232569

1 INTRODUCTION

Deep neural network (DNN) based approaches are gathering momentum in the networking community. Studies from academia and industry have shown remarkably promising improvements by DNN-based solutions (e.g., deep reinforcement learning) in a variety of classic problems, such as cloud resource allocation [10], end-to-end video adaptation [11], routing [17], wireless bandwidth allocation [18], and so forth. It will be no surprise to see DNNs similarly applied to other networking problems.

By building solutions around DNNs, this recent line of research signifies a potential *paradigm shift*, from the traditional "white-box" approach (which is driven by domain-specific knowledge) to a "black-box" one (which relies on opaque data-driven models).[1] On one hand, not only have these DNN-based solutions substantially improved performance along important metrics, but they suggest a general approach to a wide range of systems problems that have traditionally been tackled in isolation using hand-crafted solutions. On the other hand, however, DNN-based approaches can backfire for its lack of *interpretability*. The inability to understand these deep models lies at the root of a multitude of concerns; e.g., it is difficult to completely trust the model will perform well in new environments, to debug why a wrong decision was made, and to defend it against adversarial manipulation.

Moreover, unlike other machine-learning problem domains, networking/systems problems (e.g., routing, scheduling) have explicit structures and constraints that have traditionally inspired interpretable domain-specific, albeit suboptimal, solutions. Unfortunately, this first-principles approach is incompatible with opaque models like DNNs. This prevents any synergy between DNNs and domain-specific knowledge, which could otherwise achieve the best of both worlds.

Inspired by recent efforts in ML community towards inter -pretable ML [1, 8], this paper calls upon the networking community to similarly enable interpretation of DNNs trained

[*] This work is supported by Natural Science Foundation of China (No. 61772139) and CERNET Innovation Project (No. NGII20170209).

[1] The increased interests in complex black-box solutions coincide with the trend of harnessing the power of massive measurement data and building models in a "big data" fashion. While both are under the aegis of data-driven approaches, we focus more on issues of black-box solutions, rather than data-driven modeling, which merits a separate discussion on its own.

APNet '18, August 2–3, 2018, Beijing, China
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6395-2/18/08…$15.00
https://doi.org/10.1145/3232565.3232569

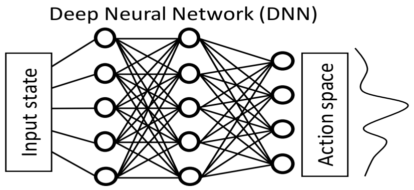


Figure 1: ***Use of DNN in decision making.***

in networking/systems settings. We review the recent progresses and sketch a research roadmap driven by three questions, each elucidating a key aspect of DNNs for networking.

- *How is a decision made?* (§3) The ability to explain the decision-making process sheds light on the inner workings of an ML model and the implicit hypothesis behinds it, and is a crucial step towards model interpretability.
- *When will it fail?* (§4) For a DNN to be deployed in real world, one must understand its worst cases, whether it will perform well in different environments, and whether it is vulnerable to adversarial inputs.
- *Can domain-specific insight be integrated?* (§5) Many networking problems have expert-crafted solutions, so we naturally want to see if they can improve DNNs or if new insights can be learned from DNNs.

2 WHY IS INTERPRETABILITY CRUCIAL?

We start with some background on DNNs used in networking settings and then motivate why interpretability is a crucial yet missing pieces these DNNs.

2.1 Early promises of DNNs

Many DNN-based solutions are introduced on the premise that performance optimization in networked systems faces enormous complexities: multiple networked subsystems operating independently but heavily influencing each other (e.g., routing), the need to balance between multiple seemingly conflicting performance metrics (e.g., how to simultaneously minimize playback stalls and maximize video resolution), and complex relations between control actions and their outcomes (e.g., congestion control). Traditional rule-based approaches are ill-suited to cope with these complexities; instead, DNNs' high complexities and model expressiveness make them a better fit.

Indeed, DNN-based solutions, such as deep reinforcement learning, have already shown promising improvement across several networking problems. Table 1 summarizes its uses in at least four problems. Each problem is formulated as a reinforcement learning process, in which the key step is to use a DNN to map an input "state" to a probability distribution over "actions". Each action triggers a reward (e.g., job slowdown), and the DNN updates its parameters to maximize expected reward until they converge.[2] Across all these problems, the DNN-based solutions have seen substantial improvement over state-of-the-art solutions (Table 1).

Problem	Input states	Output action	Example gains
Cloud job scheduling [10]	Unused resources, new jobs' demands	Next job to schedule	32.4% less job slowdown
Adaptive-bitrate streaming [11]	Throughput history, buffer length, available bitrates	Bitrate selection	12-25% better QoE
Routing [17]	Traffic matrix history	Routing plan	23-50% less congestion
Cellular traffic scheduling [18]	Traffic demand, cellular condition, congestion, etc	Traffic rate assignment	14.7% higher utilization

Table 1: ***Early promises of DNNs in networking.***

2.2 Concerns of treating DNNs as blackboxes

Despite the early promises, these DNN-based solutions are extremely hard to interpret with intuitive explanations. Comprising of tens of thousands (or more) of parameters all affecting output in complex, non-linear ways, the inner workings of a DNN are hard to explain, even by their developers. Here, we identify three problems resulting from the absence of interpretability of these DNNs in networking/systems.[3]

Hard to confer causality: Some previous work did attempt to explain how a trained DNN works by hypothesizing intuitive explanations and empirically check if they match the DNN's behavior on test data. However, *this approach* merely suggests statistical association, not causality from an input to an output. Without a proper understanding of the causality, we will not be able to trust if DNN works the same way we believe it does.

Hard to trust: When mistakes have high-stake consequences, people tend to distrust a decision-making process, if they do not understand its worst-case performance and generalizability. In this sense, we not only care *how often* the model is right, but *when* the model is right. DNNs are known to be vulnerable to adversarial inputs. In contrast, more interpretable models are easier to be trusted, because it allows for *post hoc* debugging, which explains why system developers and administrators often favor white-box models over black-box ones when they have similar performance [7].

Incompatible with domain-specific knowledge: For many systems problems, domain-specific knowledge is abundant. It could help put DNNs and their gains into perspective, make them more reliable, and reduce model complexity without sacrificing performance. However, the opacity of DNNs makes incorporating domain-specific knowledge difficult. In all DNN-based solutions so far, once the system states and decisions are encoded as the intput/output, the training of DNN is completely agnostic to domain-specific knowledge, making it hard to discern whether the performance gains result from a new insight or a better way of using some known

[2] We use input "features" interchangeably with "state", and output "decision" interchangeably with "action".

[3] We mainly focus on the intepretability of the DNNs in deep reinforcement learning, not that of reinforcement learning.

insights. Conversely, this prevents any synergies between DNNs and domain knowledge.

Our roadmap: As DNNs gain momentum in networking research, the interpretability of DNNs is urgently needed. Next, we will decompose interpretability into three questions which constitute the research agenda that we are pursuing.

3 WHY DNN MAKES THE DECISION?

We begin by asking "*why* does DNN make certain decision?" The ability to explain a decision is the prerequisite of other properties of interpretability. As simple as it may sound, answering it can be tricky but deeply revealing.

3.1 Important features

Before showing how an input is turned into an output, we first ask what input features are most influential. The input of a DNN often includes all features that might be useful in the decision-making. The idea is that irrelevant features will be automatically "turned off" during training, so using more features should only improve its expressiveness without hurting the performance (if trained on enough data). But it complicates the interpretation of DNN, since many features are not used or have negligible impact on the output.

To reveal what features a DNN depends on, a popular technique is to use *saliency map* [15], which is a heatmap showing how much impact of each input feature on the output. Here is a simple way of generating such a saliency map for a given DNN. Let us use $\mathbf{x}$ to denote the input feature vector (state), $\mathbf{y}$ the output vector of probability distribution over possible actions, $\mathbf{y} = f(\mathbf{x})$, and y_i the probability of choosing the i^{th} action. We can then compute the gradient vector $\mathbf{w}_i$ of y_i to a given input vector $\mathbf{x} = \mathbf{x}_0$ with $\mathbf{w}_i = \frac{\partial y_i}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\mathbf{x}_0}$. The j^{th} element of $\mathbf{w}_i$ shows how much a small change on the j^{th} feature of $\mathbf{x}$ will change how likely i^{th} action is picked. These gradients indicate each feature's influence on the output[4]. When a DNN comprises convolutional layers, the Grad-CAM [14] method can be employed to generate the saliency map.

We use the DNN-based logic of DeepRM as a case study, and compute the saliency map to visualize the influence of different input features on its output. The DNN was trained on a sequence of jobs to minimize average job slowdown (job's duration in the system divided by its actual processing time) using the open source code [10]. Figure 2 shows the saliency map of a specific output action (choosing the first job in the queue) with respect to an example input state. In an input state. the horizontal and vertical coordinates indicate the resource slices and the time units respectively. Each pixel is a binary value (1 means the resource is needed/occupied, and 0 otherwise) in the input states. The CPU (or RAM) state is captured by four maps: the leftmost map shows the CPUs (or RAM) occupied by the jobs that have been scheduled and it scrolls up row-by-row after each time slot; the other three maps illustrate the CPU (or RAM) resource demands of the jobs awaiting for scheduling. A negative gradient is visualized in blue and a positive gradient is visualized in red at the saliency map, and the depth of a color represents the relative value of this gradient. So if the resource demand of a job is labeled dark red (or blue) in the saliency map, it means increasing the resource demand of that job will greatly increase (or reduce) the chance of making the same decision. For instance, if the CPU or memory demand per time cycle increases at job #3, more red pixels are covered, leading to a higher probability of choosing job #1; if the CPU demand per time cycle increases at job #2, the chance of scheduling job #1 decreases. Recall that one saliency map only visualizes the impact of input states on a specific output action, so even if the sum of the calculated values on a saliency map increases, it does not necessarily lead to the job being selected.

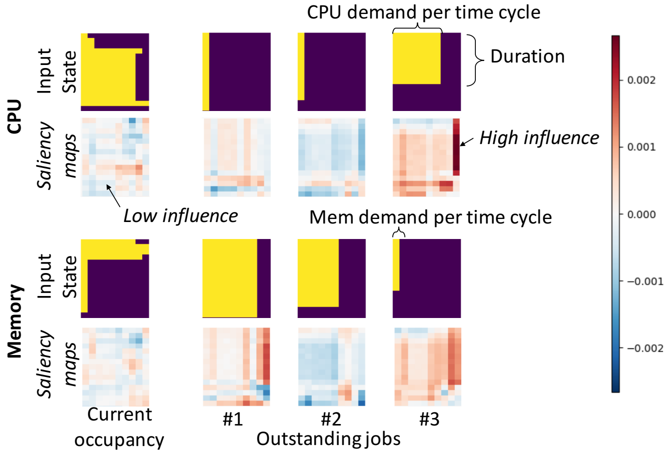


Figure 2: ***An example input state of DeepRM DNN, and the corresponding saliency map indicating the influence of each input feature on the probability of choosing one action. (Some states are omitted for simplicity.)***

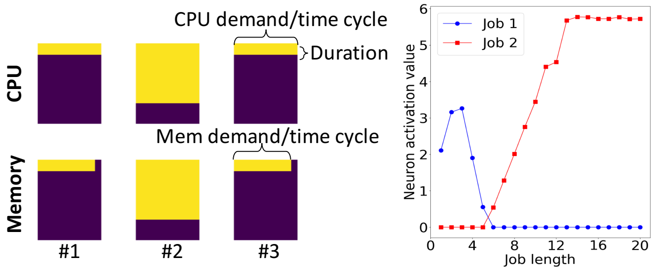
From the figure, we can see two observations.

- The current CPU/RAM occupancy has relatively small impact on the output. Even when their gradients are of non-zero values, the positive ones and negative ones cancel out each other's impact.
- Some of the outstanding jobs always have high impact on the output. (Some jobs have dark-color regions.)

3.2 From input to high-level features

Knowing which features influence the output is useful, but it would be more informative if we could explain *how* they influence it. This requires "opening up" the DNN to understand what *high-level representations* are being used. Given

[4] Two caveats need to be noted when interpreting a DNN saliency map. First, the map only indicates "local" interpretation; that is, the may vary across different inputs. Second, gradient-based saliency maps of DNNs suffer from so called "gradient shattering", that is substantial noise as the gradients propagate through the layers in a DNN, which could be mitigated with more refined techniques [2].



(a) State maximally activating a neuron

(b) Impact of job length on neuron activation

Figure 3: *Visualizing what high-level representations a particular intermediate neuron focuses on. We see (a) regular patterns (CPU/-intensive, long or short jobs) in the state that maximally activates a neuron, and (b) threshold effect of job length on neuron activation.*

NN's inherent multi-layer structure, a natural approach is to examine the activation of *intermediate neurons* in hidden layers [19]. The basic intuition is that the influence of any input feature on the final DNN output must through activating (or not activating) some intermediate neurons in the hidden layers. We then visualize this mapping from two aspects.

Maximally activating intermediate neurons: First, we can look at the input that maximally activates a neuron. The reasoning behind it is that a pattern of input to which the neuron is responding maximally could be a good first-order representation of what a neuron is doing. This is the basic insight behind much recent work to examine DNN's high-level representations [3]. Take DeepRM (three layers with a fully-connected hidden layer of 20 neurons) as an example. Figure 3(a) shows the input state that maximizes the gradient of one of the 20 intermediate neurons. To search the maximum activation, we enumerate the resource occupancy of each input map separately.

We can see that the neuron is maximally activated when one of the 2nd job are long, and 1st and 3rd are short. Moreover, neuron is maximally activated by high CPU and RAM requirements of a job with a long duration. We observe similar phenomenons across all intermediate neurons (the placement of long/short jobs varies), which suggests that the intermediate neurons probably look at the duration of the outstanding jobs. To verify this speculation, we use the second idea.

(In)distinguishable input: The second intuition is that two inputs are distinguishable, only if they lead to different activations on a certain neuron. To illustrate it, we run a simple experiment: Figure 3(b) plots the activation of a neuron (the same neuron as in Figure 3(a)) as a function of the length of each job (here, we use the 1st and 2nd job in Figure 3) while setting other jobs empty. We can see that the 2nd job does not make any difference on the neuron when it is over 15 cycles, and always deactivates the neuron when less than 5 cycles; and different job lengths matter only between 5 and 15 cycles. In contrast, the 1st job's impact peaks at a length of 3 cycles, and effectively deactivates the neuron when the length is over 6 cycles. In short, the job length does have a threshold effect on neuron activation. This observation holds across all intermediate neurons.

These observations suggests that we might be able to *decompose* of the decision-making process into high-level representations that can be studied separately. Instead of testing the properties of a DNN in a black-box manner, we could now explain *why* there might be such a correlation. For instance, it is believed that DeepRM outperforms alternatives because it can hold a large job even if there is enough available resources to run it. A black-box approach can establish a statistical correlation between selecting a job and the job size. In contrast, by examining the intermediate neurons, we can actually explain correlation by DNN works internally: the intermediate neurons are only activated when the job size is over or below certain thresholds, i.e., the neurons effectively act as a "filter" of job size. Such interpretation is more generalizable and reliable.

3.3 Research thrusts

We summarize the section by posing an overarching question, which can drive future research in this direction:

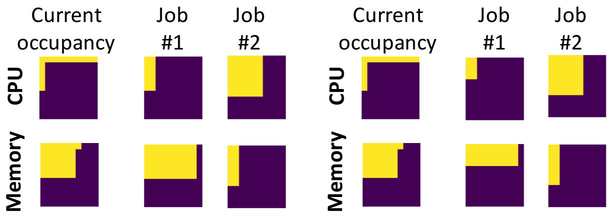
> *Can we explain a DNN's inner working by domain-specific knowledge in a white-box manner?*

Instead of treating DNNs as black boxes and testing hypothesis statistically, a white-box explanation strives to explain a DNN's decision from first principles: ideally, we want to map DNN's internal mechanism step-by-step to intuitive explanation intelligible to domain experts. This vision can open up many possibilities, including post-hoc debugging, and transparent logic that imitates DNN's behavior. An ideal answer to this question would require to completely reverse-engineer the DNNs, which unfortunately is believed to be impossible [16] in a general case. Nonetheless, ML communities have developed a plethora of techniques that could decode specific structures in DNNs [9].

Where domain knowledge is needed: Interpretation of DNNs does require domain knowledge: what domain-specific pattern is the DNN responding to, what logic is the DNN approximating, etc. Automatically generating and testing these domain-specific hypothesis will require close collaboration between domain experts and ML experts. Fortunately, our preliminary experiments in this section have suggested that it is indeed plausible to test domain-specific hypothesis by visualizing/interpreting features and high-level representations of DNNs used in networking/systems settings.

4 WILL IT FAIL MISERABLY?

DNNs do improve average performance, but meanwhile they are more vulnerable to adversarial or unseen input than alternative rule-based models. Here, we show that DNNs trained for networking problems suffer from the same weakness.



(a) Original job sequence in training

(b) Adversarial job sequence: Shortening Job #1 of (a)

Figure 4: ***An adversarial example of DeepRM, created by slightly decreasing the length of job #1. In both cases, the best scheduling order is first #2 then #1. But DeepRM will schedule job #1 first, which increases the average slowdown from 4.3 time unit (#2→#1) to 4.95 (#1→#2). This is because the slightly shorter length of job #1 "tricks" DeepRM into believing it is a short job which need be prioritized.***

4.1 Adversarial input states

Adversarial examples of a model are those that differ only slightly to a seen example in the training data but can "trick" the model into making arbitrary wrong decisions. Generating adversarial examples for DNNs has been studied as a useful tool to understand the robustness and corner cases in many DNN applications, e.g., image classification [12].

DNNs in systems is of no difference. We create a simple adversarial example of the resource allocation DNN in DeepRM by randomly manipulating the time duration of a job in Figure 4. In both job sequences, the 2nd job should be processed first, because the dominant resource of 1st job (memory) will completely block the use of a resource type [5]. By reducing the length of job #1, we can trick DeepRM into believing the job is a short job and thus should be prioritized, because it has not seen many short jobs that require more dominant resources.

Generative Adversarial Networks (GANs) are commonly used to generate adversarial examples [16]. The application of GANs to DeepRM is promising, yet difficult because the generated input state needs to be meaningful, e.g. adjacent CPU and RAM occupancy, and simultaneous execution of jobs in CPU and RAM. Indeed, creating adversarial examples with restrictions remains to be an open problem in machine learning communities.

4.2 Heterogeneous environments

Control logic in systems often need to operate under heterogeneous environments and workloads, especially when the logic cannot be updated easily (e.g., hardwired in hardware, or hard-coded in software application). Conventional rule-based logic is amenable to analysis of would perform in a different environment, so even if the logic cannot be changed, developers still have the ability to predict failures or degradation. In contrast, machine learning-based algorithms can optimize performance for workloads drawn from the same distribution as the training data, but for the algorithm to carry over to a different environment requires the additional *transferability*. Unfortunately, DNNs are generally believed to have poor transferability: a DNN can fail miserably when tested against data draw from an unseen distribution, even when it is simple enough to be handled by simpler alternative models [16]. The DNN's opacity also makes it difficult to understand its transferability.

In our experiments over DeepRM, we found it is relatively easy to generate a job sequence that makes it fail miserably by changing the parameters of the generative model of training data. DeepRM may perform even worse than the shortest-job-first (SJF) strategy when tested against jobs of unseen distributions of job sizes or arrival intervals. We believe other DNN systems in Table 1 will underperform if the test environment deviates from the training environment.

In fairness, previous work did strive to ensure DNNs are trained on representative workload and test their generalizability under different workloads [11]. In fact, one advantage of high-dimensional models like DNNs is its ability to train on a large amount of data from different sources. Despite these "best effort" practices, we still need a systematic approach to understanding the transferability of DNNs trained in networking settings.

4.3 Research thrusts

It is crucial that before deploying any DNN-based decision-making logic, we know its adversarial examples and in what environment will the DNN fail. We conclude this section by the the following question.

> *Can we stress test any DNNs in networking/systems by automatically generating test examples inspired by real networking/systems scenarios?*

Mathematically, the process of creating adversarial examples is fairly intuitive. Given an input $\mathbf{x}$, a DNN that maps $\mathbf{x}$ to an output $\mathbf{y} = f(\mathbf{x})$, and any target output $\mathbf{y}'(\neq \mathbf{y})$, we can create an adversarial example by finding $\mathbf{d}$ that minimizes $||\mathbf{d}||$, subject to $f(\mathbf{x}+\mathbf{d}) = \mathbf{y}'$, i.e., we can create an adversarial example $\mathbf{x}+\mathbf{d}$ that tricks the DNN to make any decision $\mathbf{y}'$ by making a small change $\mathbf{d}$ on a seen input. Generating DNN adversarial examples has attracted much attention in ML community, but most techniques [13] are derived from the aforementioned formulation.

Where domain knowledge is needed: Unfortunately, the above formulation cannot be directly used to generate adversarial examples that make sense in networking. In image classification, adding pixel-level noises to an image is unlikely to alter its real label, i.e., the real label of $\mathbf{x}+\mathbf{d}$ is not $\mathbf{y}'$. But changing a systems state even slightly may fundamentally change the best decision, so the generated example $\mathbf{x}+\mathbf{d}$ may well map to the targeted action $\mathbf{y}'$, which means the adversarial example is not adversarial after all. Moreover, we need domain knowledge to make sure the adversarial examples are "meaningful", rather than contrived cases that will rarely occur in practice.

5 MACHINE-GENERATED ALGORITHMS TO RULE IT ALL?

So far our discussion has focused on DNN itself. Similar to other domain-agnostic methods, DNNs can benefit from learning from domain-specific knowledge. In this section, we introduce a method in networking and systems to illustrate how integrating DNNs with domain knowledge can improve their transferability, robustness, and training efficiency.

5.1 Integrating domain knowledge is the key

Many weaknesses of DNN-based solutions can be mitigated using domain-specific knowledge.

Better transferability: By nature, domain-specific policies will not be biased by training data, so domain-specific policies can potentially help prevent DNNs from overfitting the training data that are not representative for the targeted environment. For instance, one can prevent DeepRM from overfitting the training workload by setting a new objective that maximizes rewards on the training workload while imitating a more generalizable domain-specific policy (e.g., shortest job first).

Better robustness: In systems problems, the penalties of making certain decisions is not immediately observable or directly measurable, so machine-learning strategies like deep reinforcement learning cannot straightforwardly minimize these penalties by explicit objectives. For instance, large-scale systems often maintain resource utilization below certain threshold (e.g., 80%) to tolerate busty traffic or component failures, but these cases are so rare that can easily be ignored or "averaged out" by DNNs optimizing for the mean performance.

Better training efficiency: Given its enormous number of parameters, deep reinforcement learning naturally require substantial amount of simulation and training data to converge. By leveraging domain knowledge, one could preclude decisions that are suboptimal based on domain knowledge, thus substantially reducing the search space of exploration and increasing the training efficiency.

5.2 Regulating DNNs with domain knowledge

One basic strategy to integrate domain-specific knowledge in DNNs is to use the output of a domain-specific policy to *regulate* DNN training process, so that the resulting DNN will jointly optimize the loss function while imitating as much as possible the behavior of a domain-specific policy. Here, we briefly describe a state-of-the-art of DNN regularization technique [6].

The basic idea is to use two processes, called "teacher" and "student", to regulate the training of a DNN f with a domain-specific logic g [6]. Both f and g take input $\mathbf{x}$ and output a probability distribution $\mathbf{y}$ over the action space; if the domain-specific logic is deterministic, $g(\mathbf{x})$ will be a one-hot distribution. At a high level, "teacher" uses logic rules to refine the behavior of g and obtain a different neural network l, while "student" trains f such that it minimizes the difference to the ground truth as well as the difference to the output of g. More details are omitted due to limited space.

To see this approach in action, we take resource allocation as an example, and integrate the shortest-job-first style logic as the domain-specific policy in the training of DeepRM. The intuition is that DeepRM might aggressively withhold jobs with the assumption that smaller jobs always arrive in the same frequency as the training data, whereas shortest-job-first logic makes minimal assumption on the workload and so is more generalizable, albeit less performant. By penalizing DeepRM when it deviates from shortest-job-first logic (e.g., withhold a job when there is resources to run it) with only marginal performance gain, this process may make DeepRM more reliable while consistently performing better than shortest-job-first logic on different workloads. The key challenges are how the experience of resource scheduling can be extracted as logic rules and how they are mapped into a *good* teacher. Despite that the domain-specific knowledge is versatile among different networking applications, the rationale of integrating human knowledge into neural network is generic.

5.3 Research thrusts

Inspired by the initial successes, we hope to bring more domain-specific knowledge to harden the DNNs in networking/systems.

> *How to systematically integrate domain knowledge of networking/systems in DNNs without sacrificing their superior performance?*

Other areas, e.g., natural language processing, have paid tremendous efforts and showed great benefits in doing that [4]. Networking and systems research does have so many invaluable lessons and principles that this community should strive to distill and apply in the new DNN-based approaches.

6 CONCLUSION

Deep neural networks are here to stay. Instead of treating DNNs as black boxes, this paper argues that we should develop techniques and leverage domain-specific insights to strive to interpret the DNNs trained in networking settings. This is not only driven by the recent drive towards interpretable ML, but more importantly, motivated by the need in many networking settings for solutions to be explainable and robust. Our preliminary results barely scratch the surface of this daunting task, but we hope to inspire more efforts in the community.

REFERENCES

[1] Interpretable ML Symposium (NIPS 2017). http://interpretable.ml/.

[2] D. Balduzzi, M. Frean, L. Leary, J. Lewis, K. W.-D. Ma, and B. McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? In *International Conference on Machine Learning*, pages 342–350, 2017.

[3] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

[4] M. V. França, G. Zaverucha, and A. S. d. Garcez. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine learning*, 94(1):81–104, 2014.

[5] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI*, volume 11, pages 24–24, 2011.

[6] Z. Hu, X. Ma, Z. Liu, E. Hovy, and E. Xing. Harnessing deep neural networks with logic rules. In *Proc. of ACL 2016*, volume 1, pages 2410–2420, 2016.

[7] J. Jiang, V. Sekar, H. Milner, D. Shepherd, I. Stoica, and H. Zhang. Cfa: A practical prediction system for video qoe optimization. In *NSDI*, pages 137–150, 2016.

[8] Z. Lipton. The mythos of model interpretability. icml 2016 workshop on human interpretability in machine learning (whi 2016), 2016.

[9] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[10] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. Resource management with deep reinforcement learning. In *Proc. of ACM Workshop on HotNets 2016*, pages 50–56. ACM, 2016.

[11] H. Mao, R. Netravali, and M. Alizadeh. Neural adaptive video streaming with pensieve. In *Proc. of ACM SIGCOMM 2017*, pages 197–210. ACM, 2017.

[12] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597. IEEE, 2016.

[13] K. Pei, Y. Cao, J. Yang, and S. Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18. ACM, 2017.

[14] R. S. Ramprasaath, C. Michael, D. Abhishek, V. Ramakrishna, P. Devi, and B. Dhruv. Grad-cam:visual explanations from deep networks via gradient-based localization. In *International Conference on Computer Vision*. IEEE, 2016.

[15] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[16] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[17] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar. Learning to route. In *Proc. of ACM HotNets 2017*, pages 185–191. ACM, 2017.

[18] Z. Xu, Y. Wang, J. Tang, J. Wang, and M. C. Gursoy. A deep reinforcement learning based framework for power-efficient resource allocation in cloud rans. In *Proc. of IEEE ICC 2017*, pages 1–6. IEEE, 2017.

[19] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer vVsion*, pages 818–833. Springer, 2014.