# Experimental Assessment of Reversibility-Aware Deep Reinforcement Learning for Optical Data Center Network Reconfiguration

Massimiliano Sica[1,3], Sandeep Kumar Singh[1], Roberto Proietti[2], Massimo Tornatore[3], S. J. Ben Yoo[1,*]

[1]*Department of Electrical and Computer Engineering, University of California, Davis, CA, USA, 95616*
[2]*Department of Electronics and Telecommunication, Politecnico di Torino, 10129, Torino, Italy*
[3]*Department of Electronics, Information and Bioengineering, Politecnico di Milano, 20133, Milano, Italy*
*\*Corresponding author: {sbyoo}@ucdavis.edu*

*Abstract*—The performance of communication-intensive distributed machine learning (DML) workloads and other emerging applications can suffer from a traffic-topology mismatch in traditional data-center networks. This degradation can be alleviated by performing a logical network topology reconfiguration. However, how to dynamically reconfigure the logical topology and steer the bandwidth efficiently with a control plane capable of efficiently adapting to the current data center traffic patterns without considerable overhead is still an open question. This paper presents a reversibility-aware deep reinforcement learning algorithm (RA-DRL) for optical switch reconfiguration in data center networks and validates it in an experimental testbed. Using our testbed, we show that appropriate optical-switch reconfiguration, driven both by a baseline DRL and an RA-DRL method, can improve the training performance of DML workloads under network congestion. More importantly, by incorporating the concept of reversibility in the training of the DRL agent, we demonstrate a 5x training-time decrease for a distributed computer-vision application and an improvement in convergence time by up to 64%.

## I. INTRODUCTION

The enormous growth of cloud computing has led to an increase in the number of network services with different requirements and, consequently, to more complex traffic patterns [1]. Thus, today's data center (DC) and high-performance computing (HPC) networks are characterized by increasingly spiky, diverse, and unpredictable traffic. Current DC/HPC networks have dealt with this problem using static over-provisioned architectures designed to handle worst-case scenarios. Unfortunately, this kind of static networks architectures does not efficiently adapt to unpredictable traffic and typically results in quality-of-service (QoS) degradation. In addition, worst-case static provisioning requires very expensive and unnecessary cabling, excessive heat production, and high power consumption [2], [3].

Reconfigurable optical DC networks are being considered as an alternative to electrical DC networks [4]. But performing optical-circuit reconfiguration on live traffic can momentarily degrade some of the applications' QoS, like the completion time of a computing application's workload. Additionally, a reconfigured topology might not remain optimal to serve dynamic applications for a long time, and frequent recon-

figuration might be required. Thus, reconfiguration strategies using integer linear programming (ILP) formulations [5], [6] are inadequate for large-scale DC/HPC networks. Similarly, heuristic-based reconfiguration strategies are often inadequate due to the lack of generalization and the tendency to converge to sub-optimal solutions [7], [8].

In this context, researchers started exploring machine learning-based solutions that have the potential to scale properly without considerable human intervention [9]–[11]. Although the current deep reinforcement learning (DRL) based solutions are promising [12]–[14] thanks to their fast decision capabilities and high adaptability, they have some disadvantages. Specifically, DRL follows a trial-and-error approach which leads to making many mistakes (taking sub-optimal or failure-leading actions) that can slow down the training process. DRL training can be lengthy, energy-consuming, and data-hungry.

In this work, we move beyond statically wired networks and we develop a smart infrastructure capable of dynamically provisioning the resources needed for each application, leading to an improvement in QoS and a simpler network architecture. Our technical contributions to this study can be summarized as follows. First, we deploy two cloud applications (DML and Iperf) in our experimental intra-DC testbed, and we evaluate the improvement in DML training performance by reconfiguring the intra-DC network topology. To solve the problem of dynamically reconfiguring network resources, we use a DQN agent [15]. The DQN-based DRL method is used as a baseline. Second, we adopt a reversibility-aware (RA) DRL method [16] to show that RA-DRL can improve the convergence time of the DQN agent by reducing the number of actions leading to the network collapse state or sub-optimal solutions during the training of the DQN agent. The proposed RA-DRL agent achieves a 5x improvement in training time for DML with respect to the baseline DQN. Moreover, the RA-DRL algorithm can improve the agent's performance, leading to up to 64% faster convergence.

The rest of the paper is organized as follows. Sec. II describes our experimental testbed and introduces the problem definition and the DQN baseline solution. We discuss the more

advanced RA-DRL method for reconfiguration in Sec. III. The experimental setup and results are presented in Sec. IV. Sec. V concludes the paper.

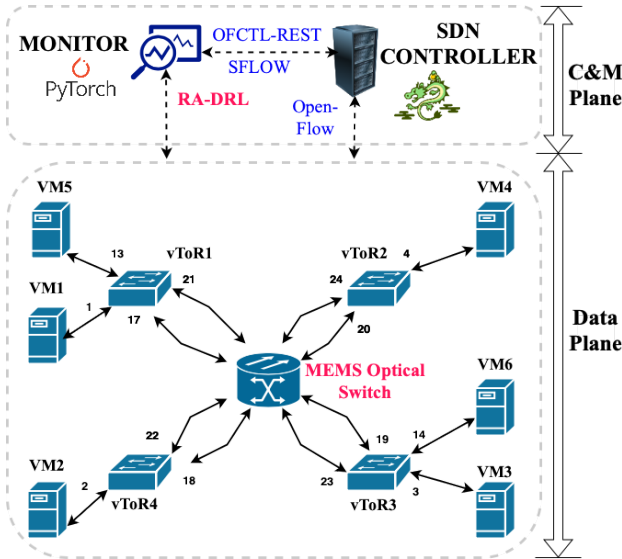## II. DRL-Enabled Optical Switch Reconfiguration



Fig. 1. System's architecture consisting of a software-controlled and managed plane and a data plane with hosts and network devices.

### A. System Architecture

Fig. 1 shows the architecture of our experimental testbed, which represents an example of a small-scale DC/HPC system. It consists of a data plane and a control and management (C&M) plane. The data plane is composed of six virtual machines (VM1 to VM6), four virtual top-of-the-rack (vToR) switches (vToR1 to vToR4), and an optical circuit switch (OCS). The OCS is connected with two bidirectional fibers to each vToR, and by means of reconfiguration it allows to provision different topologies. The C&M plane hosts a monitor and a software-defined network (SDN) controller. The Monitor server is where the brain of the application is, and it sends commands to the SDN controller to install new flows on vToRs, reconfigure the OCS or gather traffic metrics from the data plane. The C&M plane also runs our proposed reversibility-aware DRL algorithm module for OCS reconfiguration to help decongest the network and improve the training time of a DML application running over multiple virtual machines.

### B. Problem Definition and Solution

The goal of the work is to reduce the congestion in the network by reconfiguring the OCS using reinforcement learning based approaches in presence of multiple applications. We demonstrate a DC control system that takes the real-time traffic matrix and network topology as inputs and outputs an OCS reconfiguration. Then, we update the flow tables and OCS configuration in the data plane for routing the traffic. To solve the routing optimization problem, we use a DRL
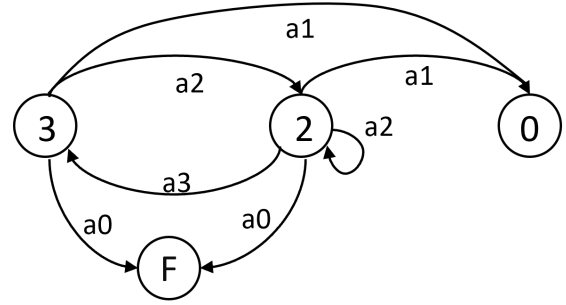


Fig. 2. An example of an MDP for the experimental scenario considered.

algorithm together with traffic monitoring. In general, DRL tries to learn an optimal reconfiguration policy $\pi$ by taking a set of actions $A$ on possible states $S$ to maximize the reward $R$. A DRL agent acts on a dynamic environment, described by a Markov decision process (MDP) with a tuple $< s, a, r, s' >$. The agent collects a reward $r$ by taking action $a$ in state $s$ to reach the next state $s'$. Note that there are some states after which the environment no longer changes, e.g., win or loss state in a game. All the different types of interactions of the agent with the environment (taking actions, collecting rewards, moving to the next state) are defined within the context of an episode. An episode is terminated whenever a failure or success state is reached or after reaching a maximum number of actions. This number (e.g., 10 in our case) is generally chosen via trial and error to balance the length of the episode and the learning speed of the agent.

We model the multi-application traffic optimization problem with an MDP representing the link congestion status between the vToRs and the OCS. To illustrate the problem we show an example in Fig. 2 where the MDP has four states $S = \{0, 2, 3, F\}$. Here, a state $s \in S$ represents the number of congested links (vToR to OCS) in the network. Specifically, the state $F$ refers to the "failure state" in which no traffic flows due to a communication breakdown by a faulty OCS reconfiguration scheme. Each action in $A = \{a1, a2, a3, a4\}$ over state space $S$ leads to a transition between two states. This transition corresponds to a reconfiguration pattern expressed through an OCS matrix, like the one shown in Fig. 3, where the entry value one means that the two ports of OCS are connected. Each action will lead to a different network topology which may lead to a decrease in network congestion, and the goal is to converge to state 0 where the optimal congestion scenario is reached.

To learn an optimal routing policy in the MDP, we formulate a DRL reward function where positive rewards are collected when actions lead to less congested states. We formally define the reward function in Eq. (1) below, which we designed to minimize the number of congested links. The variable $\sigma_t$ represents the number of congested links used by the DML algorithm at a certain time step, to optimize its routing policy. Variable $L$ represents the total number of links. The cost function $A(t + 1)$ is given by Eq. (1).

$$A(t+1) = \frac{\sigma_t}{L} - \frac{\sigma_{t+1}}{L} \qquad (1)$$

By looking at Eq. (1) it becomes clear that the reward becomes larger if the number of congested links decreases. In this way, by maximizing the reward, we minimize the number of congested links where the DML is running, which is given by $A(t) - A(t+1)$, which represents our reward function. For example, when the number of links is $L = 24$ (link in our setup) and an action $a1$ is taken in state 3, we collect a positive reward of 0.125 ($\frac{3}{24} - \frac{0}{24}$). On the other hand, if action $a0$ is taken in state 3 or 2, it will lead to a failed state $F$ with a negative reward of -0.125, which has been chosen via a trial and error procedure in order to smooth the convergence of the agent.

The next subsection describes our proposed methodology for real-time OCS reconfiguration involving multiple steps.

|    | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|----|----|----|----|----|----|----|----|----|
| 17 | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  |
| 18 | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  |
| 19 | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  |
| 20 | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 21 | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 22 | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  |
| 23 | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 24 | 0  | 0  | 0  | 0  | 0  | 1  | 0  | 0  |

Fig. 3. An OCS port connectivity example with 1 representing corresponding ports are connected, and 0 as disconnected.
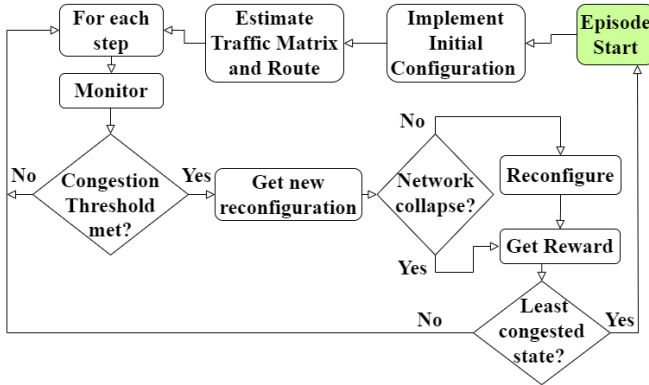


Fig. 4. A flow chart for the (RA-)DRL-based OCS reconfiguration methodology adopted in this paper.

### C. Workflow Description

A workflow for our proposed solution is shown in Fig. 4.

The episode starts by implementing an initial OCS configuration to guarantee connectivity among all the VMs to start the DML. After the initialization, for every step up to ten in an episode, we monitor the traffic, check the current network state, and verify if the reconfiguration threshold is met. In fact, the agent should only reconfigure the network if it is needed (the reconfiguration threshold is exceeded), otherwise we would only incur unnecessary overhead. If the threshold is not met, we go back to the monitoring phase. Otherwise, we trigger the DRL (or RA-DRL) to get a new OCS reconfiguration (Fig. 3). Real-time network analytics are collected using Sflow, a tool that allows us to gather network statistics using a Rest API. The metrics are stored per flow, where each flow is defined IP-to-IP. The OCS is managed using Standard Commands for Programmable Instruments (SCPI) via a TCP socket. We build the DRL agent using a deep Q-learning approach which is naturally suited for this problem given the discrete action space. In fact, the action space is made up of a set of different reconfiguration topologies that are chosen by the agent depending on the output of the neural network. Before actually implementing the reconfiguration, we check whether or not it will lead to a collapsed (or failure) state $F$. If it does not, then we implement the action and get the associated reward. Otherwise, we just get a negative reward of -0.125 and terminate the episode. Once the new OCS configuration has been implemented, we install new flow tables in the vToRs to perform routing over the new obtained topology. The routing works by computing the $k-$shortest paths between servers and selects the one with the most cumulative bandwidth available. Once the paths have been chosen, by means of the Ryu *ofctl-rest* API an OpenFlow message is sent to the ToR switches to install the needed flows. After the flows have been installed, we collect the reward and store it in the replay buffer for training the agent. The last step is to check whether we have reached the optimal state 0 or the failure $F$ state. If we reached either of the two states, then we move on to the next episode by terminating the current applications and restarting them again; otherwise, we fall back to traffic monitoring again while the applications keep running.

## III. RA-DRL FOR OPTICAL SWITCH RECONFIGURATION

One of the main limitations of reinforcement learning, in general, is the lack of awareness when taking some actions that could lead to an "irreversible" outcome, such as a failure state or network breakdown.

Recently, the authors of [16] proposed to embed a "reversibility" estimation in reinforcement learning agents. Reversibility estimation allows teaching the agent to maintain a safer behavior in high-risk environments, for example, a robot handling fragile material. The RA-DRL algorithm subtracts the reversibility factor $\rho_t$ from the reward of standard DRL given in Eq. (1). $\rho_t$ represents the probability that a state comes before another state on average. The reward in the RA-DRL is given by Eq. (2)).

$$A(t+1) = \frac{\sigma_t}{L} - \frac{\sigma_{t+1}}{L} - \rho_t \qquad (2)$$

The $\rho$ value in Eq. (2) represents the probability of reversibility of each action, and its goal is to penalize irreversible

actions. For example, going to the $F$ and getting stuck on state 2 should be strongly penalized. One may argue that terminating the episode when state $F$ or 0 is reached will teach the agent to avoid the optimal action leading to an optimal state 0. To avoid this issue, the penalty $\rho$ is not applied to the action leading to the optimal state 0. The next question is how to compute $\rho_t$, which we describe below.

### A. Reversibility Estimation

The reversibility factor $\rho$ is estimated using a so-called self-supervised learning process, which uses a neural network with auto-labeling of samples. The neural network has an input layer, a hidden layer with ReLu activation, and an output layer with a single output with sigmoid activation for estimating the probability of reversibility. The structure of the neural network is similar to the ones found in classical supervised learning. However, the training phase is considerably different.

In detail, the first step is to randomly sample a pair of consecutive states, say (s1, s2), from the replay buffer – an ordered queue – that stores the MDP state evolution. After sampling, we feed them to the neural network in random order and provide the actual ordering of the states as a target value. Note that the true order of states is never explicitly provided to the neural network (like in classical supervised learning) but inferred directly from the ordered data. Then, both states are passed independently through the first hidden layer before concatenating them together. The concatenation is passed through the output layer, which will output the probability that s1 comes before s2 on average. In order to compute the loss of the neural network, we use the non-shuffled state pair as targets or the true order. The self-labeling of the data through the use of temporal order information for learning the reversibility makes the RA-DRL method attractive.
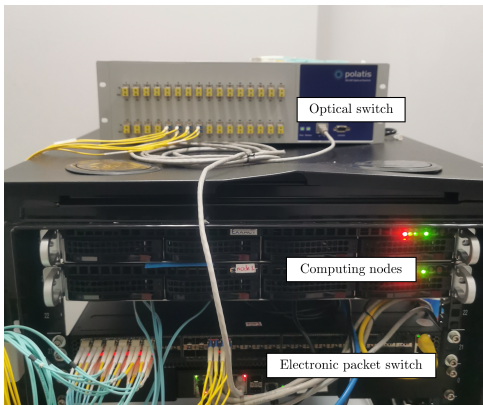


Fig. 5. An experimental testbed with an optical switch, computing nodes, and an electronic packet switch (or ToR) virtualized into 4 vTors.

## IV. EXPERIMENTAL SETUP AND RESULTS

### A. Experimental Setup

To validate the reversibility-aware DRL method for optical switch reconfiguration, we set up some experiments on the testbed shown in Fig. 5 with 6 VMs, 4 vToRs, and a MEMS

optical switch (see the detailed connectivity in Fig. 1). We consider two applications: DML and Iperf. The DML is deployed in four virtual machines (VM1 to VM 4). The DML consists of an image classification task of the Cifar10 dataset, which contains 60,000 $32 \times 32$ color images in 10 classes, with 6000 images per class. The Iperf application runs on VM5 and VM6. It generates 9 Gbit/s User Datagram Protocol (UDP) traffic from VM5 to VM6 and acts as background traffic. The Iperf is started once per episode in order to congest the links where the DML traffic flows and trigger the execution of the DRL/ RA-DRL algorithms.

We train the DRL and RA-DRL agents to learn a policy to optimize the routing to achieve the least congestion state 0 as formulated by the MDP in Fig. 2. Note that state 1 is not present because we need at least two links between vToR to vToR via OCS to enable the communication between two VMs. A state $s$ is represented by a one-hot encoding with a vector size of 4. The one-hot encoding is the input for our DRL agent, whose structure is as follows. It has an input layer of size 4 neurons with normalization, two hidden layers of sizes 15 and 30 neurons with the ReLU activation, and an output layer of 4 neurons with the ReLU activation. The different layer sizes have been considered as hyper-parameters and, as such, have been chosen based on a trial-and-error approach.

We train the agents offline for 2,000 episodes following an $\epsilon$-greedy exploration with discount factor $\gamma = 0.9$ and a replay buffer of size 250. $\gamma$ is a parameter that can be tweaked in order to make the agent more or less focused on short-term gains, with a $\gamma = 1$ the agent will evaluate each of the available actions based on the total of all the future rewards. $\epsilon$-greedy exploration is used to deal with the exploration-exploitation dilemma and consists in taking a random action with a probability $\epsilon$. An episode is terminated after either 10 reconfiguration steps have been implemented without success or the optimal or failure state has been reached. The same experimental setup has been used for the DRL and the RA-DRL methods.
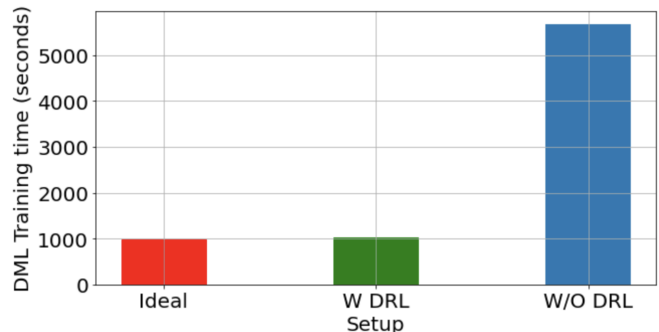


Fig. 6. DML training time obtained under three scenarios over 20 epochs.

### B. Results

Fig. 6 shows the DML's training time for three scenarios: i) Ideal shown in red when DML as a single application runs on the VMs, ii) with DRL-based reconfiguration, shown in

green, under two applications (DML and iPerf) running, and iii) without DRL (or RA-DRL), shown in blue, under the two applications. We observe the following DML training times for the three scenarios.

- Ideal: 977 seconds ($\approx$ 16 minutes)
- With (W/) DRL: 1026 seconds ($\approx$ 17 minutes)
- Without (W/O) DRL: 5676 seconds ($\approx$ 94 minutes)

The above result is interesting as it provides a proof of concept that DML workloads can have their training time improved thanks to optical reconfiguration, and that DRL can be used to successfully drive the whole process. The rewards evolution and the loss for the DRL agent are shown in Fig. 7 and Fig. 8, respectively. The agent's loss shows the expected behavior by slowly converging to approximately zero around training epoch 400. Also, the reward evolution behaves as expected by converging to the maximum reward of 0.125. The spike around episode 500 in 7 is likely due to the exploration process, which consists in taking a random action with a probability $\epsilon = 0.8$, which decreases in time but does not become zero.
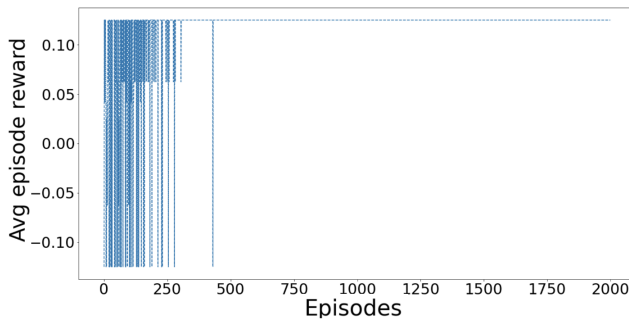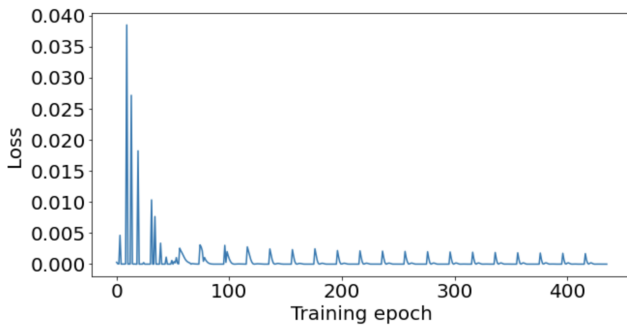


Fig. 7. DQN Reward evolution



Fig. 8. DQN Loss evolution

Next, we investigate the performance of the RA-DRL method. In our experiment, the trained RA-DRL agent converges to the optimal state 0 (see Fig. 2) and thus shows the same performance gain as with DRL in Fig 6 when tested under two applications. The reason for the above is related to the fact that both agents, after offline training, converge to the optimal solution. However, the main benefit of RA-DRL comes during the training phase. The idea is to embed the

concept of causality in the behavior of the DRL agent in order to speed up the training process.

The comparison between the regular DQN agent for the DRL and the self-supervised aided one for RA-DRL is shown in Fig. 9 with the exploration $\epsilon$ of 0.8. From the plot, we can see that the RA-DRL is able to converge faster than the regular DRL.
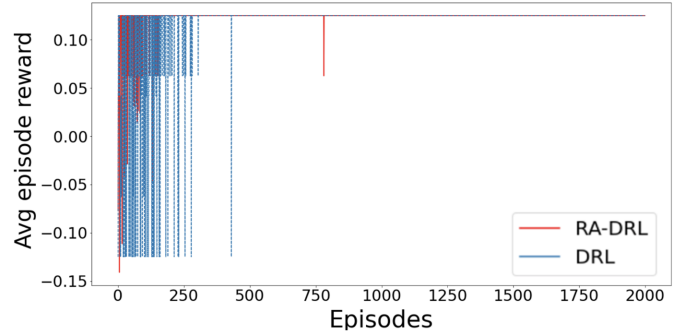


Fig. 9. Reward evolution DRL vs RA-DRL with exploration $\epsilon = 0.8$

However, having such a high exploration value does not allow for a clear comparison between the DRL and RA-DRL, since the oscillations due to exploration and the effects on convergence can be relevant. The benefits of the RA-DRL on convergence may not always be very evident in every experiment. Therefore, we opt for an exploration value $\epsilon = 0.1$, and the corresponding average episode reward is shown in Fig. 10.
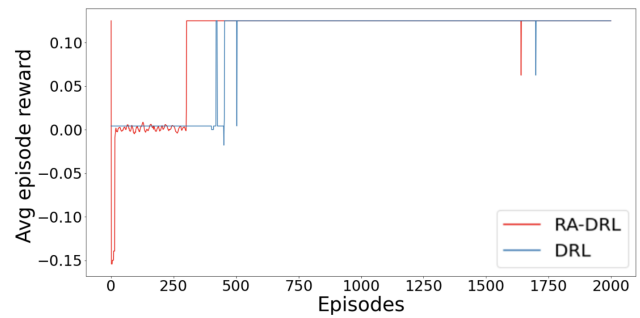


Fig. 10. Reward evolution DQN (blue) vs reward evolution DQN with a self-supervised module (red) $\epsilon = 0.1$.

In order to precisely quantify the advantage of using the RA-DRL method, we perform five experiments made of ten runs of DRL and ten runs of RA-DRL with $\epsilon = 0.1$. This allows us to understand how much gain in convergence time could be obtained. The results are shown in Fig. 11. Interestingly, the classical DRL does not converge to the optimal solution every time, while the RA-DRL does. With $\epsilon = 0.1$, the DRL agent can get stuck permanently in state 2 (Fig. 2). On the other hand, the RA-DRL will recognize state 2 as an irreversible state and avoid it, leading to convergence in every experiment. Converging to state 2 would be a suboptimal solution. It can be observed that the RA-DRL always outperforms the DRL

algorithm by a minimum factor of 10% to a maximum of about 64%, which proves our initial hypothesis.
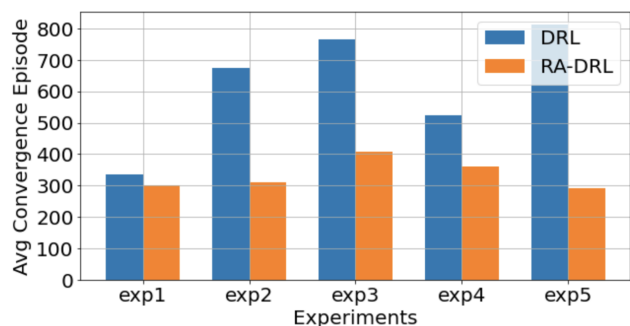


Fig. 11.  Average convergence episode for five experiments.

Finally, the loss of the self-supervised aided DQN network for the RA-DRL with respect to the number of training epochs is shown in Fig. 12. It is worth pointing out that the self-supervised loss is not showing the ideal behavior (smooth convergence to a smaller value); despite decreasing and converging around the value of 0.2, the convergence is not very stable. This behavior requires a future investigation.
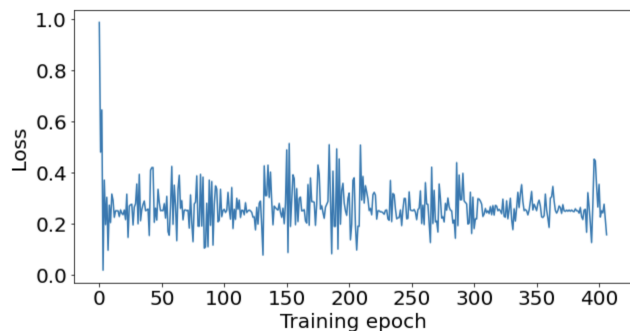


Fig. 12.  Self-supervised loss with respect to training epochs.

## V. CONCLUSIONS

This paper presented a deep reinforcement learning (DRL)-based optical reconfiguration algorithm to improve the training time of a distributed machine learning (DML) application running over 4 nodes in an experimental testbed in the presence of network congestion. We demonstrated a $5\times$ times training time improvement by generating a new topology via optical switch reconfiguration and routing to completely separate the DML from the congested traffic flow. We were also able to improve the performance of the DRL agent using a reversibility-aware technique leading to faster convergence up to 64%. This work shows that DML workloads can improve their training times thanks to optical reconfiguration and that a reversibility-aware method can improve the performance of a regular DQN agent and speed up the training process considerably. In future work, we will evaluate the scalability of the proposed method through network simulation.

## REFERENCES

[1] M. Khani, M. Ghobadi, M. Alizadeh, Z. Zhu, M. Glick, K. Bergman, A. Vahdat, B. Klenk, and E. Ebrahimi, "Sip-ml: high-bandwidth optical network interconnects for machine learning training," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 657–675.

[2] F. A. Moghaddam, P. Lago, and P. Grosso, "Energy-efficient networking solutions in cloud-based environments: A systematic literature review," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, pp. 1–32, 2015.

[3] A. Venkatraman, "Global census shows datacentre power demand grew 63% in 2012," *ComputerWeekly. com*, vol. 8, 2012.

[4] L. Poutievski *et al.*, "Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 66–85.

[5] 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," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 319–330.

[6] H. Liu, M. K. Mukerjee, C. Li, N. Feltman, G. Papen, S. Savage, S. Seshan, G. M. Voelker, D. G. Andersen, M. Kaminsky *et al.*, "Scheduling techniques for hybrid circuit/packet networks," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, 2015, pp. 1–13.

[7] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: A hybrid electrical/optical switch architecture for modular data centers," in *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010, pp. 339–350.

[8] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. E. Ng, M. Kozuch, and M. Ryan, "c-through: Part-time optics in data centers," in *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010, pp. 327–338.

[9] D. Rafique and L. Velasco, "Machine learning for network automation: Overview, architecture, and applications [invited tutorial]," *Journal of Optical Communications and Networking*, vol. 10, no. 10, pp. D126–D143, 2018.

[10] M. Wang, Y. Cui, S. Xiao, X. Wang, D. Yang, K. Chen, and J. Zhu, "Neural network meets dcn: Traffic-driven topology adaptation with deep learning," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 2, pp. 1–25, 2018.

[11] S. K. Singh, C.-Y. Liu, S. B. Yoo, and R. Proietti, "Machine-learning-aided dynamic reconfiguration in optical dc/hpc networks," in *2022 International Conference on Optical Network Design and Modeling (ONDM)*. IEEE, 2022, pp. 1–6.

[12] X. Guo, F. Yan, X. Xue, B. Pan, G. Exarchakos, and N. Calabretta, "Qos-aware data center network reconfiguration method based on deep reinforcement learning," *Journal of Optical Communications and Networking*, vol. 13, no. 5, pp. 94–107, 2021.

[13] P. Almasan, J. Suárez-Varela, K. Rusek, P. Barlet-Ros, and A. Cabellos-Aparicio, "Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case," *Computer Communications*, vol. 196, pp. 184–194, 2022.

[14] R. Proietti, X. Chen, Y. Shang, and S. B. Yoo, "Self-driving reconfiguration of data center networks by deep reinforcement learning and silicon photonic flex-lion switches," in *2020 IEEE Photonics Conference (IPC)*. IEEE, 2020, pp. 1–2.

[15] 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*, vol. 518, no. 7540, pp. 529–533, 2015.

[16] N. Grinsztajn, J. Ferret, O. Pietquin, M. Geist *et al.*, "There is no turning back: A self-supervised approach for reversibility-aware reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 1898–1911, 2021.