IZI: Intelligent Zero-Trust Network for IoT

Report 3: Deep learning algorithm development (part 2)

Mikhail Zolotukhin, Pyry Kotilainen and Timo Hämäläinen
Faculty of Information Technology, University of Jyväskylä, Finland.*

Abstract

Artificial intelligence and deep learning are revolutionizing cyber-security research. Modern machine learning algorithms allow for an accurate network traffic analysis and intrusion detection and classification. In this project, we employ several deep learning algorithms for three tasks developing a network defense framework that would be able to detect attacks in high-speed network environments, adapt detection models under constantly changing network context caused by adding new applications and services. The key component of the framework is supposed to rely on multiple reinforcement learning agents that evaluate the state of the network environment under consideration and make the most optimal real-time crisis-action decision on how the network security policy should be modified in order to minimize the risk of subsequent attacks in the future. These decisions are partially based on the information provided by deep classification and anomaly detection models that are deployed on virtual security middle boxes. In this report, we provide more evaluation results using publicly available network traffic datasets and simulation environments.

1 Introduction

As mentioned in the second report, a typical deep neural network consists of multiple layers of nonlinear processing units. The main idea behind deep learning is using the first layers to find compact low-dimensional representations of high-dimensional data whereas later layers are responsible for achievement of the task given, e.g. regression or categorical classification. Spatial dependencies in the data can be extracted with convolutional layers [1] and temporal dependencies - with the help of recurrent neural networks [2] or attention mechanism [3].

Speaking of deep RL, a neural network can be used to estimate the agent's policy with loss function being estimated based on the probability of the action taken multiplied by the cumulative reward obtained from the environment. Updating the policy network parameters by taking random samples may introduce high variability in probabilities and cumulative reward values, because trajectories during training can deviate from each other at great degrees. This results in unstable learning and the policy distribution skewing to a non-optimal direction. One way to reduce variance and increase stability is subtracting the value function from the cumulative reward. This allows one to estimate how much better the action taken is compared to the return of an average action. The value function can be estimated by constructing the second neural network, which estimates the environment's state value in the manner similar to DQN. The resulting architecture is called advantageous actor-critic (A2C), where the critic estimates the value function, while the actor updates the policy distribution in the direction suggested by the critic [4].

To improve stability of the learning even further, trust region policy optimization (TRPO) relies on minimizing a certain surrogate objective function that guarantees policy improvement with non-trivial step sizes [5]. TRPO uses average KL divergence between the old policy and updated policy as a measurement for a region around the current policy parameters within which they trust the model to be an adequate representation of the objective function, and then chooses the step to be the approximate minimizer of the model in this region. Although TRPO has achieved great and consistent high performance, the computation and implementation of it is extremely complicated. The current state-of-art algorithm policy optimization (PPO) attempts to reduce the complexity of TRPO implementation and computation by tracing the impact of the actions with a ratio between the probability of action under current policy divided by the probability of the action under previous policy and artificially clipping this value in order to avoid having too large policy update

*This project has received funding from the European Union's Horizon 2020 research and innovation programme under the NGI_TRUST grant agreement no 825618

[6]. Another option to reduce the complexity closer to a first-order optimization is proposed in [7]. Actor-critic using Kronecker-Factored trust region (ACKTR) speeds up the optimization by reducing the complexity using the Kronecker-factored approximation.

Finally, in order to train the reinforcement learning agents a simulation environment is supposed to be constructed since we cannot deploy, train and test those agents in a real production network. Traffic in such environment can be attempted to be generated with the help of conditional generative adversarial networks (GANs). GAN can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information [8]. We can perform the conditioning by feeding this information into the both the discriminator and generator as additional input layer. In our case, this extra information includes several packets sent in a network traffic flow, and the GAN is trained to generate features of the next packet, i.e. its payload size, TCP window size, TCP flags, inter-arrival time, etc.

The purpose of this document is to test various deep learning algorithms we are planning to use for implementation of the defense framework proposed in the project. The rest of the document is organized as follows. In Section 2, we implement and test several deep learning solutions for the problem of online intrusion detection. Reinforcement learning and generative adversarial methods are outlined in Sections 3 and 4 correspondingly. Section 5 concludes the paper and outlines future work.

2 Classification

As mentioned int he previous report, to evaluate deep learning model capabilities to detect intrusions we use network packet captures from CICIDS2018 [9]. It contains 560 Gb of traffic generated during 10 days by 470 machines. The dataset in addition to benign samples includes following attacks: infiltration of the network from inside, HTTP denial of service, web, SSH and FTP brute force attacks, attacks based on known vulnerabilities. We concentrate on the intrusion detection based on the analysis of network traffic flows. For each such conversation, at each time window, or when a new packet arrives, we extract the following information: flow duration, total number of packets in forward and backward direction, total size of the packets in forward direction, minimum, mean, maximum, and standard deviation of packet size in forward and backward direction and overall in the flow, number of packets and bytes per second, minimum, mean, maximum and standard deviation of packet inter-arrival time in forward and backward direction and overall in the flow, total number of bytes in packet headers in forward and backward direction, number of packets per second in forward and backward direction, number of packets with different TCP flags, backward-to-forward number of bytes ratio, average number of packets and bytes transferred in bulk in the forward and backward direction, the average number of packets in a sub flow in the forward and backward direction, number of bytes sent in initial window in the forward and backward direction, minimum, mean, maximum and standard deviation of time the flow is active, minimum, mean, maximum and standard deviation of time the flow is idle. All the features can have different scale and therefore they are transformed into range between zero and one with the help of min-max standardization.

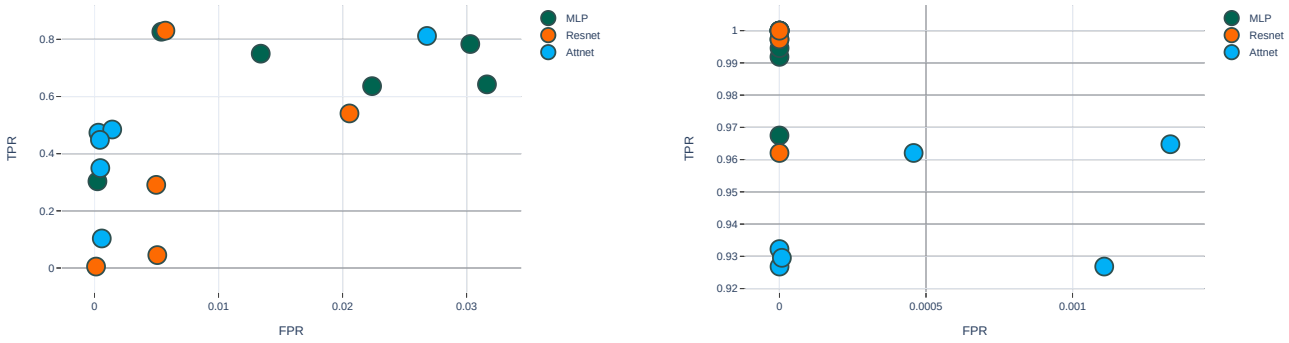
In this report, we present corrected results for the supervised intrusion detection. The problem with the previous results was using validation loss as the metric for early stopping. In case of imbalanced data, however, the proper way is to concentrate on increasing area under ROC curve (AUC). We substituted the validation metric for the early stopping in our experiments and this allowed us to improve the results slightly. The second improvement involves initializing the bias of the last layer of the network based on the numbers of normal and malicious flows in the dataset. Results for HTTP and other traffic are presented on the Figure 1.

3 Reinforcement learning

This section presents performance results for three state-of-art RL algorithms A2C, ACKTR and PPO evaluated in several virtualized environments. We concentrated on these algorithms as they can be applied for both discrete and continuous environments. The most important hyper-parameters for the algorithms are listed below:

- A2C: number of environments: 16, number of time steps: 8000000, policy: MLP, entropy coefficient: 0.001, gamma: 0.95, learning rate: 7e-4
- ACKTR: number of environments: 16, number of time steps: 8000000, policy: MLP, entropy coefficient: 0.001 gamma: 0.99, learning rate: 0.06
- PPO: number of environments: 16, number of time steps: 8000000, policy: MLP, entropy coefficient: 0.001 gamma: 0.99, lambda: 0.95, learning rate: 2.5e-4, number of mini-batches: 16, number of epochs per update: 8

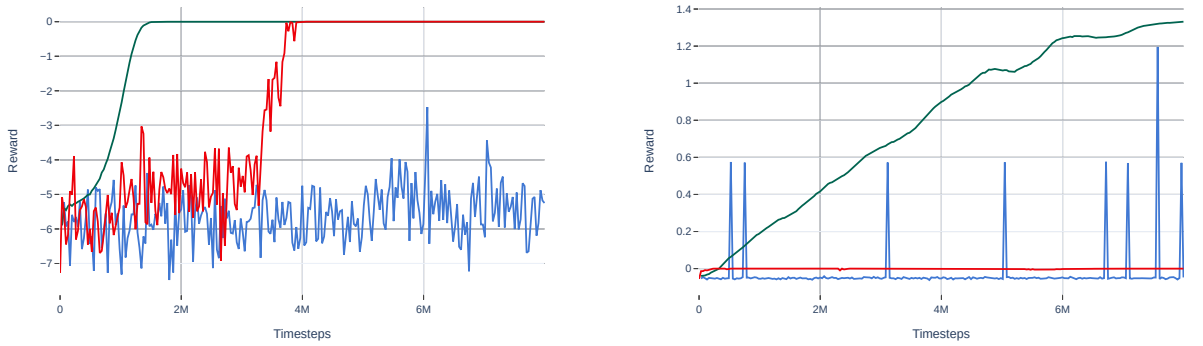
As previously, PPO consistently provides good results in terms of both average reward and convergence speed. There is only one environment ACKTR outperforms PPO. We further will use PPO as our baseline algorithm while using ACKTR in case PPO struggles to find any good policy.



(a) HTTP traffic

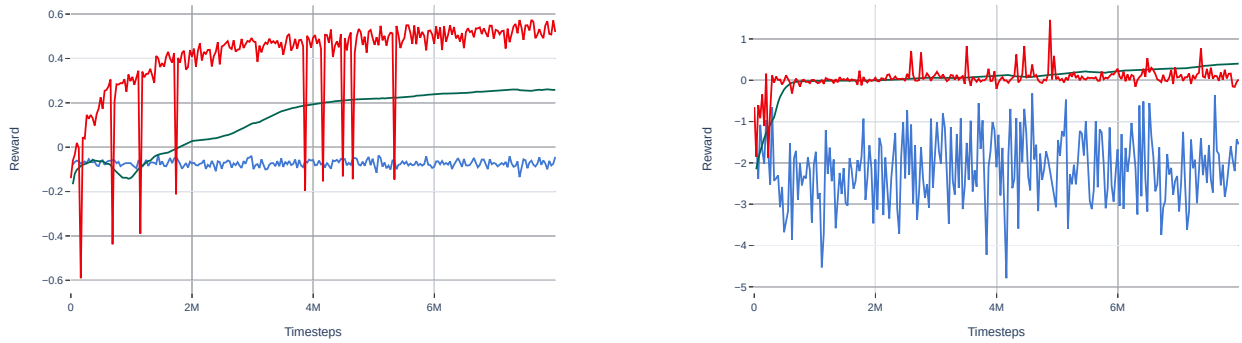
(b) Other TCP traffic

Figure 1: Performance evaluation of malicious traffic detection supervised deep learning algorithms. For HTTP traffic, attacks include protocol and application DoS attacks, cross-site scripting, password bruteforcing, SQL injections, and communication between infected devices and C&C. Other TCP traffic includes flows initiated during the infiltration attack.



(a) Pendulum.

(b) Continuous mountain car.



(c) Bipedal walker.

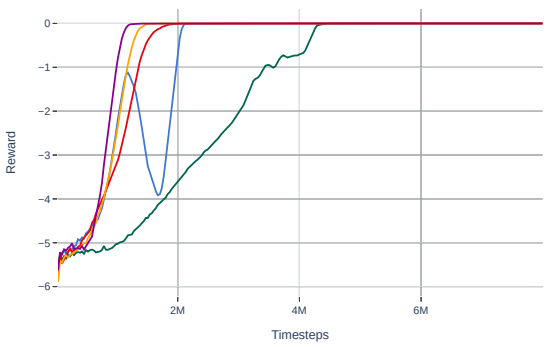
(d) Lunar lander.

Figure 2: Performance of A2C (blue), PPO2 (green) and ACKTR (red) in several OpenAI gym environments.

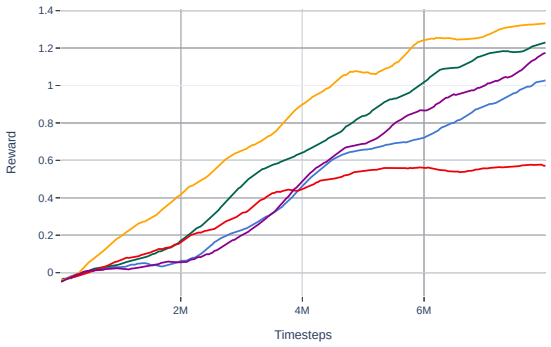
Furthermore, we run several experiments with different network architectures listed below:

- Two shared fully-connected layers of 64 nodes for policy and value function
- One shared fully-connected layers of 64 nodes for policy and value function, and one additional layer of the same size for the value function
- One shared fully-connected layers of 64 nodes for policy and value function, and one additional layer of the same size for the policy function
- One shared fully-connected layers of 64 nodes for policy and value function, one additional layer of the same size for the value function and one additional layer of the same size for the policy function
- Two separate fully-connected layers of 64 nodes for policy and value function

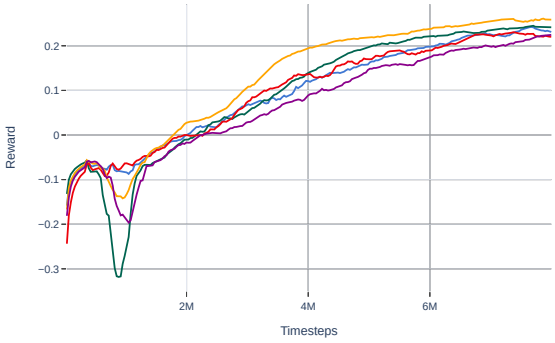
The results on Figure 3 show that the network with one shared layer followed by two separate streams for policy and value function looks the most promising architecture variant. We also experimented with using shared LSTM layer for both policy and value function, but the results showed that much more steps is required for the algorithm convergence in this case, which can be critical in case of more complicated environment that requires more time per iteration.



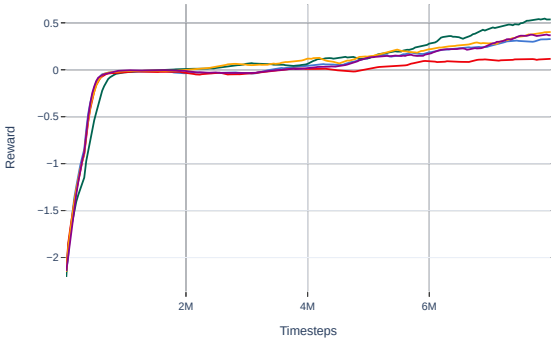
(a) Pendulum.



(b) Continuous mountain car.



(c) Bipedal walker.



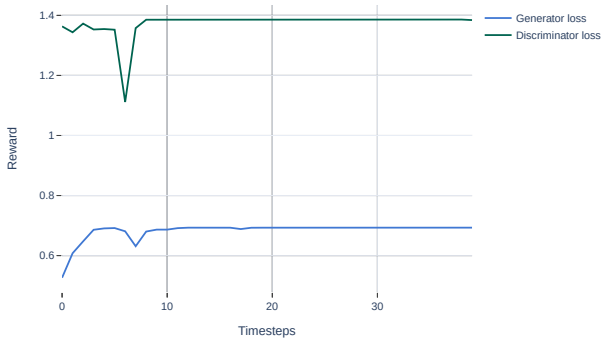
(d) Lunar lander.

Figure 3: Performance of PPO2 with different policy and value network architectures in several OpenAI gym environments. Architectures: shared MLP (blue), one separate layer for value (green), one separate MLP layer for policy (red), one shared MLP layer (yellow), separate MLPs (purple)

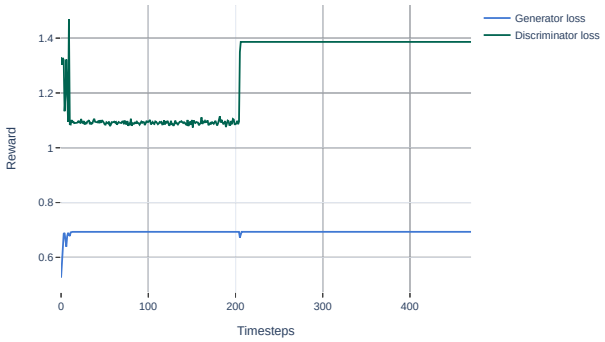
4 Adversarial generative models

Finally, we tested generative adversarial models for constructing network traffic flows. As mentioned in the introduction, we used conditional GAN. Features extracted from few previous packets of the flow are used as an input to a masked

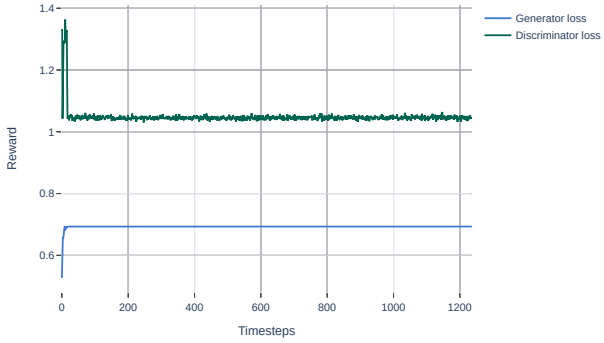
LSTM layer for the generator network. These features include direction (request or reply), inter-arrival time, payload size, TCP window size, and TCP flags The second input of the generator is a random noise vector. Outputs of these two layers are concatenated and the result is fed to an MLP, output of which is feature vector for the next packet. The discriminator network also takes features extracted from the previous packets of the flow as an input. The second input is the feature vector generated by the generator. The generator produces flows that are closer to the real flows extracted from the datasets while the discriminator network tries to determine the differences between real and fake flows. The ultimate goal is to have a generative network that can produce flows which are indistinguishable from the real ones. Unfortunately in our case, the algorithm did not converge to a solution. We also experimented with different generator and discriminator network architectures, e.g. using one-dimensional convolutional layers instead of LSTM, but the results remain almost the same. Figure 4 shows the losses of the networks for different types of the traffic.



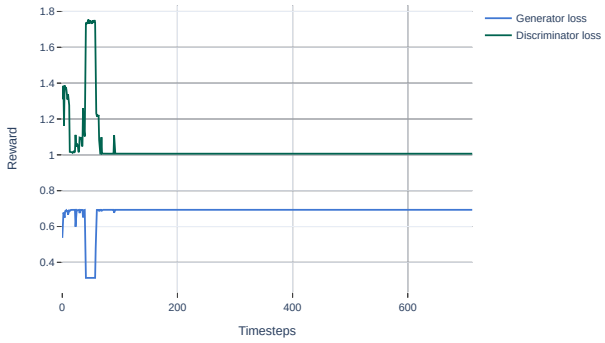
(a) Normal HTTP.



(b) Malicious HTTP.



(c) Normal SSH.



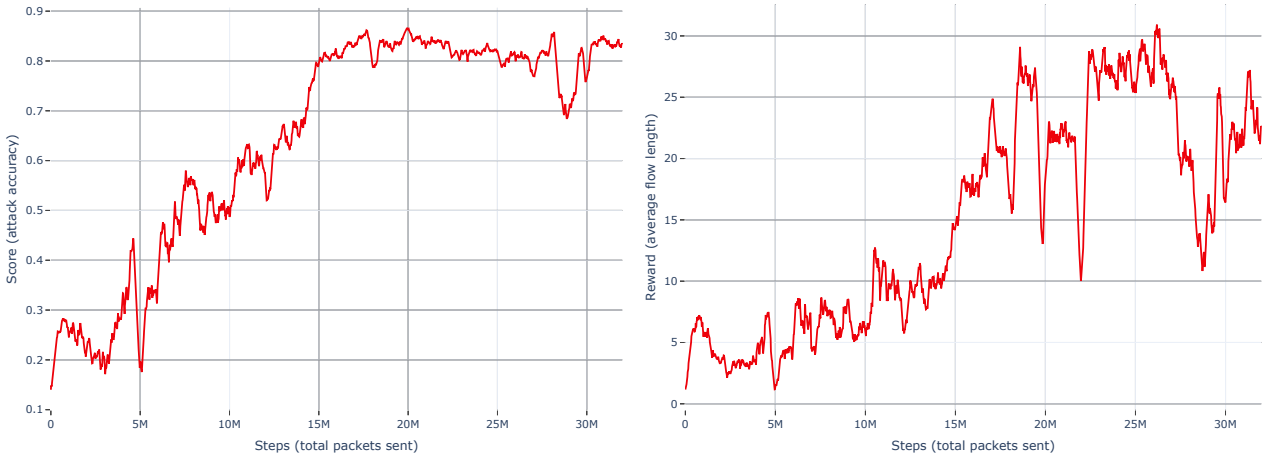
(d) Malicious SSH.

Figure 4: Performance of conditional GAN for generating different network traffic. Unfortunately neither generator nor discriminator converge.

5 Discussion and future work

In this report, we presented corrected results for supervised deep learning based intrusion detection and carried out more experiments with state-of-art reinforcement learning algorithms. We also experimented with generative adversarial networks, but unfortunately we did not manage to generate realistic network traffic flows using GAN models. We decided to abandon this approach for the traffic generation, instead we are planning to use normal traffic extracted from the datasets and deviate some of the flow parameters such as packet size, inter-arrival time, and TCP window size.

We also tried to apply reinforcement learning for spoofing a neural-network-based intrusion detection system by manipulating those flow parameters: probability of a packet to be sent in the current time window, size of the packet by padding the packet with zero bytes, time interval between two adjacent packets and size of the socket's receive buffer (in theory it should affects TCP window). The approach is to maintain a malicious TCP flow with the server as long



(a) Attack accuracy.

(b) Average flow duration.

Figure 5: Performance of PPO2 for HTTP password brute-force detection evasion. MLP is shared between policy and value functions.

as possible without it being detected. The idea is to add a packet to a malicious flow in such a way that it is classified by an IDS as normal. If at some point the flow is classified as a malicious, it is immediately blocked. In general, such approach works in the sense that the attacker becomes more and more efficient in keeping the connection alive, but the flow does not last forever. Figure 5 shows some preliminary results for the HTTP password brute-force attack against the vulnerable application as used by the authors of CICIDS2017 dataset. As it can be seen from the figure, more than 80% of the packets sent by the attacker become undetected after training the reinforcement learning agent, but sooner or later each malicious flow is classified as such anyway and it is blocked: average duration of a malicious flow is only 20 request-reply tuples which is enough for conducting the brute-force attack.

The next step is to implement both traffic generation application and intrusion detection models in form of virtualized appliances. For the former ones, we are going to use Docker containers, but for the latter ones we probably have to use virtual machines. After that, we will decide what SDN controller is the best suited for the prototype implementation. One obvious option is using POX as it is Python based and therefore we should not have difficulties adding new modules to it. The second option is using more sophisticated OpenDayLight and add new modules as external applications. By the end of this month, we are going to complete the aforementioned tasks in order to have an SDN environment ready for RL agents integration.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. Proc. of the 25th International Conference on Neural Information Processing Systems (NIPS), Vol. 1, 2012.

[2] S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural Computation, 1997.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. Proc. of the 31st International Conference on Neural Information Processing Systems (NIPS'17). pp. 6000–6010, 2017.

[4] V. Mnih, A. Badia, M. Mirza, A. Graves, T.Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. arXiv preprint arXiv:1602.01783, 2016.

[5] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel. Trust Region Policy Optimization. arXiv preprint arXiv:1502.05477, 2015.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[7] Y. Wu, E. Mansimov, S. Liao, R. Grosse and J. Ba. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. arXiv preprint arXiv:1708.05144, 2017.

[8] M. Mirza and S. Osindero. Conditional Generative Adversarial Nets. arXiv preprint arXiv:1411.1784, 2014.

[9] I. Sharafaldin, A. Lashkari, and A. Ghorbani. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. Proc of the 4th International Conference on Information Systems Security and Privacy (ICISSP), pp. 108–116, 2018.