Generalizing Transfer Learning in Reinforcement Learning with Regularizations in CoinRun

Suen Tsz Him, Håkon Osland and Théophile Leparmentier
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
thsuenab@ust.hk, heosland@ust.hk and tel@ust.hk

Abstract

In this final report, we present a new way of looking at transfer learning in Reinforcement Learning (RL) through aggressively regularizing the model by adding noise and dropout. We note that there is currently a lack of literature on the topic, and especially on reducing overfitting of RL algorithms. We hope to provide new ground for research on using a combination of regularization strategies such as Gaussian Noise Layers and Dropout layers traditionally used in deep neural networks and apply them on the context of RL using the Coinrun environment created by Open AI. At the end of the project, we are able to achieve much better performance than the baseline JERK algorithm using just Gaussian Noise regularization or Dropout, while combined strategies proved less effective. Based on what we have found, we hope to inspire new ways of applying more complex noise and dropout methods in the future.

1. Introduction

Our team seeks to explore different methods and combinations of deploying Q-learning to improve on existing performance on games called CoinRun, and at the same time, maintain a decent level of performance when the model is transferred to be used at another level of difficulty. To achieve the mentioned target, we will train the plain vanilla Nature-CNN network, a Just Enough Retained Knowledge (JERK) algorithm on the game CoinRun, add different kinds and combinations of regularizations such as dropout and noise layers; then transfer the learning the model has done and test it on other levels, maintaining the level of performance it has on the training levels. Overall, we are able to achieve nearly double the performance than the baseline JERK algorithm using just Gaussian Noise regularization, and around 1.5 times better performance using Dropout, while combined strategies performed worse than the baseline. This is likely due to over-regularization, and we will be looking into further tuning the hyperparameters, as well as other techniques in adding noise and dropout in the future to improve the performance.

2. Related Work

Our research has been inspired by the previous OpenAI competition which used Sonic the Hedgehog, its related paper "Gotta Learn Fast: A New Benchmark for Generalization in RL" (Nichol et al., 2018) [1] and two recent papers, "Quantifying Generalization in Reinforcement Learning" of Cobbe et al. [2] and "Assessing Generalization in Deep Reinforcement Learning" by Packer et al. [3]. With only 15 citations in the three combined papers, we believe there is a significant opportunity for further research in the field of a generalizable reinforcement learning algorithm and the prevention of overfitting with RL algorithms.

Efforts by Cobbe focused on using traditional efforts in regularizing deep networks, including dropout, L2 regularization, batchnorm and data augmentation. It has been successful in raising performance by solving 10+ more levels (though the exact number was not specified but rather, a graph is shown).

On the other hand, Packer focused more on observing how different algorithms would perform in a testing environment that is significantly different from the training environment, i.e. the reward structure, or the way the game is "played" could be different. We consider its most important finding to be that vanilla deep RL algorithms could perform better than more complex ones, not only able to interpolate but extrapolate to some extent as well. This is a crucial reason for us to choose the vanilla nature-CNN as one of our baselines instead of EPOpt or RL [2].

We would like to extend on these efforts of further generalizing the networks by experimenting with other ways of regularization, including adding noise layers and combining with dropout layers and maintaining performance across different levels of the game CoinRun.

3. Data

3.1. The CoinRun Environment

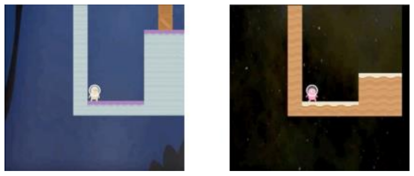


Figure 1: Two Snapshots from the Coinrun environment

We make use of the Coinrun environment designed by Open AI (2018) [4], which can be described as a simplified version of the Sonic Game. It is worth noting in particular, that 1) levels are procedurally generated (one can only proceed to the next level after finishing the current); 2) the goal of the game is to collect a single coin that lies in the end of the level; 3) colliding with any obstacle result in the agent's immediate death. Compared to Sonic the Hedgehog, the Coinrun environment has reduced complexity: 1) fewer goals - no longer maximizing a score by getting as many coins as possible, but one coin at the end of the level; 2) fewer penalties - there exist no enemies that would actively cause the agent's demise, but only passive obstacles that would cause its death when they are run into.

3.2. Data Collection and Preprocessing

Partly because of the way Reinforcement Learning is set up, where gameplay image data is collected asynchronously as the agent is experiencing, or playing the game, there was no actual additional need to collect the data from another source. The image data used in training is of size 64 * 64 * 3, taken at 24 frames per game second (a second in the environment) as the game is rendered. However, in order to stabilize the performance of the network, we scaled the image before training by dividing all pixels of the image by 255 [5].

4. Methods

We will go through three main parts in the methods section, including the details of the training, regularization and the transfer learning we are planning to do.

4.1. Training Process

The vanilla baseline models/algorithms we are using are Nature-CNN and JERK.

Nature-CNN: the default baseline in Open AI contests. It is a 3-layer convolutional network and trained on Proximal Policy Optimization (Schulman et al., 2017) [6].

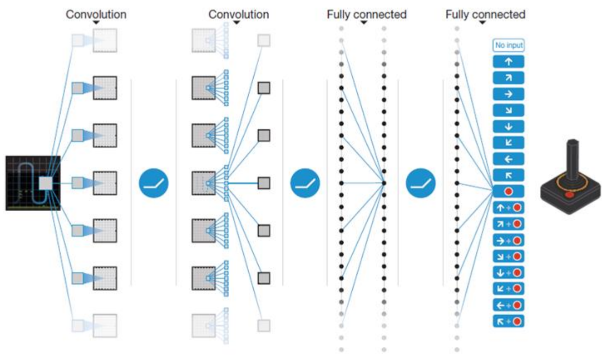
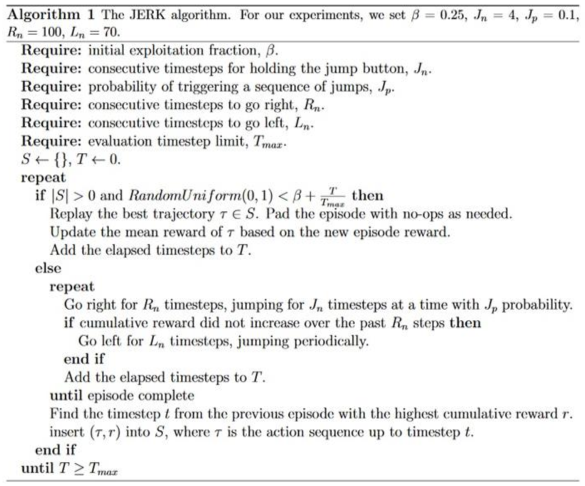


Figure 2: A simple visualization of the Nature CNN [6]

JERK: The JERK algorithm is based on the Sonic paper of Nichol et al. [1]. This algorithm is a simple one that focuses solely on awards but ignores observations. It trains quickly but has a tendency to repeat past actions that have historically rewarded the agent better than others. This causes its performance to fluctuate somewhat unpredictably since the environments are stochastic, and in the test cases, unseen.


Algorithm 1 The JERK algorithm. For our experiments, we set $\beta = 0.25$, $J_n = 4$, $J_p = 0.1$, $R_n = 100$, $L_n = 70$.

```
Require: initial exploitation fraction, $\beta$.
Require: consecutive timesteps for holding the jump button, $J_n$.
Require: probability of triggering a sequence of jumps, $J_p$.
Require: consecutive timesteps to go right, $R_n$.
Require: consecutive timesteps to go left, $L_n$.
Require: evaluation timestep limit, $T_{max}$.
$S \leftarrow \{\}$, $T \leftarrow 0$.
repeat
  if $|S| > 0$ and $RandomUniform(0,1) < \beta + \frac{T}{T_{max}}$ then
    Replay the best trajectory $\tau \in S$. Pad the episode with no-ops as needed.
    Update the mean reward of $\tau$ based on the new episode reward.
    Add the elapsed timesteps to $T$.
  else
    repeat
      Go right for $R_n$ timesteps, jumping for $J_n$ timesteps at a time with $J_p$ probability.
      if cumulative reward did not increase over the past $R_n$ steps then
        Go left for $L_n$ timesteps, jumping periodically.
      end if
      Add the elapsed timesteps to $T$.
    until episode complete
    Find the timestep $t$ from the previous episode with the highest cumulative reward $r$.
    insert $(\tau, r)$ into $S$, where $\tau$ is the action sequence up to timestep $t$.
  end if
until $T \geq T_{max}$
```

Figure 3: Setup of the JERK Algorithm [1]

Finally, our model would be then be trained on CoinRun for 10 episodes in each epoch, aiming to maximize the mean reward in the process.

4.2. Regularization Process Exploration

A dizzying array of methods to regularize deep networks has been proposed throughout the history of deep learning research [7]. We seek not to implement every method available, as it makes no significant value added to existing research; but rather, some of the newer ones that have not been thoroughly tested and mix and match to find a previously undiscovered solution applied in the context of reinforcement learning.

For this project, we have decided to integrate dropout and

Gaussian noise layers into the model to allow for more general models that would not purely function in the training environments.

First, dropout has been widely used in the machine learning community since its formulation by (Srivasta, et al., 2014) [8]. Its purpose was to randomly "drop" neurons in a layer at training time, preventing them from "co-adapting too much". Ultimately, it would be similar to sampling from a number of "smaller" networks and averaging the output, before using the normal-sized network with smaller weights on individual units to predict the output at test time. We believe this would be beneficial to our RL algorithm because 1) the test environment will be more difficult, and different from the training environment. Preventing units from co-adapting too much to the training environment, and having certain ways of action pertaining to rewards given in easier levels should prove effective; 2) it has demonstrated effectiveness in the same paper of possibly reducing error rate from 1.18% to 0.79%, a significant improvement in classifying MNIST digits.

Second, Gaussian noise layers was proposed by Diederik P Kingma and Max Welling (2013) in their paper on Auto-Encoding Variational Bayes [9]. The key here is to add some level of stochasticity in the hidden layer to the inputs and reparametrize before activating it and passing on to the output. This adds on another layer of randomness, which we believe simulates how the test environment may "surprise" the agent with things it has not experienced before.

4.3. Alternative methods

Currently, we are looking into Spectral Dropout which "drops the less significant spectral components to preserve the discriminative ability of network activations" [10]. At the same time, in contrast to "reducing noise", we would like to make use of carefully crafted noise through adding an Adversarial Noise Layer (ANL), allowing for the extraction of cleaner feature maps and further preventing overfitting (You et al., 2018) [11]. However, constructing these layers are significantly more complex and hence, we settled for a simpler implementation of noise and dropout instead.

4.4. Transfer Learning and Evaluation Process

In order to transfer the knowledge we have gained on one level and let the agent test on another, we will follow the standard procedure suggested by the Sonic paper [2].

This means we will play CoinRun for 10,000 timesteps for an arbitrary but fixed number of runs. Then, we will take the average of the total reward (levels cleared) across all runs, getting the "score" which we can evaluate our model on, and will also be the "performance" we refer to below in this paper.

We will then compare the results of our regularized model (the nature-CNN model) with the unregularized versions, as well as the JERK algorithm. We aim to prove two things: 1) transfer learning is able to have better performance than an untrained, random agent; and 2) using our combination of regularizations can improve the performance of the agent in test time.

5. Experiments

5.1. Vanilla baselines Implementation

The team has trained a nature-CNN model with the following architecture. Put simply, we had a 64 X 64 X 3 image; a first hidden convolutional layer with 32 filters of 8 X 8 with stride 4, using ReLU (Rectified Linear Unit) as activation. The second hidden layer is also a convolutional layer with 64 filters of 4 X 4 with stride 2, using ReLU as activation. Next, it has a third convolutional layer with 64 filters of 3 X 3 with stride 1 using ReLU. This is then connected to the final fully-connected hidden layer with 512 ReLU units. The output layer is a fully-connected layer with an output corresponding to the number of actions the agent could take, which is 4 here.

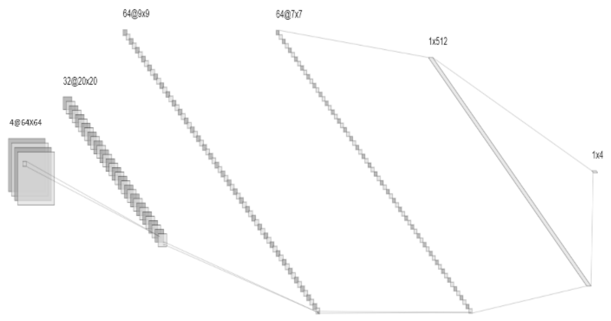


Figure 4: Architecture of nature-CNN

As we did not modify the filters nor layers of the model, it was exactly as it was proposed by Mnih [12].

Below are the hyperparameters we have used for the model:

Hyperparameters description	**Value**
Learning Rate	2.5e-4
Total Time Step(s)	5e3
Buffer Size	1e4
Exploration Fraction	0.2
Exploration Final EPS	0.02

Figure 5: Hyperparameters used by the nature-CNN

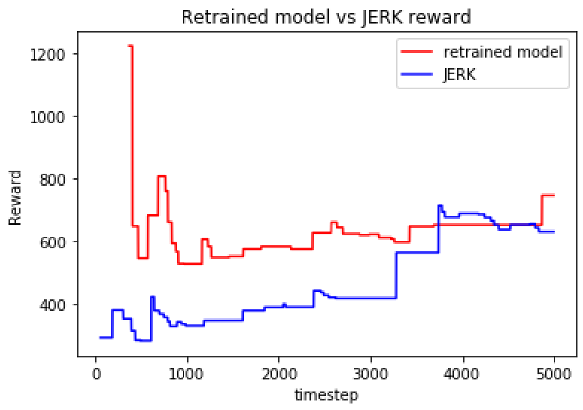


Figure 6: Vanilla Model Results [5]

In the first base implementation, we see that the base transferred model is performing better than JERK. We then proceed to add regularizations to attempt to further improve the performance of the model.

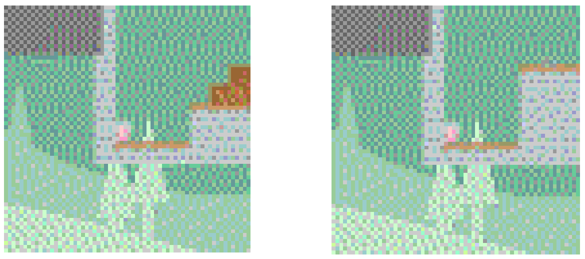


Figure 7: GIFs of actual training environment

5.2. Dropout

We experimented with different number of layers of Dropout layers, with dropout probability as the only hyperparameter, which is set to 0.5.

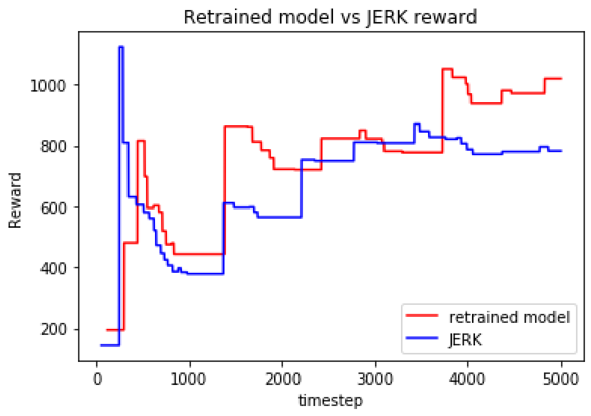


Figure 8: 1-layer of Dropout

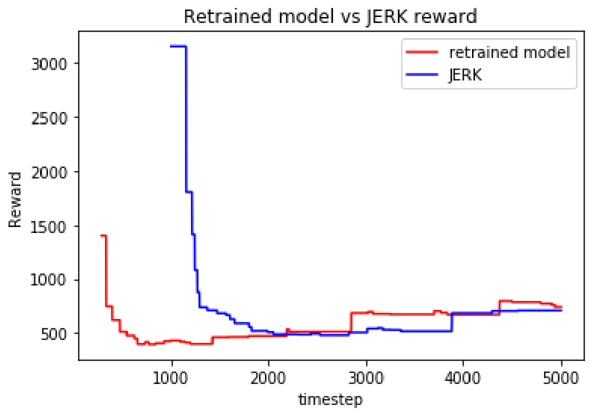


Figure 9: 2-layers of Dropout

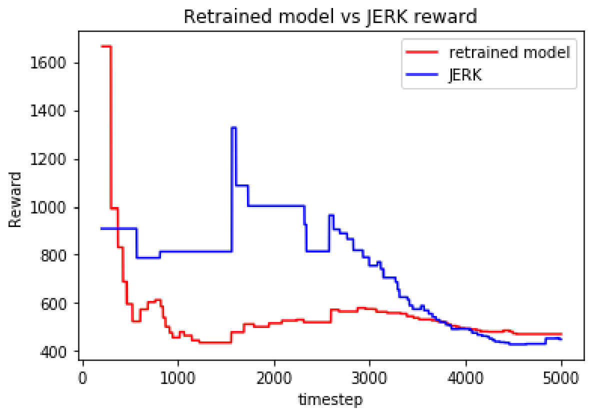


Figure 10: 3-layers of Dropout

5.3. Gaussian Noise

We experimented with different number of layers of Gaussian noise, with standard deviation of the Gaussian noise as the only hyperparameter, which is set to 0.5.

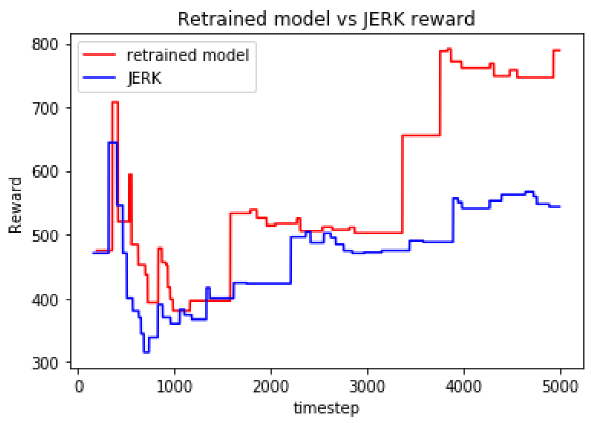


Figure 11: 1-layer of Gaussian Noise

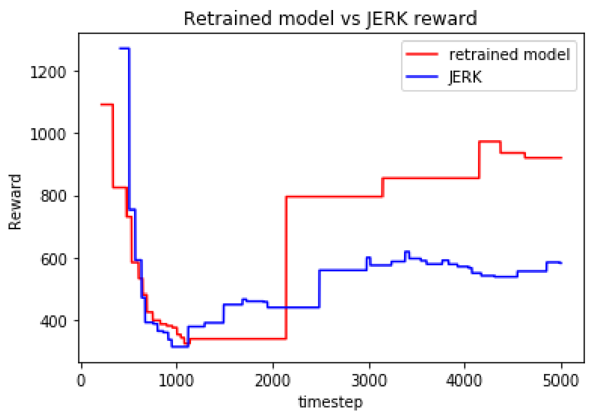


Figure 12: 2-layers of Gaussian Noise

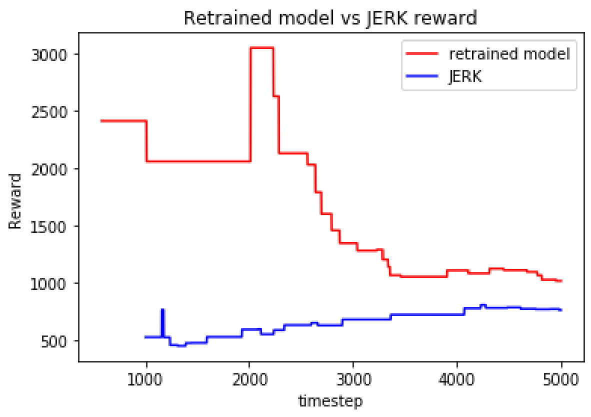


Figure 13: 3-layers of Gaussian Noise

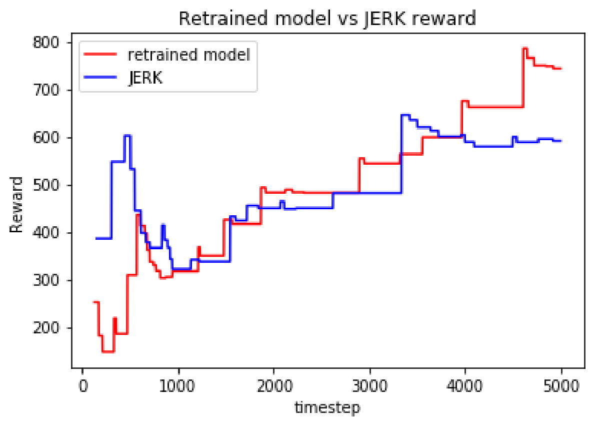


Figure 14: 1-layer of Gaussian Noise + 1-layer of dropout

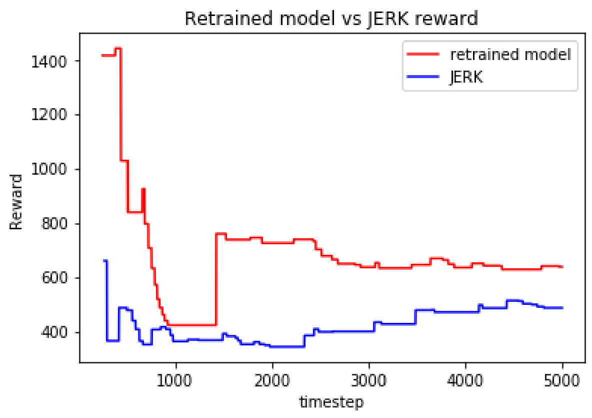


Figure 15: 2-layers of Gaussian Noise + 2-layers of dropout

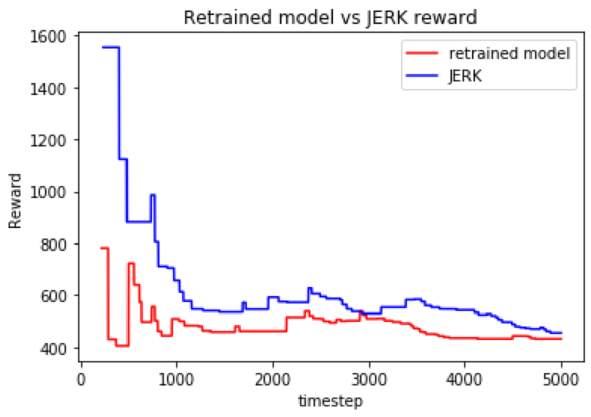


Figure 16: 3-layers of Gaussian Noise + 3-layers of dropout

5.4. Dropout + Gaussian Noise

Ultimately, we combined the methods, as it was also done in the Open AI literature [1]. However, it has worse performance than a single method in every number of layers.

5.5. Results Elaboration

Performance (mean reward)	1 layer	2 layers	3 layers
Vanilla		650	
Noise	791	936	1120
Dropout	938	788	481
Noise + Dropout	749	636	444

Figure 17: Table Summary of Results

6. Conclusion

Overall, we saw the best performer in using three layers of Gaussian noise at a standard deviation of 0.5. Based on the results, we believe in combining the noise and dropout layers, we may be over-regularizing the network, causing it to underfit instead. At the same time, we felt it was interesting that the dropout performance decreased steadily as number of layers increased, which may be due to the same reason of over-regularization.

Finally, one common phenomenon that happened across all experiments was that results plunged after the first few hundred timesteps. This requires further investigation, but we believe it may be due to insufficient hyperparameter fine-tuning.

In the future, we are looking forward to more exciting research to be done on applying more complex noise and dropout techniques we mentioned in section 4.3 to generalize the agent and allow it to perform robustly across extremely different environments. We believe that this will allow applications to go beyond games, and perform in real life scenarios where agents and bots can perhaps adapt to entirely different natural disaster environments and still mange to perform its job of saving people. We admit that what we have done in the project is a mere small step, with much of the difficult work done before us, but we are looking forward to going beyond in the near future.

All these results are also available in our Github Repository [13] and a Google Colab notebook [5].

References

[1] Cobbe, K., Klimov, O., Hesse, C., Kim, T., & Schulman, J. (2018). Quantifying Generalization in Reinforcement Learning. CoRR, abs/1812.02341.

[2] Nichol, A., Pfau, V., Hesse, C., Klimov, O., & Schulman, J. (2018). Gotta Learn Fast: A New Benchmark for Generalization in RL. CoRR, abs/1804.03720.

[3] Packer, C., Gao, K., Kos, J., Krähenbühl, P., Koltun, V., & Song, D.X. (2018). Assessing Generalization in Deep Reinforcement Learning. CoRR, abs/1810.12282.

[4] Openai. (2019, April 06). Openai/coinrun. Retrieved from https://github.com/openai/coinrun

[5] Oslan, H., Suen, E., & Leparmentier, T. (2019, May 10). Google Colaboratory. Retrieved May 10, 2019, from https://colab.research.google.com/drive/1UlJUlF7sCbRg5zRnlek5mHwBnXuBXafq

[6] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. CoRR, abs/1707.06347.

[7] Goodfellow, I., Bengio, Y., & Courville, A. (2017). Deep learning. Cambridge, MA: The MIT Press.

[8] Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov; 15(Jun):1929−1958, 2014.

[9] Kingma, D.P., & Welling, M. (2014). Auto-Encoding Variational Bayes. CoRR, abs/1312.6114.

[10] Khan, S.H., Hayat, M., & Porikli, F.M. (2018). Regularization of deep neural networks with spectral dropout. Neural networks: the official journal of the International Neural Network Society, 110, 82-90.

[11] You, Z., Ye, J., Li, K., & Wang, P. (2018). Adversarial Noise Layer: Regularize Neural Network By Adding Noise. CoRR, abs/1805.08000.

[12] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M.A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518, 529-533.

[13] Oslan. H., Leparmentier. T. & Suen, T.H. (2019, April 06). hakosl/Qlearning_project. Retrieved from https://github.com/hakosl/Qlearning_project