

**ENHANCING REINFORCEMENT LEARNING  
BEHAVIOR BY PRE-TRAINING THE MODEL**

**A THESIS SUBMITTED TO THE INSTITUTE  
OF GRADUATE STUDIES  
OF  
NEAR EAST UNIVERSITY**

**By  
Alaa Al Thafari**

**In Partial Fulfillment of the Requirements for the  
Degree of Master of Science  
In  
Computer Engineering**

**NICOSIA, 2021**

**Alaa Al Thafari: Enhancing Reinforcement Learning Behavior by Pre-Training the Model**

**Approval of Director of Institute of Graduate Studies**

**Prof. Dr. Hüsnü Can Başer**

**We certify this thesis is satisfactory for the award of the degree of Masters of Sciences in  
Computer Engineering**

**Examining Committee in Charge:**

Prof.Dr. Rahib H. Abiyev

Committee Chairman, Department of Computer  
Engineering, NEU

Assoc. Prof. Dr. Kamil Dimililer

Department of Automotive Engineering, NEU

Assit.Prof. Dr. Elbrus Imanov

Supervisor, Department of Computer Engineering,  
NEU



I certify that this research work entitled “Enhancing Reinforcement Learning Behavior by Pre-Training the Model” is my own work. No portion of the work presented in this research report has been submitted in support of another award or qualification either at this institution or elsewhere. Where material has been used from other sources it has been properly acknowledged / referred. If any part of this project is proved to be copied or found to be a report of some other, I will stand by the consequences.

Alaa Al Thafari

Signature

A handwritten signature in black ink, appearing to read 'Alaa Al Thafari', written in a cursive style.

Date

25/8/2021

## ACKNOWLEDGEMENTS

To my supervisor Dr. Elbrus Imanov who had enough faith and trust in me and my work to keep me going,

To my mother who supported me and sacrificed with a lot for this moment.

To my uncle Tarek who has the credits of me being here more than a lot of my teacher.

To my aunt Amel for her support and her efforts in making this possible.

To my advisor and chariman Dr. Rahib Abiyev.

To my family for believing in me and looking forward to more achievements in my life.

To my family in Cyprus Şefik and Duygu.

To Dr. Murat Özgören who treated me like a son and the entire NERITA family for their support, help and providing all the help I needed.

To my colleagues and friends Abdullahi Isma'il, Qussai Toumeh and Ezekiel T. Ogidan for their continuous help.

To all the friends who shared any moment with me during the period of this degree.

## ABSTRACT

Deep reinforcement learning has performed extraordinarily in sequential tasks where they are augmented with deep neural networks as an approximation function and when the learning process is based on the immediate environment. A drawback to this however is that the deep RL has to learn features for every state from the environment and must also learn a policy. As a result, deep RLs can be time intensive and computationally expensive to train. They also require very large amounts of data for training. These reasons make deep RLs largely unsuitable for many applications.

In this research, training has been sped up by attempting to solve the issue of features learning, solving a substantial part of the problem and reducing the work that needs to be done by the deep RL. It proves that the use of small dataset demonstrated by a non-expert human actor, with supervised learning during the pre-training phase results in significant improvement in both time and performance. This work has been empirically evaluated using the deep Q-network in the AI control environment Cart-Pole. The results show that having a trained policy network serves in providing a major improvement with the time required for training my deep RL model, even when data used is small and not perfect.

***Keywords:*** Reinforcement Learning, Q-Learning, Deep Learning, Pre-training

## TABLE OF CONTENTS

<b>ACKNOWLEDGEMENTS</b> .....	i
<b>ABSTRACT</b> .....	ii
<b>TABLE OF CONTENTS</b> .....	iii
<b>TABLE OF FIGURES</b> .....	v
<b>TABLE OF TABLES</b> .....	vi
<b>LIST OF ABBREVIATIONS</b> .....	vii
<b>CHAPTER 1 INTRODUCTION</b> .....	1
<b>1.1 Thesis Overview</b> .....	4
<b>CHAPTER 2 LITERATURE REVIEW</b> .....	6
<b>CHAPTER 3 OVERVIEW</b> .....	9
<b>3.1 Reinforcement Learning</b> .....	9
<b>3.2 Deep Reinforcement Learning</b> .....	10
<b>3.2.1 Model-Based</b> .....	11
<b>3.2.2 Model-Free</b> .....	11
<b>CHAPTER 4 METHODOLOGY</b> .....	13
<b>4.1 Design</b> .....	13
<b>4.1.1 Environment Setup</b> .....	15
<b>4.1.2 Memory Reply</b> .....	16
<b>4.1.3 Q-Table</b> .....	17
<b>4.1.4 Strategy</b> .....	17
<b>4.1.5 Agent</b> .....	18
<b>4.1.6 Supervised Training</b> .....	18

<b>4.2 Reinforcement Learning Algorithm</b> .....	19
<b>4.3 Data Collection</b> .....	22
<b>4.4 Supervised Learning</b> .....	23
<b>CHAPTER 5 IMPLEMENTATION RESULTS</b> .....	25
<b>5.1 Results</b> .....	25
<b>5.2 First Experiment (Reinforcement Learning)</b> .....	27
<b>5.3 Second Experiment (Reinforcement Learning + Supervised Learning)</b> .....	28
<b>CHAPTER 6 CONCLUSION</b> .....	31
<b>REFERENCES</b> .....	32
<b>APPENDICES</b> .....	34
<b>APPENDIX 1 DATA COLLECTION CODE</b> .....	35
<b>APPENDIX 2 REINFORCEMENT LEARNING CODE</b> .....	39
<b>APPENDIX 3 REINFORCEMENT LEARNING + SUPERVISED LEARNING CODE</b> .....	50
<b>APPENDIX 4. ETHICAL APPROVAL LETTER</b> .....	62
<b>APPENDIX 5. SIMILARITY</b> .....	63

## TABLE OF FIGURES

<b>Figure 1.1 - Differences between supervised learning, unsupervised learning and reinforcement learning .....</b>	<b>1</b>
<b>Figure 3.1 - Block Diagram of Reinforcement Learning Concept.....</b>	<b>9</b>
<b>Figure 4.1 - Block Diagram of the Design Stage.....</b>	<b>13</b>
<b>Figure 4.2 - Line Plot of Rectified Linear Activation for Negative and Positive Inputs.....</b>	<b>15</b>
<b>Figure 4.3 - Cart-Pole Environment Visualization .....</b>	<b>16</b>
<b>Figure 4.4 - Block Diagram of Epsilon Greedy Strategy .....</b>	<b>18</b>
<b>Figure 4.5 - Flow Chart of RL Algorithm.....</b>	<b>21</b>
<b>Figure 4.6 - Data Collection Flow Chart.....</b>	<b>22</b>
<b>Figure 4.7 - Cross-Entropy Gradient Descent.....</b>	<b>24</b>
<b>Figure 4.8 - Block Diagram of the Design Stage Including the Supervised Learning .....</b>	<b>24</b>
<b>Figure 5.1 - Model Summaries.....</b>	<b>25</b>
<b>Figure 5.2 - Reinforcement Learning Results.....</b>	<b>27</b>
<b>Figure 5.3 - Reinforcement Learning + Supervised Learning Results.....</b>	<b>29</b>



## TABLE OF TABLES

<b>Table 1 - Hyper Parameters' Values .....</b>	<b>26</b>
---	-----------

## LIST OF ABBREVIATIONS

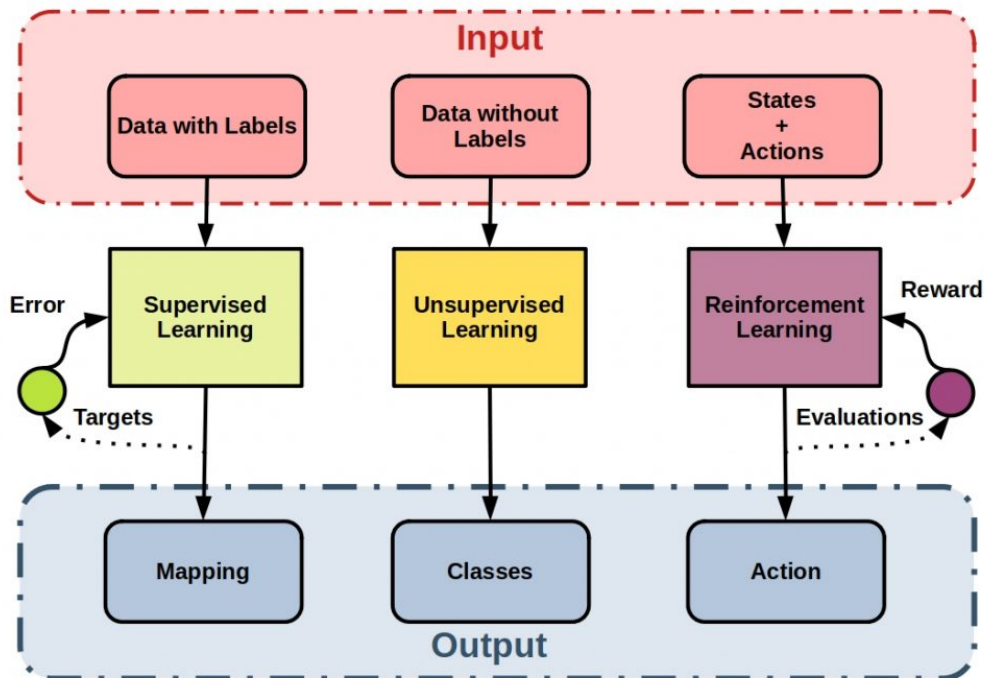
<b>AI</b>	Artificial Intelligence
<b>RL</b>	Reinforcement Learning
<b>ML</b>	Machine Learning
<b>DRL</b>	Deep Reinforcement Learning
<b>DL</b>	Deep Learning
<b>NLP</b>	Natural Language Processing
<b>CV</b>	Computer Vision
<b>SGD</b>	Stochastic Gradient Descent

# CHAPTER 1

## INTRODUCTION

Reinforcement learning is an area of machine learning focused on how the intelligence system acts in an environment. It has the objective of making a series of decisions that would yield the best rewards in an arbitrary and complex environment. The model has to figure out by itself how to maximize the intended output or reward and adjusts its decisions based on feedback.

Conventional machine learning algorithms based on supervised or unsupervised learning usually require huge datasets to work efficiently and its work is only limited to the environment that it was trained at. Any other form of input that the model is not familiar with cannot be processed without preprocessing actions which makes it difficult to handle arbitrary data.



**Figure 1.1:** Differences between supervised learning, unsupervised learning and reinforcement learning

New artificial intelligence systems are trying to have customized trained products. This means the product will be adaptive and learn from to suit the environment that it's working at without the need for professional setup, and saving the time wasted on studying the environment and training the AI model to.

This helps creating adaptive products that can be trained and customized based on the users need, the reinforcement learning provides the solution to this problem, and helps in adapting the system and updating its performance while working to have the best performance if it encounters any change in the environment, without the need to stop the system and retrain it with every change.

However, the performance of reinforcement learning is still slow and requires a lot of time to gain an acceptable performance, this raises a very important problem especially in the fields where time is expensive.

The advantage of RL is that the model can learn a certain feature even when provided with an incomplete representation of its current state. As seen in Figure 1.1, single value in the shape of a penalty or reward is returned as a training signal and this feedback also represents the quality of the model's performance.

In the supervised learning method, the agent trains through a series of states that provide recommended actions as training signals. The output is the classification of the inputs into groups of similar entities or objects.

In RL, agents learn by exploring their environment and then develop the different approaches to perform the intended task. It releases the developer from the role of having to know the best approach for each state the model may encounter as this is also developed in the training process. The downside to having this freedom however is that RL often has a tendency of being too slow to converge.

One of the most recent developments of neural networks in RL is the advent of Deep Reinforcement Learning. This is an approach where deep neural networks are used for function approximation.

However, to help make deep RL successful and useful in real-world projects, as it is

successful in the virtual world, the long training time that is required to learn a policy must be addressed.

Deep RL suffers from bad performance when initialized, along with the longer time it takes to learn because it has to learn the features from the environment directly instead of human engineered features. Apart from having to learn the policy, deep RL must learn to build applicable high-level features from the environment.

These problems become important in real-life applications like in finance and medicine where data can often be expensive.

Demonstrating with humans is one method to enhance deep RL learning time.

The use of human experience in RL is not a new approach but it recently became popular as a possible way of reducing the training time of deep RL.

By solving the two problems deep RL is trying to tackle, speeding it up can be achieved:

1. Feature's learning
2. Policy learning

In this research the main goal is to solve the problem of the time needed for features learning by pre-training the model to learn from dataset generated from the same environment.

This study proves that learning better features results in having better performance from the RL agent, without changing the strategy used in policy learning.

A well-known technique was applied to deep RL, this technique is widely used to enhance the training time in deep learning. However, the success of this technique in deep supervised learning depends on the datasets available and used to pre-train the networks.

In deep RL, data are unavailable or difficult to find most of the time.

In this research, an approach was proposed to enhance deep reinforcement learning algorithms learning time, that requires little non-expert experience data.

This approach starts applying supervised learning using human demonstrations to pre-train the deep neural network.

It has been proven that through this step the agent will learn to imitate the human actor [1].

However, in this case, pre-training the agent would implicitly learn the environment features through supervised learning.

This approach has been tested on a well-known deep RL algorithm called the Deep Q-Network, and its performance was evaluated in the AI environment Cart-Pole. The test included two cases. The first one is running the model directly and the second one is running the model after training the policy network.

The results show speed up in the learning time and the enhancement in the performance by comparing both the run time and the moving average of both test cases. The moving average is used as a parameter for analysis here as it helps us to see clearly the long term trends and smooths out any inconsequential fluctuations in the results [14].

The main feature of this approach is being generic, which means it can be applied to multiple deep RL algorithms.

## **1.1 Thesis Overview**

This work is structured as follows:

- Chapter 2 reviews the current state and developments on the study. Here some research papers are analyzed and discussed to help better understand the research problem.
- Chapter 3 an overview about reinforcement learning and deep reinforcement learning.
- Chapter 4 discusses the methodology of my work. Here, I discuss everything from data gathering to data preprocessing. I also discuss the fine details of the neural network used and its parameters. I also discuss the evaluation metrics used to measure my results.
- Chapter 5 discusses the results from my work and highlights the results for evaluation metrics from the classification task carried out.

- Chapter 6 concludes the work with a summary of the methods employed and how they affect results obtained. It also discusses the future perspectives of this research.

## CHAPTER 2

### LITERATURE REVIEW

Reinforcement learning poses a unique and impressive approach to the problems of control theory - using data to improve future decision making in dynamic and sometimes unstable environment [7]. As shown in the first chapter of this thesis, RL poses the optimal machine learning approach to solving control theory problems in an environment where the states are unknown and dynamic. While conventional control theory seemingly solves this problem already, it does so under a number of assumptions such as linearity and time-invariance, which might not always be the case in a real world environment [8]. The RL approach to this problem allows features to be learned and policies to be developed under any conditions, solving the limiting issues of conventional control theory. With RL, this can be done even if the analytical model is not already known. For these reasons, RL is the preferred choice for control systems like in robotics [12].

While this is the case, the RL approach also has its own issues. The Q-learning algorithm proposed by Watkins in the RL community is often used in artificial intelligence and robotics for its powerful ability to learn in complex and unknown environments. It is an off policy reinforcement learning algorithm and so makes decisions without the need for an already existing policy, focusing on primarily on optimizing the reward [9]. The fact that decisions are made without consideration of any already existing policies means that if a poor action is selected, it does not affect the Q-function of the current state [15]. The method of updating the Q value is shown below.

$$Q(s, a) \leftarrow Q(s, a) + \alpha[R + \gamma \max_{a'} Q(s, a') - Q(s, a)]$$

$\alpha$  represents the learning rate and R is the reward.

However, some of the problems are not solved by the real-world application. One of the difficult tasks is maintaining the right balance between “exploration” and “exploitation”, the other is “the curse of dimensionality” problem.

These main issues are getting huge attention in various fields and have seen considerable



levels of research. These research efforts put forth a number of approaches to solving the problem such as multilayer and recurrent neural networks, generalized neural networks and back propagation [10].

In Q-learning, the exploration strategies of  $\epsilon$ -greedy and Softmax have been used more often and have proven to be effective [2].

Some other works tried to combine the fuzzy control systems with the reinforcement learning and the problems risen related to nonlinearities and uncertainties in the system were solved. However, although the study fixed the high requirements needed with external disturbance, accuracy and robustness [3], it didn't solve the time needed to get a decent result in a short amount of time or number of simulations.

An example of real-world projects is self-driving cars, a self-driving car can recognize the data collected from the sensors and makes a decision based on this data, it is important for the agent to determine the current state for it to make a correct decision. However, after the agent has fully explored the space of states, the RL can determine the value of the, which means in RL the agent must search all possible behaviors, this results in consuming longer learning time [11]. In this example, this could have serious implications.

The work proposed a pre-training framework that pre-initialize the agent's policy gradient neural network based on behaviors from pre-learned images or human experts processed images [4]. However, this still requires a lot of data that will be hard to collect if found or the creation of a special data which will be expensive in terms of time and other resources since it relies on experts.

Another work exhibited that administered pre-training can be accomplished utilizing sparse variational dropout regularization, and proved it speeds up basic RL compared to basic regularizes like  $L_2$ .

Decreases your dependence on massive amount of demonstrations, or on having preparing dataset that investigates a huge state space to accomplish great outcomes [5]. This work did solve the time-consuming issue of the reinforcement learning but it raised another problem which is the noise tuning of the data used in the training, using poorly tuned noise affected the

learning rates.

Another noteworthy work to discuss featured an approach done by using human demonstrations to play some Atari games, the research used humans to play some of the games and recorded their actions to be used as training data [6], the same approach was applied in this study but using a different platform. They were able to prove that pretraining the RL model significantly reduced the training time and that these significant improvement could be achieved even if the data used for the pretraining process was small.

The platform in this study can be run using both continuous and discrete actions, and because it's closer to real-world problem, the problems addressed in [6] such as sparsity of some action and the actions being closely related are solved.

Reinforcement learning has been used in some very complex problems and as such has become the preferred approach to solving control theory problems.

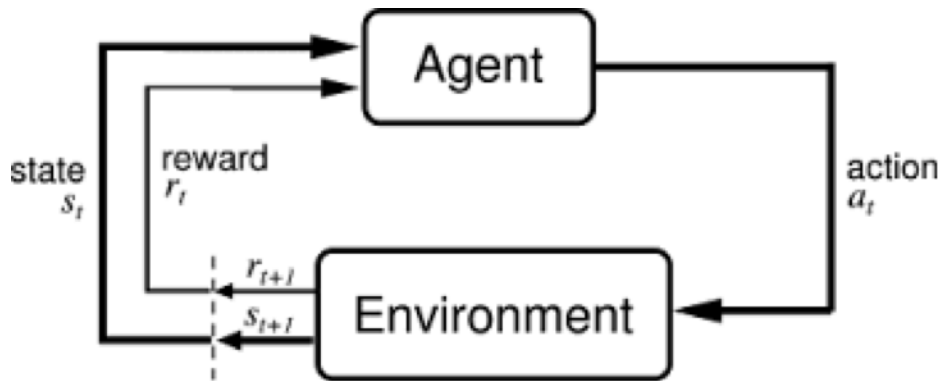
## CHAPTER 3 OVERVIEW

### 3.1 Reinforcement Learning

RL is a subfield of ML, it is considered in the computational agents learning to achieve the best results on its own in a complex and uncertain environment.

The main difference between RL and supervised learning, is that RL doesn't need labelled input/output pairs to operate.

The agent executes trial and error behavior to find a solution to the given problem, through these trial and errors, the agent gets either rewards or penalties depends on how close was the action to the solution.



**Figure 3.1:** Block Diagram of Reinforcement Learning Concept

The agent recognizes the solution based on a policy defined by the designer of the problem.

The main goal of RL is that the agent should attempt to learn the optimal policy to maximize its rewards.

The two main strengths of RL are:

1. Performance optimizing using samples.
2. Large environments handling using approximation functions.

Which makes RL useable in the following situations:

- Environment is modeled but no analytic solution.
- Only an environment simulation is modeled.
- Interaction with the environment is required to get information about it.

The main property needs to be well tuned is the exploration-exploitation rate, this rate represents the amount of states where the agent will attempt to explore the results of new actions, instead of relying on previous explored experiences.

The major challenge with reinforcement learning is the design of the simulation environment, it's a very important task because the task performed will be highly depended on it.

Reinforcement learning is applied to many fields such as:

- Game development
- Control and automation
- Operational research
- Economics
- Robot control
- Telecommunications

### **3.2 Deep Reinforcement Learning**

DRL is a subfield of ML, it assemble the RL with DL.

The main advantage of deep RL over RL, is that deep RL uses deep learning as a part of the solution, this allows agents to decide based on unstructured data without manual engineering

of the state space.

The reinforcement learning problems are usually modeled using Markov Decision Process (MDP) where the agent takes a single action at every timestep, the states of MDP in many decision-making problems are high-dimensional which makes unsolvable by RL. Therefore, DRL algorithms uses DL combined with reinforcement learning to solve this kind of problems.

The neural network is an approximator, it is used with RL to approximate what we call policy function. Because the neural network can map (state, action) pairs to Q-values instead of having to have a lookup table for all possible states and their values.

The two main techniques used to train RL policies are:

- Model-based
- Model-free

### **3.2.1 Model-Based**

In this technique, DRL evaluates advanced model of the environment's dynamics, this is often achieved using SL. After that using the learned model, actions are obtained.

### **3.2.2 Model-Free**

This technique teaches the policy without explicitly modeling the forward dynamics.

By estimating the policy gradient directly, the policy can be improved to maximize the returns. However, it will suffer from high variance which will makes it unsuitable for function approximation in deep RL.

Another class of model-free based algorithms relies on dynamic programming, which is inspired by interim difference learning and Q-learning.

This kind of algorithms often learns the neural network's Q-function, which estimates the returns based on the action taken from the state in discrete action spaces.

In continuous action spaces, in usual these algorithms do learn the policy and the value estimate.

The deep RL is applied in several fields like:

- Robots
- Games
- NLP
- CV
- Schools
- Transportation
- Finance
- Medical care

## CHAPTER 4

### METHODOLOGY

#### 4.1 Design

The design of this study is a software implementation to compare the performance of the reinforcement learning only, and the performance of the reinforcement learning after pre-training the policy network using human demonstrated data.

The approach will be tested for both study cases, using a deep Q linear network for simplicity, and on an open-source AI control platform called Cart-Pole to make the results generic.

Two case studies were compared in this study, the following is a description of each case study.

Both case studies use the same AI model, which consists of the following layers:

1. Input layer: has two inputs represents the width and height of the frame.
2. First hidden layer: is a linear hidden layer that has three filters for each color channel, thus the input of this layer is  $3 \times \text{width} \times \text{height}$  and the output was set to 24 output features.
3. Second hidden layer: is another linear hidden layer that has 24 input features and 32 output features.
4. Output layer: finalize the decision to either one of two states.



**Figure 4.1:** Block Diagram of the Design Stage

The process using the forward function is the same in both case studies including the activation functions of the layers, and it goes as follows:

1. It starts by flattening the input into a single dimension vector to be processed in a linear way later through the hidden layers.
2. The first hidden layer has a "ReLU" activation function.
3. The second hidden layer also has a "ReLU" activation function.
4. The last step is to output the result value.

The ReLU activation function is defined as the positive part of its arguments.

$$f(x) = x^+ = \max(0, x)$$

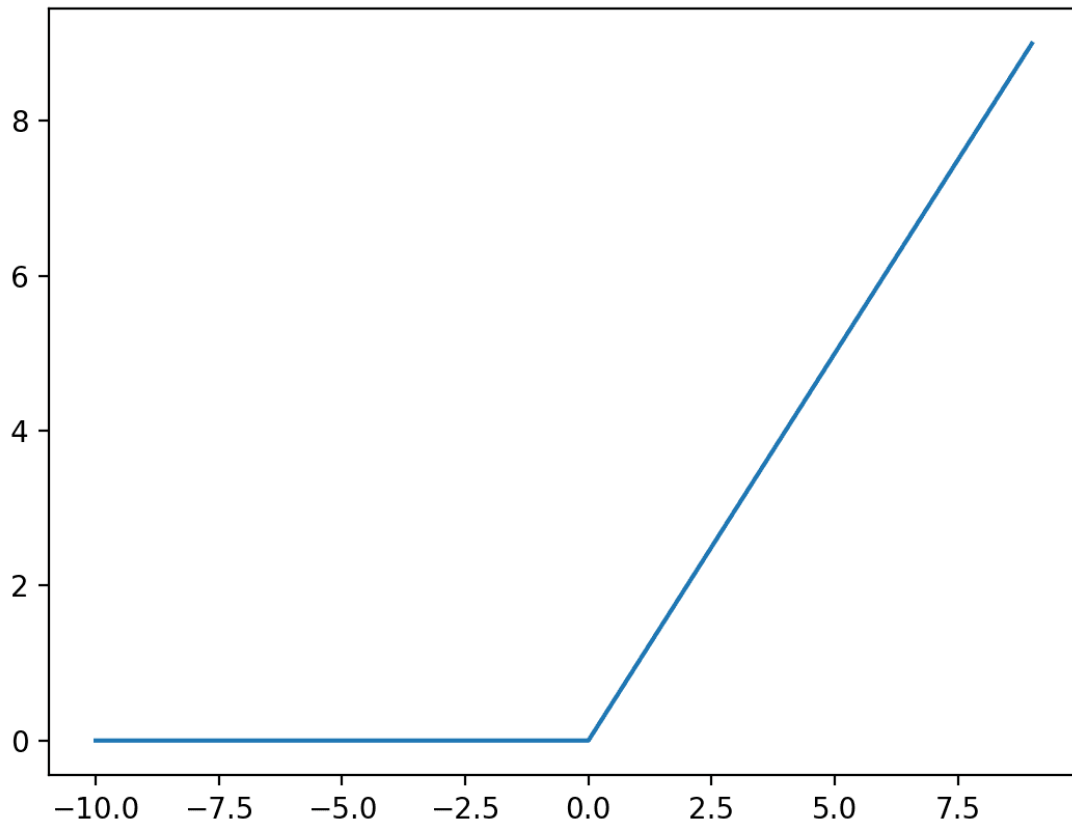
Where  $x$  is the neuron's input. There are a number of other activation functions, such as Sigmoid, Tanh, etc as well as activation functions specifically used for output layers, such as Softmax, and they all have advantages that make them best suited for specific tasks and applications [13].

There are several advantages for using the ReLU function:

- Spares activation, so only about 50% of the hidden neurons are activated.
- Providing better gradient propagation and fewer vanishing gradient problems.
- Efficient computation.
- Scale-invariant.

Both hidden layers had the ReLU activation function because it makes the model easier to train.





**Figure 4.2:** Line Plot of Rectified Linear Activation for Negative and Positive Inputs

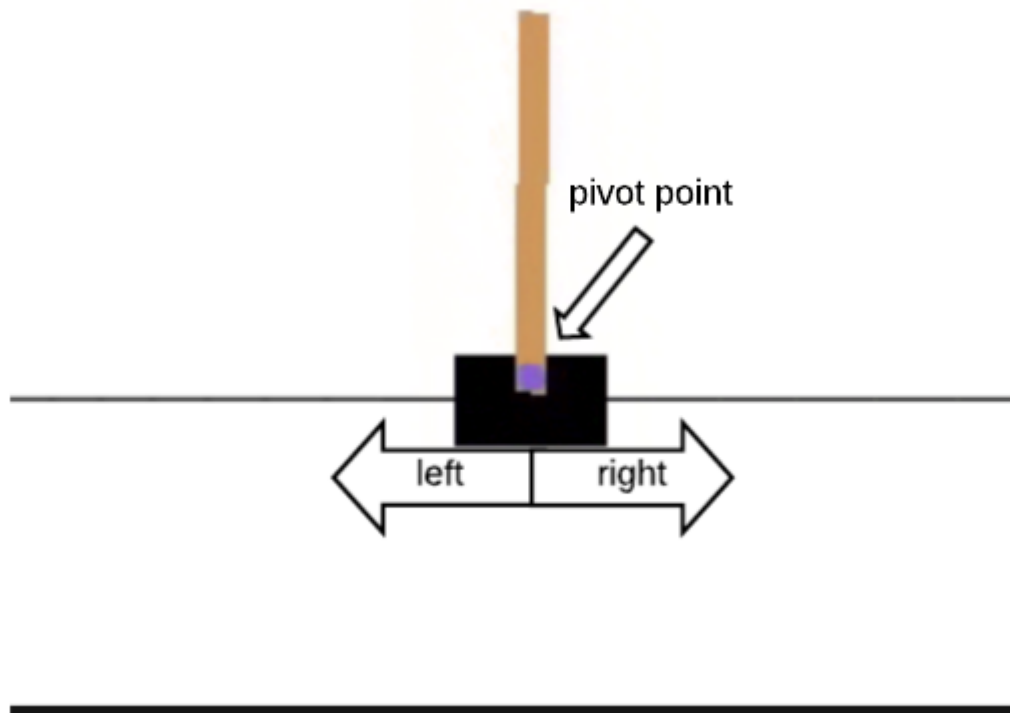
Since this is a small problem used to get generic results, the training and testing methods had to be kept as simple as possible to avoid any overfitting issues and to get the most accurate results.

#### **4.1.1 Environment Setup**

The environment used is the open source AI environment Cart-Pole, this environment is very basic and helps generalize the results of this study.

The environment simply can be described as a cart holding a vertical pole, the pole is fixed to the cart through a pivot point, this point allows the pole to have one degree of freedom around that point.

The agent's task is to keep the pole in upright position for as long as possible by extracting the state features and make a decision on whether to push the cart to the right or left, and how much force should be applied.



**Figure 2.3:** Cart-Pole Environment Visualization

A class will be defined to manage the environment, the main tasks of this class are:

1. Create and close the environment.
2. Provide the state pre-processing function.
3. Handle the actions choosing process.

#### **4.1.2 Memory Reply**

The memory reply is a class used to create an object to contain the (state, action) pairs, these values will be used later during the run for the agent to decide the optimal activity to execute dependent on its present state.

Each pair will be update as the agent learn more to enhance its performance.

### 4.1.3 Q-Table

The Q-Values generated by the agent on the run will be based on the (state, action) pairs stored in the memory.

The Q-Values class will take the current (state, action) pair and generate a Q-value based on it, it will go through the memory and check for the (state, action) pair associated with the highest Q-value.

These Q-values represent the reward, as the agent's task is to maximize its rewards.

After finding the maximum Q-value, the class will return the action that generates the highest reward.

### 4.1.4 Strategy

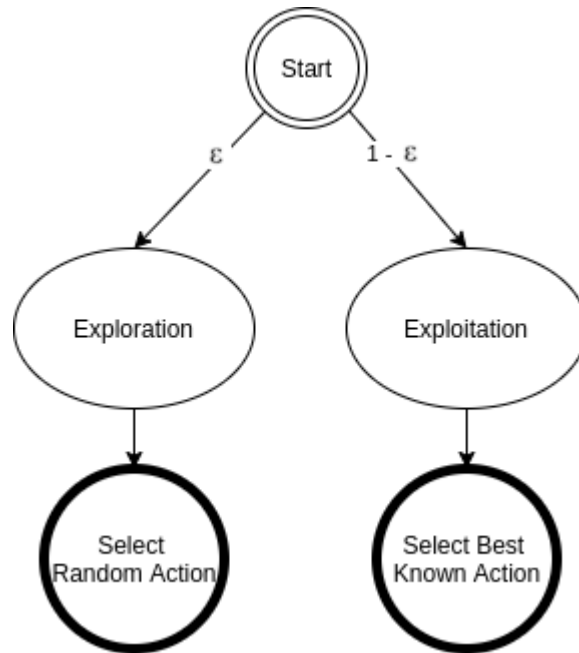
The strategy is used to calculate the exploration rate of the agent.

The exploration rate determines when and how much should the agent rely on the (state, action) pairs stored in the memory, and when to explore a new (state, action) pair.

The smaller the value generated, the more the agent tends to explore new states.

The strategy used in this study is the  $\epsilon$ -Greedy. The  $\epsilon$ -Greedy is one of the easiest strategies and it seeks the maximum reward, which means it prefers exploitation over exploration most of the time. This is a useful feature for the study purpose since the main aim is to put the training from a human demonstration in use later.

During the exploration phase, the  $\epsilon$ -Greedy strategy selects a random action and by so it increases the chances of getting an action with a higher reward.



**Figure 4.4:** Block Diagram of Epsilon Greedy Strategy

#### 4.1.5 Agent

The agent is a class responsible of selecting the next action, it uses the strategy and the number of actions available in every state.

It uses the value generated by the strategy to determine whether to explore or exploit the (state, action) pairs stored in the memory.

#### 4.1.6 Supervised Training

The supervised learning in this study is applied to the policy network in the second study case before starting the RL algorithm.

In this study a small number of data samples was used, and it's been trained for a small number of episodes.

## 4.2 Reinforcement Learning Algorithm

The reinforcement learning process starts by creating the environment using the environment class, all hyper parameters will be defined in this stage.

The optimizer to update the network weights will be defined here. In this study the optimizer used is Adam optimizer.

Adam optimizer is a variant of SGD optimizers, this redefines the slope computed over the entire dataset and can be considered as a stochastic approximation optimization or gradient descent optimization. This helps in reducing the computational expenses and time consumed in trade for lower convergence rate.

ML considers the minimization of an object function has the form of sum.

$$Q(\omega) = \frac{1}{n} \sum_{i=1}^n Q_i(\omega)$$

Where  $\omega$  is a parameter that minimizes  $Q(\omega)$  to be estimated.

Gradient descent method performs the following iteration:

$$\omega := \omega - \eta \nabla Q(\omega) = \omega - \frac{\eta}{n} \sum_{i=1}^n \nabla Q_i(\omega)$$

Where  $Q_i(\omega)$  in this case is the value of the loss function at  $i$ -th example,  $Q(\omega)$  is the empirical risk, and  $\eta$  is the learning rate.

Adam optimizer is short for “Adaptive Moment” is a variant and update of SGD.

Adam uses a moving average to plot the slope and second moments.

Adam's parameter redevelopment is shown in the following equation.

$$m_{\omega}^{(t+1)} \leftarrow \beta_1 m_{\omega}^{(t)} + (1 - \beta_1) \nabla_{\omega} L^{(t)}$$

$$v_{\omega}^{(t+1)} \leftarrow \beta_2 v_{\omega}^{(t)} + (1 - \beta_2) (\nabla_{\omega} L^{(t)})^2$$

$$\hat{m}_{\omega} = \frac{m_{\omega}^{(t+1)}}{1 - \beta_1^{t+1}}$$

$$\hat{v}_{\omega} = \frac{v_{\omega}^{(t+1)}}{1 - \beta_2^{t+1}}$$

$$\omega^{(t+1)} \leftarrow \omega^t - \eta \frac{\hat{m}_{\omega}}{\sqrt{\hat{v}_{\omega} + \epsilon}}$$

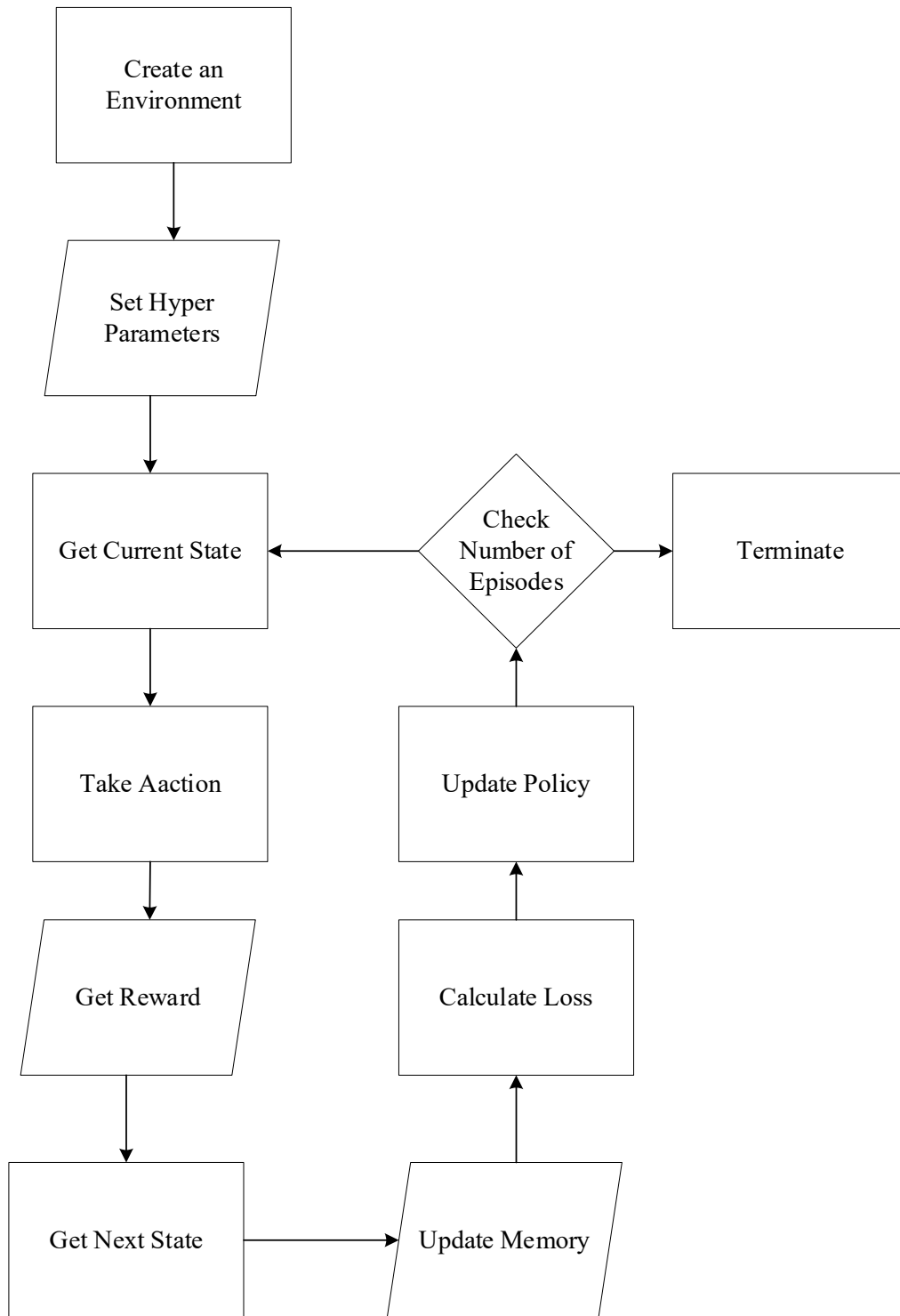
Where  $\omega^t$  is a given parameter,  $L^{(t)}$  is a loss function,  $\epsilon$  is a small number used to avoid division by zero, and  $\beta_1$  and  $\beta_2$  are the forgetting elements of gradients and second moments of gradients.

In this study Adam optimizer was used for many reasons, mentioning some below:

1. Straightforward implementation.
2. Efficient and fast.
3. Doesn't require a lot of memory.
4. Invariant to diagonal rescale of the gradients.
5. Adaptive Gradient Algorithm.

Then this process retrieves the current state of the environment and provides the states and policy to the agent to get an action.

After getting the action, it gets the reward based on that action, the updates the memory and Q-values, it calculates the loss which represents error rate from that action in the following state and it's used by the optimizer to update the policy to gain better results.



**Figure 4.5:** Flow Chart of RL Algorithm

### 4.3 Data Collection

The data for this problem just like most of the RL problems, was not available.

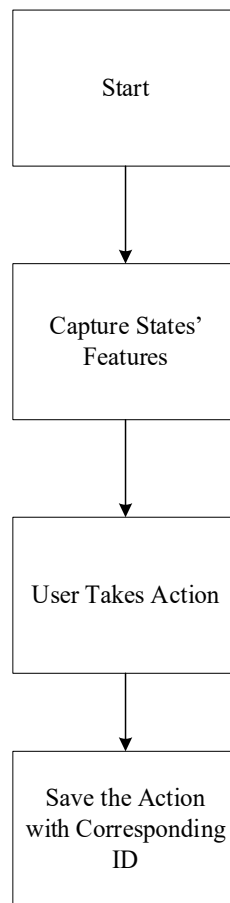
Therefore, a specific data was designed for the scope of this study.

A human demonstration was recorded, each sample had the environment and the action taken by the human actor recorded and linked using a unique ID.

The data recording was done using a third-party code designed specifically for this task and gave the human actor the ability to control the cart.

The states' features are stored with an ID every iteration, the same ID is used to index the action taken and save it in a table.

This data was later converted into a form that our model can accept.



**Figure 4.6:** Data Collection Flow Chart



#### 4.4 Supervised Learning

Supervised learning is the most popular branch of machine learning, supervised learning algorithms are designed to learn by training on previous examples.

In the scope of this study, supervised learning was used to train the policy network using the data collected from the human actor in an earlier stage.

The optimizer used in this stage is the same optimizer used in the first study case, as well as the hyper parameters, this will guarantee that the results will be transparent.

The loss function used is the cross-entropy loss function since we have only two actions.

For discrete probability distributions  $p$  and  $q$  equation is as shown below.

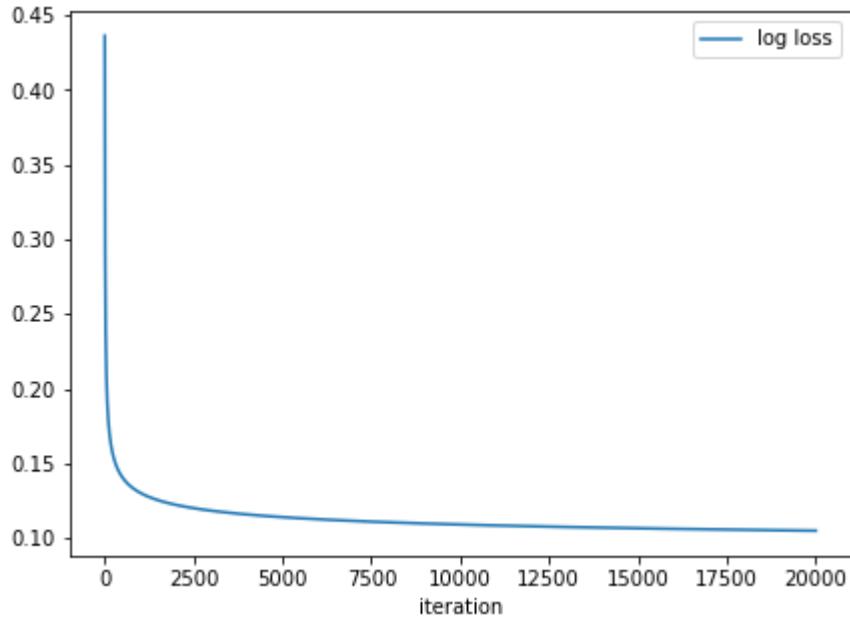
$$H(p, q) = - \sum_{x \in \mathcal{X}} p(x) \log q(x)$$

And for continuous distribution,  $p$  and  $q$  are assumed to be absolutely continuous with the respect to reference measure  $r$  (such as Lebesgue measure), the equation will be as shown below.

$$H(p, q) = - \int_{\mathcal{X}} P(x) \log Q(x) \partial r(x)$$

Where  $P$  and  $Q$  are probability density functions of  $p$  and  $q$  with respect to  $r$ .

Cross-entropy gives a loss based on how far the prediction result is far from the actual expected result.

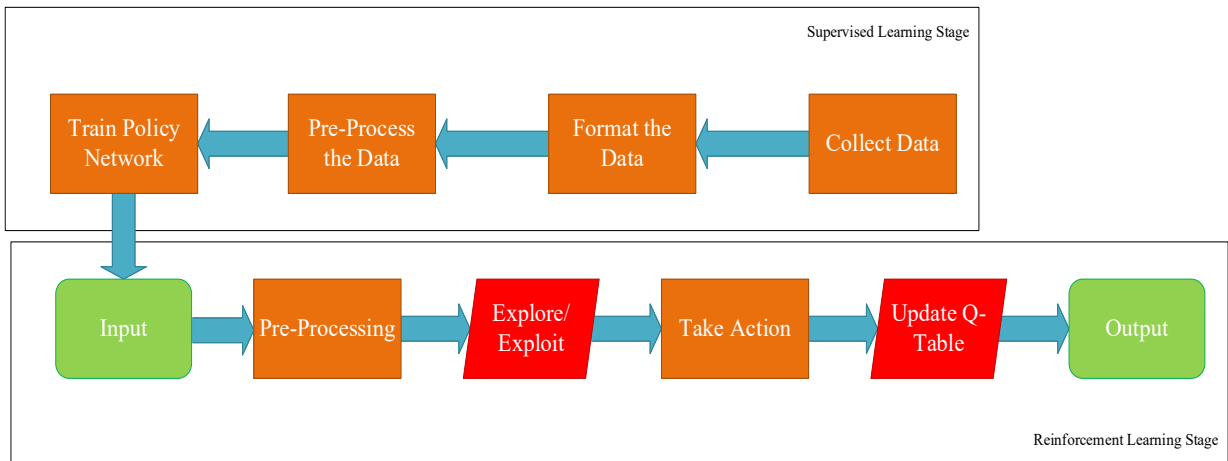


**Figure 4.7:** Cross-Entropy Gradient Descent

The value generate is between 0 and 1, the greater the error is the closer the value will be to 1.

After training the policy network, the same RL algorithm will be run using the trained network to be the policy network.

In this case, in stead of starting the policy network from a random weights values, the policy has already been trained to good values from the human actor’s demonstration.



**Figure 4.8:** Block Diagram of the Design Stage Including the Supervised Learning

## CHAPTER 5

### IMPLEMENTATION RESULTS

#### 5.1 Results

The main aim of this study is to compare the results between the reinforcement learning performance algorithm, and the performance of the same reinforcement learning algorithm after running its policy through supervised training using human demonstrated data.

Both study cases are running with the same hyper parameters, and attempting to solve the same problem.

The training dataset was collected specifically for the aim of this study and was formatted to be accepted by the model used.

The training and testing of the model were performed using PyTorch in a python programming language, and were run of a high-performance GPU unit.

```
DQN(  
  (fc1): Linear(in_features=10800, out_features=24, bias=True)  
  (fc2): Linear(in_features=24, out_features=32, bias=True)  
  (out): Linear(in_features=32, out_features=2, bias=True)  
)
```

**Figure 5.1** Model Summaries

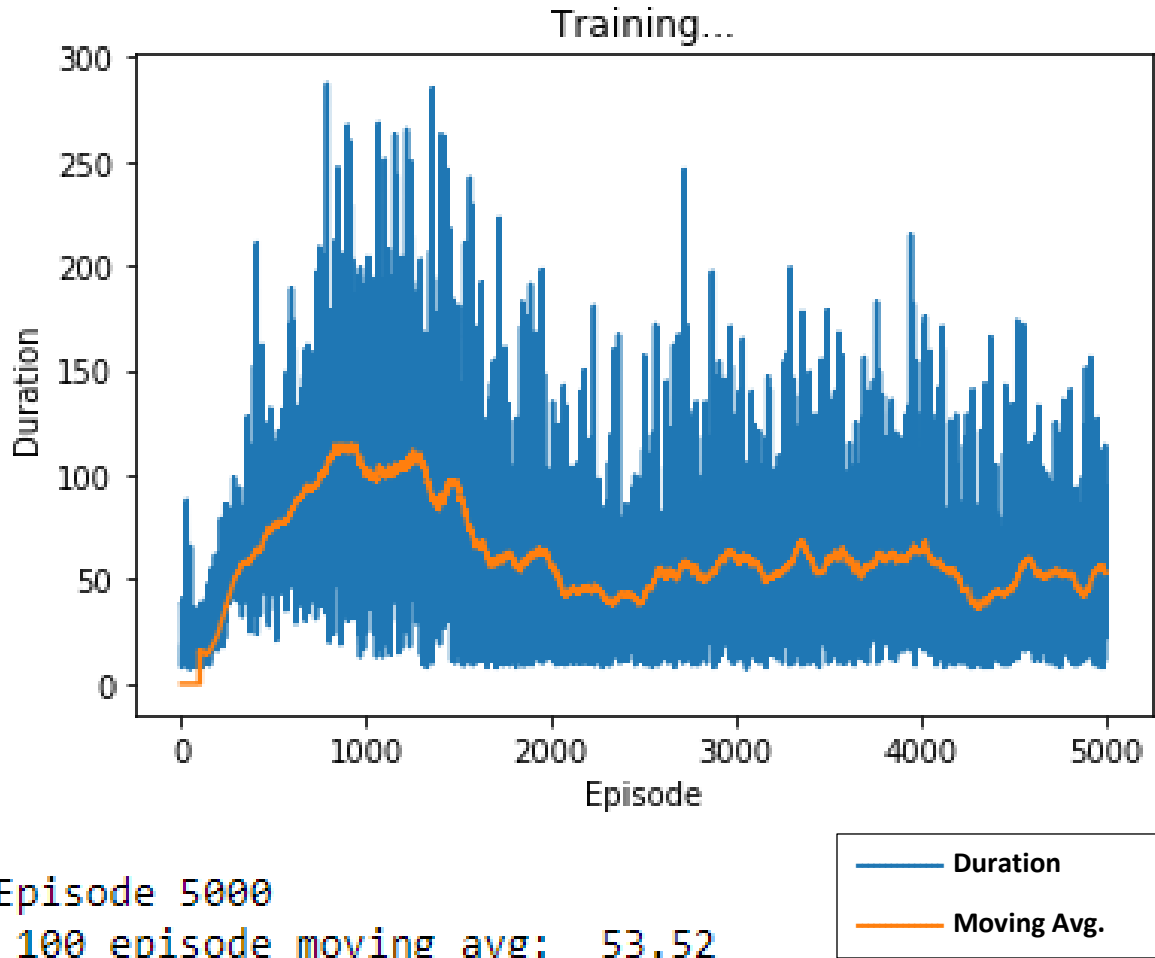
The table below shows the hyper parameters used in both study cases

	<b>RL</b>	<b>Supervised Learning + RL</b>
<b>Exploration Rate</b>	Random between (0-1) % 10	Random between (0-1) % 10
<b>RL Batch Size</b>	256	256
<b>Gamma</b>	0.865	0.865
<b>Epsilon Start</b>	1	1
<b>Epsilon End</b>	0.01	0.01
<b>Epsilon Decay</b>	0.001	0.001
<b>Memory Size</b>	100000	100000
<b>Learning Rate</b>	0.0004	0.0004
<b>Number of Episodes</b>	5000	5000
<b>Supervised Learning's Learning Rate</b>	–	0.0004
<b>Supervised Learning's Batch Size</b>	–	32
<b>Supervised Learning Epochs</b>	–	16
<b>Supervised Learning Data Size</b>	–	269

*Table 1 - Hyper Parameters' Values*

## 5.2 First Experiment (Reinforcement Learning)

The figure shown below displays the results of running the reinforcement learning algorithm directly without any sort of training, this means all weights of the policy network are randomly generated.



**Figure 5.2:** Reinforcement Learning Results

The blue line represents the duration in milliseconds that the agent was able to maintain the pole from falling in each episode.

The orange line the moving average of the last 100 episodes.

By considering both lines, it is clear that the agent started with a bad performance and it got better after around 840 episodes, the maximum duration was less than 300ms.

This performance lasted for only about 400 episodes and then it decayed back to the values it started from, and on the long run the model couldn't hold the pole for more than 150ms.

Since the main goal of using reinforcement learning is making the model adaptative and keep it learning while working, this goal was not achieved in this case.

By considering the moving average, it is easy to notice that the average of this duration reached the peak after about 1000 episodes of operating, and then started to fall.

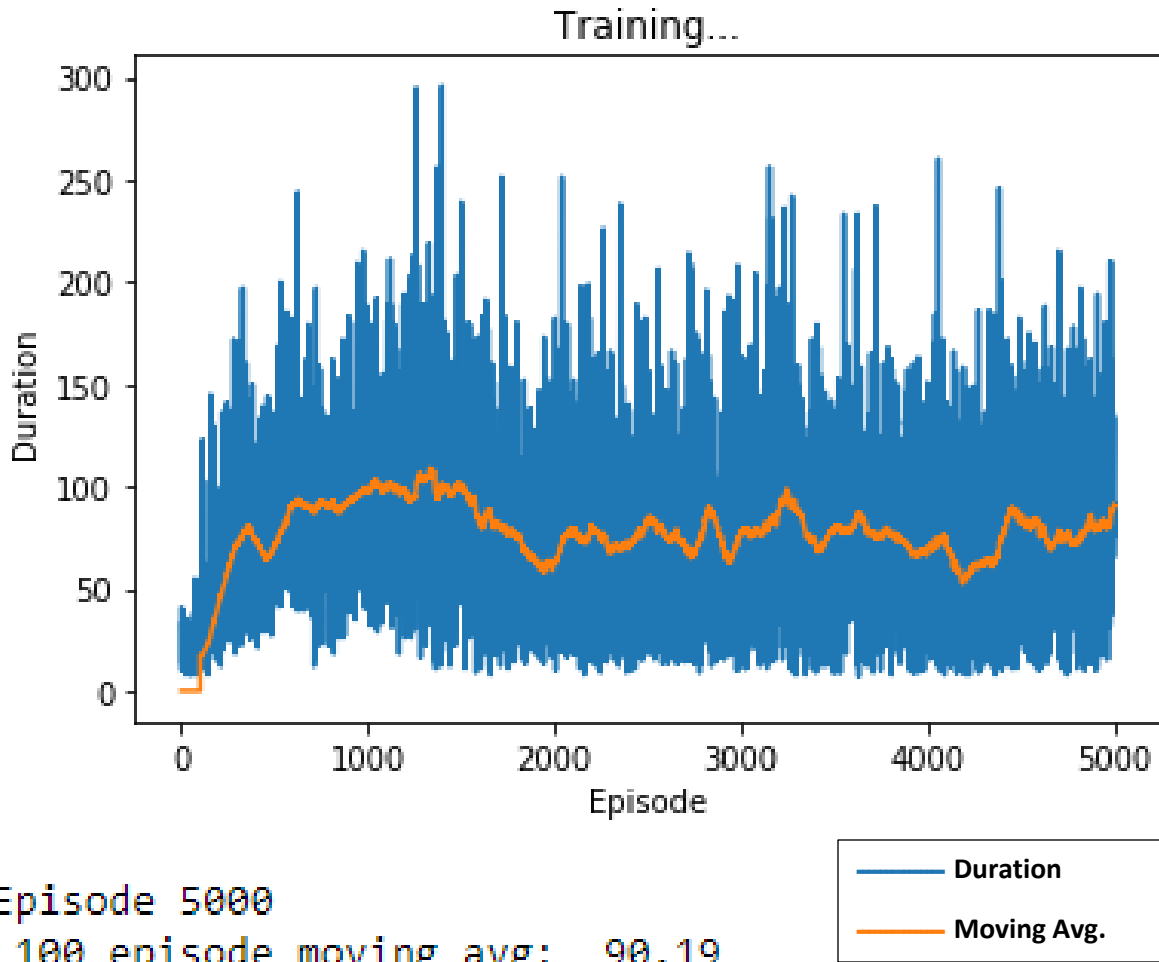
The agent ended the 5000 episodes with an average of 53.52ms.

Although the performance enhanced while going through more training episodes, but it's clear that the average didn't have a significant improvement.

The experiment ended with a falling moving average curve.

### **5.3 Second Experiment (Reinforcement Learning + Supervised Learning)**

The figure shown below displays the results of running the reinforcement learning algorithm after training the model to human demonstrated data, this means the weights of the policy network are being initialized based on good performance demonstration.



**Figure 5.3:** Reinforcement Learning + Supervised Learning Results

The blue line represents the duration in milliseconds that the agent was able to maintain the pole from falling in each episode.

The orange line the moving average of the last 100 episodes.

By considering both lines and comparing with the previous figure, it is clear that the agent had a better starting value and had better performance as through the rest of the training period.

In this case the agent took longer time to reach the same number (less than 300ms), and this is due to small training dataset, but on the other hand its performance enhanced significantly and

was able to keep the pole in upright position for 250ms in several episodes and in many other episodes it was a little less than that number.

The agent ended the 5000 episodes with an average of 90.19ms, which is a significant improvement from the previous experiment.

The experiment ended with a growing moving average curve, which gives us a hint that the performance will be better when running more episodes.



## **CHAPTER 6**

### **CONCLUSION**

As a conclusion, the study aimed to compare the difference in performance between reinforcement learning, and the reinforcement learning after training its model to a human demonstrated data using one of the machine learning training algorithms. By considering the results of both experiments, on the same problem with the same environment, conditions and parameters, it is clear that training the policy network helped improving the performance with more than 1.68 times. Since the dataset collected for this study was small, it is appropriate to say that with larger and more accurate, and professional data, the agent will have better results. The problem used in this study was a simple problem, used to test the concept and theory. Choosing a simple and small problem helps in generalizing the concept, and therefore, we can say that this study is valid for further and more complicated projects.

## REFERENCES

- [1] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. 2009. A *survey of robot learning from demonstration*. *Robotics and autonomous systems* 57, 5 (2009), 469–483.
- [2] H. Yu, X. Xu, H. Ma, Z. Zhu and C. Chen, "Control design of two-level quantum systems with reinforcement learning," 2018 33rd Youth Academic Annual Conference of Chinese Association of Automation (YAC), 2018, pp. 922-927, doi: 10.1109/YAC.2018.8406503.
- [3] Xiao-ting Cui and Xiang-dong Liu, "Fuzzy Neural Control of Satellite Attitude by TD Based Reinforcement Learning," 2006 6th World Congress on Intelligent Control and Automation, 2006, pp. 3983-3986, doi: 10.1109/WCICA.2006.1713120.
- [4] J. Kim, S. Cha, M. Ryu and M. Jo, "Pre-training Framework for Improving Learning Speed of Reinforcement Learning based Autonomous Vehicles," 2019 International Conference on Electronics, Information, and Communication (ICEIC), 2019, pp. 1-2, doi: 10.23919/ELINFOCOM.2019.8706441.
- [5] T. Blau, L. Ott and F. Ramos, "Improving Reinforcement Learning Pre-Training with Variational Dropout," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 4115-4122, doi: 10.1109/IROS.2018.8594341.
- [6] Gabriel V Cruz Jr, Yunshu Du, and Matthew E Taylor. *Pre-training neural net-works with human demonstrations for deep reinforcement learning*. arXiv preprint arXiv:1709.04083, 2017.
- [7] Recht, B. (2019). A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2(1), 253–279. <https://doi.org/10.1146/annurev-control-053018-023825>
- [8] Karanayil, B., & Rahman, M. F. (2018). Artificial neural network applications in power electronics and Electric drives. *Power Electronics Handbook*, 1245–1260. <https://doi.org/10.1016/b978-0-12-811407-0.00041-6>
- [9] Violante, A. (2019, July 1). *Simple reinforcement learning: Q-learning*. Medium. <https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56>.

- [10] Narendra, K. S., & Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1), 4–27. <https://doi.org/10.1109/72.80202>
- [11] Chen, X., & Hickey, C. (2018). Parallelized interactive machine learning on autonomous vehicles. *NAECON 2018 - IEEE National Aerospace and Electronics Conference*. <https://doi.org/10.1109/naecon.2018.8556776>
- [12] Zhang, S. (2019). Continuous control for robot based on deep reinforcement learning. <https://doi.org/10.32657/10356/90191>
- [13] Gupta, D. (2020, July 19). *Activation functions: Fundamentals of deep learning*. Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/>.
- [14] Glen, S. (2021, June 12). *Moving average: What it is and how to calculate it*. Statistics How To. <https://www.statisticshowto.com/probability-and-statistics/statistics-definitions/moving-average/>.
- [15] Jang, B., Kim, M., Harerimana, G., & Kim, J. W. (2019). Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, 7, 133653–133667. <https://doi.org/10.1109/access.2019.2941229>

## **APPENDICES**

## APPENDIX 1

### DATA COLLECTION CODE

```
#!/usr/bin/env python
import sys, gym, time
import pandas as pd
import keyboard
import pygame

e = gym.make('LunarLander-v2' if len(sys.argv)<2 else sys.argv[1])
f=False
ac=[]

if not hasattr(e.as, 'n'):
    raise Exception('Keyboard agent only supports discrete action spaces')

A = e.as.n
SC = 0
hAgentAc = 0
hRe = False
hP = False

def kp(k, m):
    global hAgentAc, hRe, hP
    if k==0xff0d: hRe = True
    if k==32: hP = not hP
    a = int( k - ord('0') )
    if a < 0 or a >= A: return
    hAgentAc = a
```

```

def kr(k, m):
    global hAgentAc, obs
    a = int( k - ord('0') )
    if a < 0 or a >= A: return
    if hAgentAc == a:
        hAgentAc = 0

def pr(k):
    global f
    f=True

def re(k):
    global f
    f=False

e.render()
e.unwrapped.viewer.window.on_kp = kp
e.unwrapped.viewer.window.on_kr = kr
i=-2
def ro(e):
    global hAgentAc, hRe, hP, i, f, ac
    hRe = False
    obser = e.reset()
    sk = 0
    tr = 0
    tTimeS = 0
    while 1:
        if i>=0:
            pygame.image.get_buffer_manager().get_color_buffer().save('.\screens\screenshot'+str(i)+'.png')

```

```

if keyboard.is_pressed('1'):
    ac.append(1)

    else:
        ac.append(0)
i=i+1
if not sk:
    #print("taking action {}".format(hAgentAc))
    a = hAgentAc
    tTimeS += 1
    sk = SC
else:
    sk -= 1

    obser, r, done, info = e.step(a)
# if r != 0:
    # print("reward %0.3f" % r)
tr += r
wOpen = e.render()
if wOpen==False: return False
if done: break
if hRe: break
while hP:
    e.render()
    time.sleep(0.1)
time.sleep(0.1)
print("timesteps %i reward %0.2f" % (tTimeS, tr))

```

```
print("ACTIONS={}".format(A))
print("Press keys 1 2 3 ... to take actions 1 2 3 ...")
print("No keys pressed is taking action 0")
while 1:
    wOpen = ro(e)
    if wOpen==False:
        data=pd.DataFrame(ac)
        data.to_csv('.\\train\\actions.csv')
        break
```



## APPENDIX 2 REINFORCEMENT LEARNING CODE

```
%matplotlib inline
import gym
import math
import random
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from collections import namedtuple
from itertools import count
from PIL import Image
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision.transforms as T
from torchvision import datasets, models
is_ipython = 'inline' in matplotlib.get_backend()
if is_ipython: from IPython import display
```

```

class qNet(nn.Module):
    def __init__(self, h, w):
        super().__init__()
        #the 3 is for color channels
        self.l1=nn.Linear(in_features=h*w*3, out_features=24)
        self.l2=nn.Linear(in_features=24, out_features=32)
        self.o=nn.Linear(in_features=32, out_features=2)
    def forward(self, temp):
        temp=temp.flatten(start_dim=1)
        temp=F.relu(self.l1(temp))
        temp=F.relu(self.l2(temp))
        temp=self.o(temp)
        return temp
exp = namedtuple('exp',
                 ('s', 'ac', 'next_state', 'reward'))
class reMem():

```

```

    def __init__(self, ca):
        self.ca=ca
        self.mem=[]
        self.count=0

```

```

def push(self,exp):

    if len(self.mem)<self.ca:
        self.mem.append(exp)

    else:
        self.mem[self.count%self.ca]=exp

    self.push+=1

def sample(self,bSize):
    return random.sample(self.mem,bSize)

def proveSample(self,bSize):
    return len(self.mem)>=bSize

class EGS():
    def __init__(self,s,e,d):
        self.s=s
        self.e=e
        self.d=d

    def exRate(self,cst):
        return self.e + (self.s-self.e) * math.exp(
            -1. * cst * self.d)

```

```

class ag():
    def __init__(self, st, nAc, device):
        self.cst=0
        self.st=st
        self.nAc=nAc
        self.device=device

    def sAc(self, s, pNet):
        r=st.exRate(self.cst)
        self.cst+=1

        if r>random.random()%10:
            ac = random.randrange(self.nAc)
            return torch.tensor([ac]).to(device) #explore
        else:
            with torch.no_grad():
                return pNet(s).argmax(dim=1).to(device) #exploit

rwA=np.array([])
rw=torch.from_numpy(rwA)
class cpMan():
    def __init__(self, device):
        self.device=device

        self.env=gym.make('CartPole-v0').unwrapped #unwrapped gets us an access
        self.env.reset()

        self.current_screen=None
        self.done=False

```

```

def res(self):
    self.env.reset()
    self.current_screen=None

def cl(self):
    self.env.close()

def render(self,mode='human'):
    return self.env.render(mode)

def nActAv(self):
    return self.env.action_space.n

def tAct(self,ac):
    _, rw, self.done, _=self.env.step(ac.item())
    return torch.tensor([rw],device=self.device)

def jS(self):
    return self.current_screen is None

def gSt(self):
    if self.jS() or self.done:
        self.current_screen=self.pScrn()
        bScrn=torch.zeros_like(self.current_screen)
        return bScrn

```

else:

```
s1=self.current_screen
s2=self.pScrn()
self.current_screen=s2
return s2-s1
```

def scrnH(self):

```
screen=self.pScrn()
return screen.shape[2]
```

def scrnW(self):

```
screen=self.pScrn()
return screen.shape[3]
```

def pScrn(self):

```
screen=self.render('rgb_array').transpose((2,0,1))
screen=self.scrnCrp(screen)
return self.transScrnDt(screen)
```

def scrnCrp(self,screen):

```
screen_height=screen.shape[1]
```

```
#strip off top and bottom
```

```
top=int(screen_height*0.4
```

```

bottom=int(screen_height*0.8)
    screen=screen[:,top:bottom,:]
    return screen

def transScrnDt(self,screen):
    screen = np.ascontiguousarray(screen,dtype=np.float32)/255
    screen=torch.from_numpy(screen)
    resize=T.Compose([
        T.ToPILImage(),
        T.Resize((40,90)),
        T.ToTensor()
    ])
return resize(screen).unsqueeze(0).to(self.device)

def plot (values,moving_avg_period):
    plt.figure(2)
    plt.clf()
    plt.title('Training...')
    plt.xlabel('Episode')
    plt.ylabel('Duration')
    plt.plot(values)
    moving_avg=get_moving_average(moving_avg_period,values)
    plt.plot(moving_avg)
    plt.pause(0.001)
    print("Episode", len(values), "\n", moving_avg_period, \
        "episode moving avg: ", \
        moving_avg[-1])
if is_ipython: display.clear_output(wait=True)

```

```

def mvAvg(period, v):
    v=torch.tensor(v, dtype=torch.float)
    if len(v)>=period:
        mvavg=v.unfold(dimension=0, size=period, step=1) \
            .mean(dim=1).flatten(start_dim=0)
        mvavg=torch.cat((torch.zeros(period-1), mvavg))
        return mvavg.numpy()
    else:
        mvavg=torch.zeros(len(v))
        return mvavg.numpy()

class QVs():
    device=torch.device("cuda" if torch.cuda.is_available() else "cpu")
    @staticmethod
    def cur(pNet, sts, acs):
        return pNet(sts).gather(dim=1, index=acs.unsqueeze(-1))
    @staticmethod
    def nxt(target_net, nxtSts):
        fStsLoc=nxtSts.flatten(start_dim=1) \
            .max(dim=1)[0].eq(0).type(torch.bool)
        nonFinStsLocs=(fStsLoc==False)
        nonFinSts=nxtSts[nonFinStsLocs]
        bSize=nxtSts.shape[0]
        v=torch.zeros(bSize).to(QVs.device)
        v[nonFinStsLocs]=target_net(nonFinSts) \
            .max(dim=1)[0].detach()
        return v

```



```

def extrTen(exps):
batch=exp(*zip(*exps))

ten1=torch.cat(batch.s)
ten2=torch.cat(batch.ac)
ten3=torch.cat(batch.rw)
ten4=torch.cat(batch.ns)

return (ten1,ten2,ten3,ten4)

bSize=256
gm=0.865
epsS=1
epsE=0.01
epsD=0.001
tUpdt=10
memSz=100000
lr=0.0004
epNum=5000

device=torch.device("cuda" if torch.cuda.is_available() else "cpu")
en=cpMan(device)
st=EGS(epsS, epsE,epsD)
agnt=ag(st, en.nActAv(), device)
mem=reMem(memSz)

pNet=qNet(en.scrnH(), en.scrnW()).to(device)
tNet=qNet(en.scrnH(), en.scrnW()).to(device)

```

```

tNet.load_state_dict(pNet.state_dict())
tNet.eval()
optimzr=optim.Adam(params=pNet.parameters(), lr=lr)

epDur=[]
for ep in range (epNum):
    en.res()
    s=en.gSt()
    for timestep in count():
        ac=agnt.sAc(s, pNet)
        rw=en.tAct(ac)
        ns=en.gSt()
        mem.push(exp(s, ac, ns, rw))
        s=ns

    if mem.proveSample(bSize):
        exps=mem.sample(bSize)
        sts, acs, rws, nxtSts=extrTen(exps)

        curQVs=QVs.cur(pNet, sts, acs)
        nxtQVs=QVs.next(tNet, nxtSts)
        tQVs=(nxtQVs*gm)+rws

        loss=F.mse_loss(curQVs, tQVs.unsqueeze(1))
        optimzr.zero_grad()
        loss.backward()
        optimzr.step()

```

```
if en.done:
    epDur.append(timestep)
    plot(epDur,100)
    break
if ep%tUpdt==0:
    tNet.load_state_dict(pNet.state_dict())

en.cl()
```

### APPENDIX 3

## REINFORCEMENT LEARNING + SUPERVISED LEARNING CODE

```
%matplotlib inline
import gym
import math
import random
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from collections import namedtuple
from itertools import count
from PIL import Image
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision.transforms as T
from torchvision import datasets, models
is_ipython = 'inline' in matplotlib.get_backend()
if is_ipython: from IPython import display
```

```

class qNet(nn.Module):
    def __init__(self, h, w):
        super().__init__()
        #the 3 is for color channels
        self.l1=nn.Linear(in_features=h*w*3, out_features=24)
        self.l2=nn.Linear(in_features=24, out_features=32)
        self.o=nn.Linear(in_features=32, out_features=2)

    def forward(self, temp):
        temp=temp.flatten(start_dim=1)
        temp=F.relu(self.l1(temp))
        temp=F.relu(self.l2(temp))
        temp=self.o(temp)
        return temp

exp = namedtuple('exp',
                ('s', 'ac', 'ns', 'rw'))

class reMem():

    def __init__(self, ca):
        self.ca=ca
        self.mem=[]
        self.count=0

```

```

def push(self,exp):

    if len(self.mem)<self.ca:
        self.mem.append(exp)

    else:
        self.mem[self.count%self.ca]=exp

    self.push+=1

def sample(self,bSize):
    return random.sample(self.mem,bSize)

def proveSample(self,bSize):
    return len(self.mem)>=bSize

lass EGS():
    def __init__(self,s,e,d):
        self.s=s
        self.e=e
        self.d=d

    def exRate(self,cst):
        return self.e + (self.s-self.e) * math.exp(
            -1. * cst * self.d)

```

```

class ag():
    def __init__(self, st, nAc, device):
        self.cst=0
        self.st=st
        self.nAc=nAc
        self.device=device

    def sAc(self, s, pNet):
        r=st.exRate(self.cst)
        self.cst+=1

        if r>random.random()%10:
            ac = random.randrange(self.nAc)
            return torch.tensor([ac]).to(device) #explore
        else:
            with torch.no_grad():
                return pNet(s).argmax(dim=1).to(device) #exploit

rwA=np.array([])
rw=torch.from_numpy(rwA)
class cpMan():
    def __init__(self, device):
        self.device=device
        self.env=gym.make('CartPole-v0').unwrapped #unwrapped gets us an access
        self.env.reset()
        self.current_screen=None
        self.done=False

```

```

def res(self):
    self.env.reset()
    self.current_screen=None

def cl(self):
    self.env.close()

def render(self,mode='human'):
    return self.env.render(mode)

def nActAv(self):
    return self.env.action_space.n

def tAct(self,ac):
    _, rw, self.done, _=self.env.step(ac.item())
    return torch.tensor([rw],device=self.device)

def jS(self):
    return self.current_screen is None

def gSt(self):
    if self.jS() or self.done:
        self.current_screen=self.pScrn()
        bScrn=torch.zeros_like(self.current_screen)
        return bScrn

```



else:

```
s1=self.current_screen  
s2=self.pScrn()  
self.current_screen=s2  
return s2-s1
```

def scrnH(self):

```
screen=self.pScrn()  
return screen.shape[2]
```

def scrnW(self):

```
screen=self.pScrn()  
return screen.shape[3]
```

def pScrn(self):

```
screen=self.render('rgb_array').transpose((2,0,1))  
screen=self.scrnCrp(screen)  
return self.transScrnDt(screen)
```

def scrnCrp(self,screen):

```
screen_height=screen.shape[1]
```

```
#strip off top and bottom
```

```
top=int(screen_height*0.4)
```

```

        bottom=int(screen_height*0.8)
screen=screen[:,top:bottom,:]
return screen

def transScrnDt(self,screen):
    screen = np.ascontiguousarray(screen,dtype=np.float32)/255
    screen=torch.from_numpy(screen)
    resize=T.Compose([
        T.ToPILImage(),
        T.Resize((40,90)),
        T.ToTensor()
    ])
return resize(screen).unsqueeze(0).to(self.device)

def plot (v,mvAvgP):
    plt.figure(2)
    plt.clf()
    plt.title('Training...')
    plt.xlabel('Episode')
    plt.ylabel('Duration')
    plt.plot(v)
    mvavg=mvAvg(mvAvgP,v)
    plt.plot(mvavg)
    plt.pause(0.001)
    print("Episode", len(v), "\n", mvAvgP, \
        "episode moving avg: ", \
        mvavg[-1])
    if is_ipython: display.clear_output(wait=True)

```

```

def mvAvg(period,v):
    v=torch.tensor(v,dtype=torch.float)
    if len(v)>=period:
        mvavg=v.unfold(dimension=0,size=period,step=1) \
            .mean(dim=1).flatten(start_dim=0)
        mvavg=torch.cat((torch.zeros(period-1),mvavg))
        return mvavg.numpy()
    else:
        mvavg=torch.zeros(len(v))
        return mvavg.numpy()

class QVs():
    device=torch.device("cuda" if torch.cuda.is_available() else "cpu")
    @staticmethod
    def cur(pNet, sts, acs):
        return pNet(sts).gather(dim=1, index=acs.unsqueeze(-1))
    @staticmethod
    def nxt(target_net, nxtSts):
fStsLoc=nxtSts.flatten(start_dim=1) \
    .max(dim=1)[0].eq(0).type(torch.bool)
    nonFinStsLocs=(fStsLoc==False)
    nonFinSts=nxtSts[nonFinStsLocs]
    bSize=nxtSts.shape[0]
    v=torch.zeros(bSize).to(QVs.device)
    v[nonFinStsLocs]=target_net(nonFinSts) \
        .max(dim=1)[0].detach()
    return v

```

```

def extrTen(exps):
    batch=exp(*zip(*exps))

    ten1=torch.cat(batch.s)
    ten2=torch.cat(batch.ac)
    ten3=torch.cat(batch.rw)
    ten4=torch.cat(batch.ns)

    return (ten1,ten2,ten3,ten4)

device=torch.device("cuda" if torch.cuda.is_available() else "cpu")

en=cpMan(device)
pNet=qNet(en.scrnH(), en.scrnW()).to(device)

data='./train/'

trn=datasets.ImageFolder(data, transform=T.Compose([T.Resize((40,90)),
    T.ToTensor()]))

trnLdr=torch.utils.data.DataLoader(trn,bSize=32)

b=next(iter(train_trnLdr))

def numCrt(prds, lbls):
    return prds.argmax(dim=1).eq(lbls).sum().item()

```

```

imgs,_=b
imgs=imgs.to(device)
import pandas as pd
memSz=100000
mem= reMem (memSz)
optimzr =optim.Adam(pNet.parameters(),lr=0.0004)
acsDir=pd.read_csv('./train/actions.csv')
ac=torch.tensor(acsDir.acs).to(device)
acsLdr=torch.utils.data.DataLoader(ac,bSize=32)
acB=next(iter(acsLdr))
for epo in range(16):
    tLs=0
    tCr=0 #to track the correct predections
    for b in trnLdr:
        imgs,_=b
        imgs=imgs.to(device)
        prds=pNet(imgs).to(device)
        ls=F.cross_entropy(prds,acB)
        optimzr.zero_grad()
        ls.backward()
        optimzr.step()
        tLs+=ls.item()
        nwd=torch.from_numpy(np.array([2]))
        tCr+=numCrt(prds,acB)
        acB=next(iter(acsLdr))
    print("ephoc:",epo,"total_correct:",tCr,"total_loss:",tLs)

```

```

bSize=256
gm=0.865
epsS=1
epsE=0.01
epsD=0.001
tUpdt=10
memSz=100000
lr=0.0004
epNum=5000

device=torch.device("cuda" if torch.cuda.is_available() else "cpu")
st=EGS(epsS, epsE,epsD)
agnt=ag(st, en.nActAv(), device)
mem=reMem(memSz)
tNet=qNet(en.scrnH(), en.scrnW()).to(device)
tNet.load_state_dict(pNet.state_dict())
tNet.eval()
optimzr=optim.Adam(pNet.parameters(), lr=lr)

epDur=[]
for ep in range (epNum):
    en.res()
    s=en.gSt()
    for timestep in count():
        ac=agnt.sAc(s, pNet)
        rw=en.tAct(ac)
        ns=en.gSt()
        mem.push(exp(s, ac, ns, rw))
    s=ns

```

```

if mem.proveSample(bSize):
    exps=mem.sample(bSize)
    sts, acs, rws, nxtSts=extrTen(exps)

    curQVs=QVs.cur(pNet, sts, acs)
    nxtQVs=QVs.nxt(tNet, nxtSts)
    tQVs=(nxtQVs*gm)+rws

    loss=F.mse_loss(curQVs, tQVs.unsqueeze(1))
    optimzr.zero_grad()
    loss.backward()
    optimzr.step()

if en.done:
    epDur.append(timestep)
    plot(epDur,100)
    break
if ep%tUpdt==0:
    tNet.load_state_dict(pNet.state_dict())

en.cl()

```

## APPENDIX 4. ETHICAL APPROVAL LETTER



### ETHICAL APPROVAL DOCUMENT


Date: 04/08/2021

To the Institute of Graduate Studies

For the thesis project entitled as “Enhancing Reinforcement Learning Behavior by Pre-Training the Model”, the researchers declare that they did not collect any data from human/animal or any other subjects. Therefore, this project does not need to go through the ethics committee evaluation.

Title: Assist. Prof. Dr

Name Surname: Elbrus Imanov


Signature: 

Role in the Research Project: Supervisor



## APPENDIX 5. SIMILARITY

Elbrus Imanov | User Info | Messages (1 new) | Instructor ▼ | English ▼ | Community | Help | Logout



Assignments | Students | Grade Book | Libraries | Calendar | Discussion | Preferences

NOW VIEWING: HOME > ALAAAL > ALL THESIS

















### About this page

This is your assignment inbox. To view a paper, select the paper's title. To view a Similarity Report, select the paper's Similarity Report icon in the similarity column. A ghosted icon indicates that the Similarity Report has not yet been generated.

### all thesis

INBOX | NOW VIEWING: NEW PAPERS ▼

Submit File Online Grading Report | Edit assignment settings | Email non-submitters

<input type="checkbox"/>	AUTHOR	TITLE	SIMILARITY	GRADE	RESPONSE	FILE	PAPER ID	DATE
<input type="checkbox"/>	Alaa Al Thefari	Abstract	0% 	--	--		1594471075	26-May-2021
<input type="checkbox"/>	Alaa Al Thefari	Chapter 1	0% 	--	--		1594469021	26-May-2021
<input type="checkbox"/>	Alaa Al Thefari	Chapter 3	0% 	--	--		1594469764	26-May-2021
<input type="checkbox"/>	Alaa Al Thefari	Chapter 5	0% 	--	--		1594470765	26-May-2021
<input type="checkbox"/>	Alaa Al Thefari	Conclusion	0% 	--	--		1594471386	26-May-2021
<input type="checkbox"/>	Alaa Al Thefari	Chapter 4	5% 	--	--		1594470143	26-May-2021
<input type="checkbox"/>	Alaa Al Thefari	all thesis	6% 	--	--		1627636671	04-Aug-2021
<input type="checkbox"/>	Alaa Al Thefari	Chapter 2	8% 	--	--		1594469318	26-May-2021

Enhancing Reinforcement Learning Behavior by Pre-Training the Model

Assist. Prof. Dr. Elbrus Imanov 