**Kung-Fu Master**

Nake S. Aparanji

**Introduction** **–**

Reinforcement Learning (RL) is a field in Machine Learning that deals with the training of an agent based on the rewards that it gets when interacted with the environment. So, the best way to polish our skills in RL is to actually train an agent to play a certain game, and we chose the ‘Kung-Fu Master’ from the Atari games. The main motive behind choosing ‘Kung-Fu Master’ is that it has a huge observation space and more than sufficient actions to perform, and hence making our tasks cut out when it comes to training the agent.

In this project, we have trained the ‘Kung-Fu Master’ agent twice using vanilla Deep Q-Network (DQN) algorithm – once using the Multi-layer Perceptron layers (MLP) and then using Convolutional Neural Networks (CNN). We then analysed the performance of both these variants.

Team members contribution –

Nake – DQN algorithm.

Surya – Neural network architecture.

Rakesh – Setting up the environment and rendering.

**Related Work –**

* Human-level control through Deep Reinforcement Learning – This research letter was published in 2015 by Macmillan Publishers. The writers of this paper developed a novel agent, called Deep Q-Network that was able to successfully learn policies directly from the high dimensional observation spaces. They created an algorithm that combined reinforcement learning with artificial neural network. Their agent was tested on various classic Atari 2600 games and its performance was better than any previous algorithms. One notable aspect of this project is the pre-processing part. As the observation space is huge (i.e., (210, 160, 3)), they have converted it to (84, 84) shape before using it to train. <https://www.nature.com/articles/nature14236#Abs3>
* Massively Parallel Methods for Deep Reinforcement Learning –

This research paper was also published in 2015 in Cornell University. The main aim of this paper is to use the concept of parallelism to better the performance of an agent. Their architecture uses 4 main components – parallel actors, parallel learners, distributed neural network and distributed store of experience. In short, multiple agents simultaneously interact with the environment, getting rewards and storing their experience. One of the key take away from this paper is that the training is not dependent on a single agent, that is, a single agent cannot be expected to successfully learn the environment in a given time. The experiences of multiple agents can be shared and each of them can learn in lesser time.

<https://arxiv.org/abs/1507.04296>

**RL Environment –**

Our environment, Kung-Fu Master, is part of the Atari games taken from the Gymnasium API. In this game, the Kung-Fu Master has to fight his way through the various kinds of enemies in an attempt to rescue Princess Victoria. The Kung-Fu Master gets 5 lives, he loses a certain amount of health every he is hit, finally losing a life. Also, there are 5 levels in the game – The Fist Fighter, The Boomerang Thrower, The Giant Kicker, The Lightning Magician and The Gang Master. The Kung-Fu Master gets points whenever he kicks or punches the enemy. These points get accumulated to finally give an end score. If the Kung-Fu Master is able to defeat the enemy under the allocated time then extra points are added to the player’s score. The same with player’s health as well, if health is then more points get added to the total score.

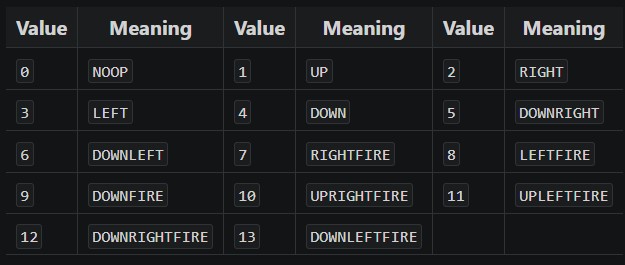
RL aspects –

* Observation Space – Gymnasium provides 2 kinds of observation space – one with RGB images and the other with Grayscale images. The shape of RGB matrix is (210, 160, 3) and that of Grayscale matrix is (210, 160). For our project, we will be using the grayscale version because of its simplicity (channels missing).

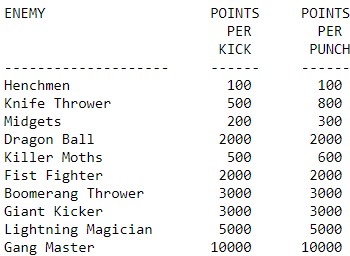




* Action Space – There are 14 discrete actions available for the Kung-Fu Master to take.



* Reward – The reward in this game is based on kind of enemy the Kung-Fu Master punches or kicks, stronger the enemy higher the reward.



We haven’t changed the environment or the reward function, other than using the ‘grayscale’ version of observation space instead of ‘rgb’ version.

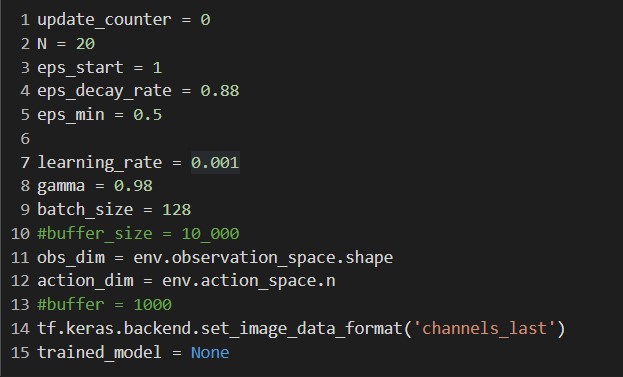
**Methodology –**

* Deep Q-Network (DQN) –

DQN is an extension of Q-Learning in which the q-values are evaluated using deep neural networks instead of Q-table. Here, the input is the observation space and it is passed through deep neural networks before calculating a set of output units. There will be 14 output units in the output layer as there are 14 actions. One of these 14 actions has to be chosen by the agent to take a step into the environment, this is done by calculating argmax of q-values. As the policy is updated based on the experiences of the agent, DQN comes under off-policy learning.

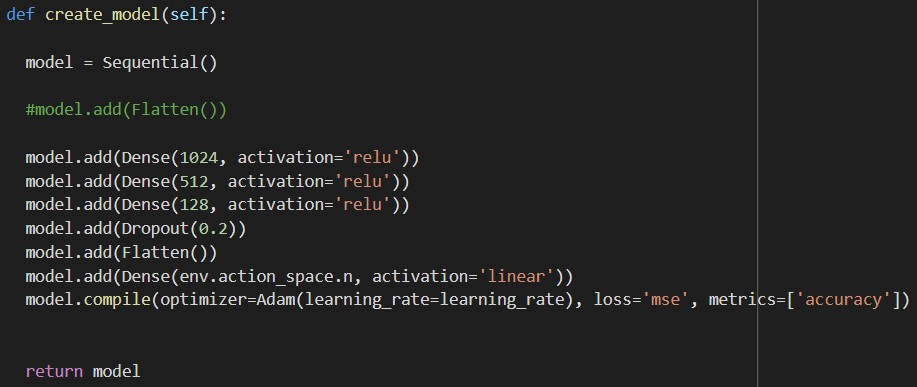
In our project, 2 models are used to train the agent – ‘dqn\_model’ and ’target\_model’; ‘target\_model’ is used to make predictions and the ‘dqn\_model’ is used to make the agent learn. After every 10 iterations the experience that the agent gains is stored in a replay memory using ‘deque’. For both MLP and CNN, we are using Adam optimizer, Mean Squared Error loss and Accuracy metric to compile the model.

The parameters that we used are:



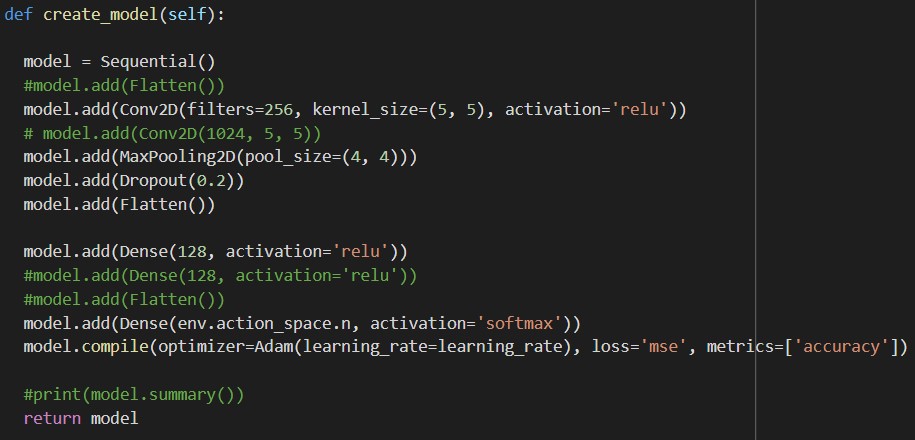
* DQN with MLP –

The neural networks used here are the basic Multi-layer Perceptron. A set of dense layers are used to construct a network that will help in finding a pattern/structure in the observation space. A dense layer is a fully connected layer where every unit is connected to every other unit in the subsequent layer. Dropout layer is used to reduce overfitting, it randomly removes a set units from the neural networks. The input received by the neural networks is extremely complex, hence we are using the ‘relu’ activation function. But at the output layer, the activation layer used is ‘linear’ as the complexity of the values is fairly simple. Flatten layer helps us in flattening the data just before it reaches the output layer.



* DQN with CNN –

Here, we are using the ‘convo2d’ layer on top of the neural network architecture. This layer helps us in applying convolution operation on the observation space to extract important patterns from the image. The ‘maxpooling2d’ layer down samples the observation space based on the kernel size.

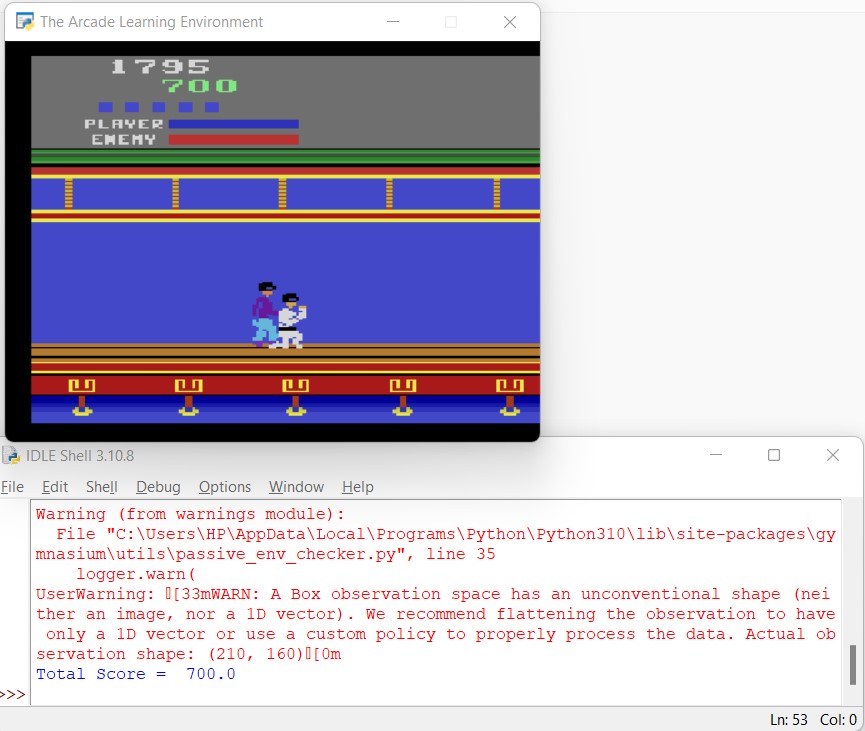


**Experiments and Results –**

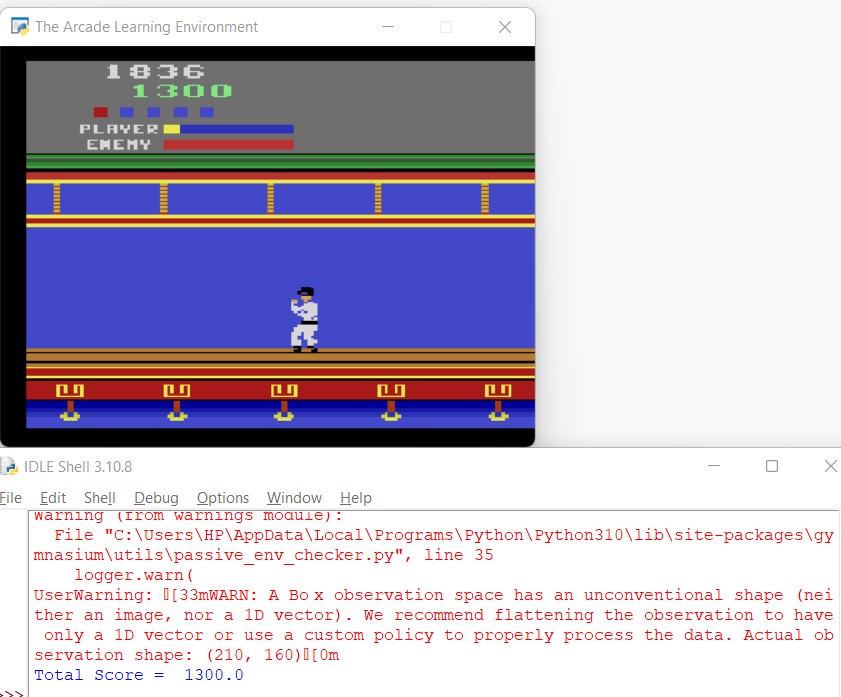
We trained the DQN with MLP agent for 500 episodes and the DQN with CNN for 300 episodes. Due to the limited computation power, we weren’t able to train the agent in one go; so we had to train it for around 4 episodes, then save the model, then load it again and continue to train the model. The trade off between exploration and exploitation was also a key aspect of this project. As the observation space is extremely complex, the agent is supposed to take random actions for an excessive period of time so that it gains much needed experience. But in our case, the agent was taking random actions for a very short number of episodes, and then without learning the agent was making wrong decisions. To overcome this we kept changing the ‘epsilon decay rate’ so as to keep a balance between exploration and exploitation.

The scores that we got for both MLP and CNN implementations is extremely less compared to the scores noted by other research papers. But, that is mainly due to the insufficient training of the agent (and that is due to limited computational power). But, we do have scores for MLP and CNN implementations and they are not too similar. Our experiments show that DQN with CNN performs exceedingly better that DQN with MLP. DQN with CNN gives us a score of 1300 while DQN with MLP gives us a score of 700.

DQN with MLP -



DQN with CNN -



**Conclusion and Future Works –**

To summarize, we have implemented DQN using 2 kinds of layers – MLP and CNN. From our results, we conclude that DQN with CNN performs exceedingly better than DQN with MLP. The reason behind this is the automatic pre-processing (max-pooling layer) of input that the CNN layers give us when compared to MLP layers.

We can extend our project to DQN with LSTM layers and then make an analysis of the performance of all these layers. Also, we could possible apply other variants of DQN including ‘NoisyDQN’ and ‘DDQN’.

**References –**

* <https://gymnasium.farama.org/environments/atari/kung_fu_master/>
* <https://www.endtoend.ai/envs/gym/atari/kung-fu-master/>
* <https://atariage.com/manual_html_page.php?SoftwareLabelID=268>
* <https://colab.research.google.com/drive/15SaoXnmxMp7XmyYMIpsWbhEaa6Ds77Li?usp=sharing#scrollTo=TjqycZ2yhpiX>
* Lab assignments.