İSTANBUL TECHNICAL UNIVERSITY
FACULTY OF COMPUTER AND INFORMATICS

PLAYING A VIDEO GAME BY DEEP LEARNING

Graduation Project Final Report

Emre Yeniay
150110013

Department: Computer Engineering
Division: Computer Engineering

Advisor: Dr. Damien Jade Duff

July 2018

Statement of Authenticity

I hereby declare that in this study

1. all the content influenced from external references are cited clearly and in detail,

2. and all the remaining sections, especially the theoretical studies and implemented software that constitute the fundamental essence of this study is originated by my individual authenticity.

İstanbul, July 2018

Emre Yeniay

PLAYING A VIDEO GAME BY DEEP LEARNING

(SUMMARY)

Today's technology is developing at a rapid pace. In the past, graphics cards were only used for graphics applications and video games. Nowadays, graphics processing units are also being used for parallel computation as they are designed to do the same thing many times. Big data has become more readily available. The traditional data processing approaches become inadequate on big data. All these progresses made deep learning to become very popular gradually.

The main purpose of this project is developing a software using deep learning methods which can play a video game. Video games are providing a natural environment suitable for artificial intelligence. By using artificial neural network inspired by neural network in brain, it is aimed to train software to learn. The platform which will be used on is Nintendo Entertainment System, also known shortly as NES. The famous in its time and still well-known game Super Mario Bros. is selected to be played as the main game of the project. This platform genre game will be played on personal computers using an emulator.

The intelligent agent that is developed takes a tile-based map of the environment as state information. It returns information about the selected movement. Moreover, Mario, the main character of the game, is given a prize or penalty by the environment by looking at whether he changed his position, died, or passed the episode. In this way the process is progressing and games are being played.

This learning method is called reinforced learning method. The software learns to play the game in this way. When designing the agent, the Q-learning algorithm, which is very popular in machine learning, was used as the learning algorithm. A network structure consisting of 1 input layer, 3 hidden layers and 1 output layer was generated. The incoming information comes to the input layer first in the neural network and is then processed in hidden layers. As a result, the potential prize information about the possible movements emerges from the output layer. Unless it is told to act randomly, the act with the highest potential reward is selected by the agent.

Various parameters were investigated and tested while the artificial neural network was developed. It is implied why these parameters were chosen in this paper. In design phase, three different models were designed to investigate the effect of batch size and learning phase timing on the system. The first of these models use the online learning method as it performs the learning process using a single experience in each frame. In the second model, the agent’s playing and learning phases are separated. The agent is putting experiences into a memory instead of learning at that moment. After the episode is finished, he gets a random sample with large batch size from the experience memory and trains the network that way. The last model has developed as a midpoint between the first two models. While the agent is playing, it is also learning at the same time, but this happens by taking a random sample with small batch size from experience memory. The differences and similarities of these models were given to the reader along with the pseudocodes.

Python is used as programming language in this project. As the primary main library, Keras is used to implement deep learning. Super Mario Gym was used as the environment. In addition, the TensorFlow which is required for Keras and OpenAI Gym which is required for Super Mario Gym was also utilized. The learning part in the software was accomplished using only the CPU. According to similar projects, there is a limited processing power and therefore a smaller number of processed frames.

After the analysis and design phases, the first goal was to find the best model for the current problem. Therefore, some tests were performed on the three developed models as the first detailed experiment, and the results were saved. We tried to find the most appropriate model among them. As the second experiment, it was aimed that the best model found in the first experiment would complete the first level of the game by increasing the number of learning. After this experiment the node weights of the taught model were saved in a file. In the last experiment, the model played a level of the game which it had never seen before. It was investigated that it might be as good as a model that makes random decisions. It was tested how well the model can generalize.

As a result of the first experiment, it is observed that keeping the previous experiences in a memory then training network with a large batch size after the game was over, is leading to a more efficient and good learning process. This model was chosen on this model for other experiments. In the second experiment, the system developed with this model managed to finish the first level of the game in a short time by the increase in number of played episodes. Later on, this model is tested on another Super Mario Bros. level which was never played by the program before. As a result of the third experiment, it was concluded that the trained model is much more successful than a randomly acting model, although it has never experienced this level before. The model was found capable of generalizing.

As the conclusion, it is shown how to learn to play a well-known video game in a learning process using deep learning. Comparisons were made utilizing different models. It has been proved that the deep learning can work using a system with limited hardware. This work can be considered as a guide for people who want to work or study on deep Q networks.

PLAYING A VIDEO GAME BY DEEP LEARNING

(ÖZET)

Günümüz teknolojisi ilerleyen bir hızla gelişmektedir. Eskiden ekran kartları sadece grafik uygulamaları ve oyunlar için kullanılıyordu. Günümüzde ise grafik işlemcileri aynı işlemi defalarca yapmak üzere tasarlandıkları için paralel hesaplama işlemleri için de kullanılmaya başlandılar. Büyük veri daha kolay ulaşılabilir hale geldi. Geleneksel veri işleme yöntemleri büyük veri üzerinde yetersiz kalmaya başladı. Tüm bu gelişmeler, derin öğrenmenin giderek popülerleşmesini sağladı.

Bu projenin temel amacı, derin öğrenme metotları kullanarak video oyunu oynayabilen bir yazılım geliştirmektir. Video oyunları, yapay zekâ için kendi doğaları gereği uygun bir ortam hazırlamaktadır. Beyindeki sinir ağına benzer bir yapıda olan yapay sinir ağları kullanılarak yazılıma öğrenme yetisi kazandırmak hedeflenmiştir. Üzerinde çalışılmış olan platform Nintendo Entertaiment System, kısaca bilinen adıyla NES'tir. Bu konsoldan Super Mario Bros. adlı zamanında sükse yapmış ve halen bilinen oyun, uygulamanın deneneceği ana oyun olarak seçilmiştir. Bir platform oyunu olan bu oyun, bir emulatör üzerinden modern kişisel bir bilgisayarda çalıştırılmıştır.

Geliştirilen akıllı ajan, durum bilgisi olarak ortamdan karo bazlı bir harita almaktadır. Ortama da seçtiği hareket ile ilgili bilgiyi geri vermektedir. Ayrıca, oyunun ana karakteri olan Mario'nun yer değişimine, ölüp ölmediğine veya bölümü geçip geçmediğine bakılarak ortam tarafından ajana ödül veya ceza verilmektedir. Bu şekilde süreç ilerlemekte ve oyun oynanmaktadır.

Bu öğrenme metoduna takviyeli öğrenme metodu denilmektedir. Yazılım, oyunu oynamayı bu yolla öğrenmektedir. Ajan tasarlanırken, öğrenme algoritması olarak makine öğrenmesinde oldukça popüler olan Q-learning algoritması kullanıldı. 1 giriş katmanı, 3 gizli katman ve 1 çıkış katmanından oluşan bir ağ yapısı oluşturuldu. Çevreden gelen bilgi, nöral ağ içindeki giriş katmanına gelmekte ardından gizli katmanlarda işlenmektedir. Sonuç olarak yapılabilecek hareketlerle ilgili potansiyel ödül bilgisi sonuç katmanından çıkmaktadır. Rasgele hareket etmesi söylenmedikçe, en yüksek potansiyel ödüle sahip hareket ajan tarafından seçilmektedir.

Yapay nöral ağ geliştirilirken çeşitli parametreler araştırıldı ve denendi. Bu parametrelerin neden seçildiği hakkında bahsedildi. Dizayn aşamasında öğrenim işleminin yapıldığı zaman ve yığın miktarının öğrenmeye etkisini araştırmak için üç farklı model tasarlandı. Bu modellerden ilki çevrimiçi öğrenme metodu ile her karede tek bir tecrübeyi kullanarak öğrenme işlemini gerçekleştirmektedir. İkinci modelde ajanın oyunu oynama ve öğrenme fazları birbirinden ayrılmıştır. Ajan elde ettiği deneyimleri o anda öğrenmek yerine hafızaya atmaktadır. Bölüm bittikten sonra bu hafızadan rastgele aldığı büyük bir yığın ile öğrenmesini gerçekleştirmektedir. Son modelde ise ilk iki model arasında orta yol bir yol geliştirilmiştir. Ajan oynarken aynı anda öğrenme işlemi de yapmakta ama bu hafızadan aldığı rastgele küçük yığınlar şeklinde olmaktadır. Bu modellerin farkları ve benzerlikleri, sözde kodları ile birlikte okuyucuya verildi.

Programlama dili olarak Python kullanılmaktadır. Başlıca kütüphane olarak, derin öğrenmeyi uygulamak için Keras'tan yararlanılmaktadır. Ortam olarak Super Mario Gym kullanılmıştır. Bunun yanında Keras için gerekli olan Tensorflow ve Super Mario Gym için gerekli olan OpenAI Gym'den de faydalanılmıştır. Yazılımda öğrenme işlemleri sadece yerel işlemci kullanılarak yapılmıştır. Benzer projelere göre kısıtlı bir işlem gücü ve bu yüzden az sayıda işlenen kare bulunmaktadır.

Analiz ve tasarım aşamalarından sonra ilk hedef, mevcut problem için en iyi çözümü öneren modeli bulmaktı. Bu yüzden ilk detaylı deney olarak geliştirilen üç model üzerinde bazı testler yapıldı ve sonuçlar not edildi. Aralarında bu proje için en uygun model bulunmaya çalışıldı. İkinci deney olarak bulunan en iyi modelin, öğrenme sayısı artırılarak ilk bölümün bitirilmesi hedeflendi. Bu deneyden sonra öğretilen modelin düğüm ağırlıkları bir dosyaya kaydedildi. Son deneyde ise model daha önce hiç görmediği bir bölümü oynadı. Rastgele kararlar veren bir modele göre kadar iyi olabileceği araştırıldı. Modelin ne kadar iyi genelleştirme yapabildiği test edildi.

Yapılan ilk deney sonucunda önceki deneyimleri bir hafızada tutan ve oyun bittikten sonra yüksek bir yığın miktarı ile öğrenme işlemini daha verimli ve iyi bir öğrenme gerçekleştiği gözlemlendi. Diğer deneyler için bu model üzerinde karar kılındı. İkinci deneyde ise, oynanan bölüm sayısı artırılarak bu model ile geliştiren sistem kısa sürede oyunun ilk bölümünü bitirmeyi başarmıştır. Daha sonra bu model daha önce hiç program tarafından oynanmamış bir başka Super Mario Bros. bölümü üzerinde denendi. Üçüncü deneyin sonucu olarak, öğretilen modelin daha önce hiç deneyimlememesine rağmen rastgele hareket eden bir modele göre çok daha başarılı olduğu kanısına varıldı. Modelin genelleştirme yetisi olduğu bulundu.

Sonuç olarak, derin öğrenme kullanılarak yapılan öğrenme işleminin iyi bilinen bir video oyunu oynamayı nasıl öğrenebileceği gösterildi. Değişik modeller üzerinden karşılaştırmalı testler yapıldı. Derin öğrenmenin kısıtlı donanıma sahip bir sistem ile bile çalıştırılabileceği kanıtlandı. Bu çalışma, derin Q ağlarında çalışmak veya öğrenmek isteyenler için bir rehber niteliğinde oldu

Contents

1 Introduction and Problem Summary 1
2 Comparative Literature Survey 3
3 Developed Approach and System Model 5
3.1 Environment Model 5
3.1.1 Nintendo Entertainment System 5
3.1.3 Super Mario Bros. 5
3.1.3 State Data 6
3.1.4 Reward Function 7
3.1.5 Action Space 8
3.2 Agent Model 9
3.2.1 Q-Learning 9
3.2.2 Network Structure 10
3.2.3 Hyperparameters 13
3.2.4 Training Models 14
3.2.4.1 Model A 14
3.2.4.2 Model B 15
3.2.4.3 Model C 16
4 Experimentation Environment and Experiment Design 18
4.1 System Environment 18
4.1.1 Software Specifications 18
4.1.2 Hardware Specifications 19
4.2 Experiments 19
4.2.1 Model Test 19
4.2.2 Training Test 20
4.2.3 Generalization Test 20
5 Comparative Evaluation and Discussion 22
5.1 Experiment Results 22
5.1.1 Model Test Results 22
5.1.2 Training Test Results 25
5.1.3 Generalization Test Results 26
5.2 Discussion 26
6 Conclusion and Future Work 28
7 References 29

1 Introduction and Problem Summary

The possibility of artificial beings which can think, learn and behave as humans were always a debate for centuries. Although it was a debate in history, now it is a popular and huge topic in computer science. It also widely found a part in sci-fi novels or movies, which show us there is an expectation that beings with artificial intelligence will be seen common in our life in future.

Artificial intelligence term was invented as a phrase firstly in 1956 by a computer scientist named John McCarthy according to Smith [1]. It was also the first time an academic conference came true on such subject. The main discussion topic was an artificial being that could reason and solve problems as a human.

Just after three years, Arthur Samuel defined machine learning as “the ability to learn without being explicitly programmed.” according to Munoz [2]. This way, machine learning became a subset of artificial intelligence but without any complex decision trees. It works by sending plenty amount of data to algorithm, and then waiting for algorithm to adjust itself, learn and improve.

One of the approaches to machine learning is deep learning. It is inspired by structure and function of the human brain and its neural network. There are neuron like nodes and layers in system which connect and solve the problem by sending message to each other. Big data availability and improving technology made deep learning a popular subject in computer science nowadays. Although it cannot be considered as a brand-new concept, it started to become more feasible due to the availability of high computational power. Some researches from Nvidia which is a graphics processing unit company, achieved to develop self-driving cars on a road without explicit labels using a deep convolutional neural network [3].

The main objective of this project is developing a software using deep learning which works on personal computers and can learn how to play a video game itself. The software will achieve that objective using reinforcement learning methodology. In reinforcement learning, the system consists of an intelligent agent and an environment. Intelligent agent or mostly known as agent is the smart part of the system which makes decisions. It will make choices and act depending what it observes on the environment. Meanwhile, the environment will progress based on that action and give feedback information about the new state of the environment. There will be also a reward or punishment given to agent by environment. The agent will try to determine whether it caused a desired or undesired situation using this reward information. Using this information, it is aimed that the agent is able to learn a policy to win the game.

The video game is not developed from scratch in this project. The purpose was playing an existing video game made for humans instead of an artificial game developed for machine learning projects. That setting made the project more amusing and interesting. However, modern games with 3D graphics are hard to analyze, learn and play even for a highly developed artificial intelligence. That is why choosing a video game from an old gaming console instead of modern game consoles or computers made more sense for this project.

NES (Nintendo Entrainment System) is chosen as the gaming console to pick a game from. It has weaker hardware, limited input and output options and easier games compared to modern computers or game consoles. Moreover, the games played on this platform is still remembered by people even in today. However, since a personal computer will be used to train a game from NES, a method was required to run such game on modern personal computers. By virtue of a video game emulator, playing a NES game on computers which we use today became possible. They emulate a video game console's hardware and play its games on the real platform it is working on.

The selected game on NES console to train agent is a well-known 2D platformer named Super Mario Bros. It has a main character named Mario who needs to rescue the princess by passing levels. However, it is not an easy task since there are lots of enemies and obstacles in his way. The agent is expected to pass the first level of Super Mario Bros at least once without using any checkpoint.

In the neural network, input layer will be fed by inputs taken by environment program Gym Super Mario. It uses OpenAI Gym as its backend environment toolkit and generates a 16x13 sized tile-based state depending on the elements displayed in screen. The squares in tiles can have different values based on its element such as obstacles, Mario or enemies.

The agent will be developed using Keras as its primary neural network API. The agent created by Keras will process the state as the input in neural network. It will be processed through hidden layers. At the end, the action will be determined at the output layer of neural network. The agent will send the outcome to the environment and earn a reward. That way agent will try to maximize future outcomes.

The primary focused learning method will be Q-Learning, a reinforcement learning technique. In this method, the agent gets state information about the environment and choose action which has the greatest expected future reward. After getting reward and the next state, the agent learns an experience.

2 Comparative Literature Survey

History of Computing class papers in University of Washington has beneficial information about the artificial intelligence concept and its history [1]. It also gives information about the famous AI research goal named Turing test. Evaluators try to determine which participant is a computer and which participant is a human by observing their responses. It could be implied to my project as an experiment at the end of the project. However, behavior of the agent can be easily distinguishing from how a human plays the game in my project. Trying to mimic human behavior instead of getting the best result while playing a video game can be topic of another project or research.

The most comprehensive book available about deep learning is published by MIT Press [4]. The online version of the book is available in a website and can be reached for free. The book starts giving information about linear algebra and machine learning at first as basics. Then, it gives detailed information about deep learning and its applications. Moreover, the website has lectures and exercises available for people who want to improve themselves and get better in deep learning. The knowledge gathered by reading this book is simply unmatched.

Probably, one of the oldest relevant resource founded about the topic of the project is "Temporal Difference Learning and TD-Gammon" published by Tesauro in 1995 [5]. Tesauro developed a backgammon learning program which learnt how to play the game entirely by temporal difference method which is a reinforcement learning technique. Although it is not used much nowadays, temporal difference can be seen as a crucial step and aided other people to develop today's popular learning algorithms such as Q-learning and SARSA.

The project which inspired me to select this video game as the environment is shared with people in an article named "Playing Atari with Deep Reinforcement Learning" [6]. The paper is published by a company named DeepMind Technologies which is acquired by Google in 2014. It has the most relevant resources and studies for my topic. This paper made a big impression using a deep Q-learning neural network to train the agent to play video games. Their agent is trained by a variant of Q-learning algorithm and applied to 7 different Atari 2600 games without changing the network architecture or hyperparameters. Six out of seven games gave admirable results at the end of their research. Compared to my project, the video games they used to train the agent were from an older video game console and simpler than the video game I used. Furthermore, they were able to train the agent for much more frames than I trained. However, they used raw pixel data to train agent; meanwhile, I used the tile-based state information that environment returns.

The other article I found related to my topic is named "A Comparison of Learning Algorithms on the Arcade Learning Environment" published by Defazio and Graepel [7]. They worked on 55 different arcade games and used 6 different learning algorithms in their project. The paper includes empirical results between learning algorithms. It proves that many arcade games can be played by a well-designed A.I. It also mentions about challenges in an arcade learning environment compared to simple reinforcement problems.

As for practical usage and implementation part, the book named "Deep Learning with Keras" is found just perfect for this project [8]. Example python codes and visual figures

are what makes this book favorable over other sources that can be found on the Internet. It has various examples including developing a game A.I. with Keras using deep learning. Since the Keras is the main API used for this project to develop neural network, this book helped me a lot in my project.

Although there are increasing number of articles, studies and researches published about deep learning; I have discovered that not all of them agree with each other in results. For example, Pandey and Dukkipati declared in their article that neural network with a single but very wide layer outperforms deep architectures with limited width layers [9]. They discuss about benefits of this approach such as reduced number of hyperparameters and decreased computational time complexity. On the other hand, Safran and Shamir suggested that deeper neural networks perform better than wider ones in their more recent article [10]. They concluded depth of a neural network played a more important role compared to its width in network's performance.

There seem to be two main reasons for such conflicts to appear between researches. The first of all, deep learning is an area which continues to be developing. Although there are recent developments and refinements in this area, it is still one of the most complex research areas in artificial intelligence. Researchers still discover something new using different methods and it seems there are still much more waiting to be found in this area.

The second reason is there is no magic bullet in deep learning. There seems to be no a well-known universal solution or a rule of thumb which can help to solve any problems people encountered. There are huge numbers of hyperparameters which needs to be tweaked by the user and solution may differ based on many different factors. All these factors make solution space to be more specific to problem base and makes results of deep learning difficult to compare with other studies.

3 Developed Approach and System Model

3.1 Environment Model

3.1.1 Nintendo Entertainment System

Training a neural network for modern 3D games is a huge challenge even for the most advanced deep neural networks. The game console which will be used to select a game from is chosen as Nintendo Entertainment System or NES for short. NES is released at 1983 for the first time in Japan by Nintendo and belongs to third-generation video game consoles. It is a video game console just like Atari 2600 used by Deepmind [6]. However, unlike NES, Atari 2600 belongs to the second-generation video game consoles instead of third-generation video game consoles. NES has more complex video games by both graphics and gameplay wise. Moreover, its controller has more buttons and input options on it. For example; breakout game can be played using only left and right directional inputs.

Playing a NES game on today's personal computers become possible using video game emulators. One of those emulators named FCEUX is used in this project. The application has a small file size and can work on Ubuntu. It supports Lua codes to make changes or get information from the emulator and already comes with Gym Super Mario package as ready to be used.

The games on NES normally runs at fixed 60 frames per second; therefore, 60 state data are returned to the agent per second. However, this approach caused a very slow playing time especially in online training models. First of all, fps value is decreased to 15 fps using frame skipping by modifying the Lua file. Second, emulator speed is selected as "nothrottle" instead of "normal" to make emulator run as fast as the system hardware allows. Those changes allowed emulator to play the game in an increased speed while there is no unwanted frame skipping and state information were still sufficient for the agent to learn. The change from 60 fps to 15 fps actually lead to gathering more stable training results.

The most of games that can be played on NES are 2D platform games. On those games, player sees the main character from its side, and controls it to jump between numerous platforms while avoid obstacles and enemies. One of these games, Super Mario Bros, became very popular all around the world with good gameplay and level design features.

3.1.3 Super Mario Bros.

Super Mario Bros. is probably the most well-known video game of NES. It is the first game ever made in Super Mario series. The main character of the game which is an Italian plumber named Mario, turned into the one of the most recognizable mascots of the video game company Nintendo. The game is more complex than other games used in deep learning such as Breakout or Pong which belongs to previous generation consoles.

There are static obstacles such as pipes or walls. There are also pits which Mario can fall into and die immediately. Mario has to jump over those obstacles and pits if he wants to progress. There are enemies such as mushroom-like creatures named Goomba and turtles named Koopa. If Mario let them to touch himself from his left or right side, they will either kill Mario or turn him into small and weak version of him. To deal with enemies, Mario can stomp on them or just ignore and jump over them.

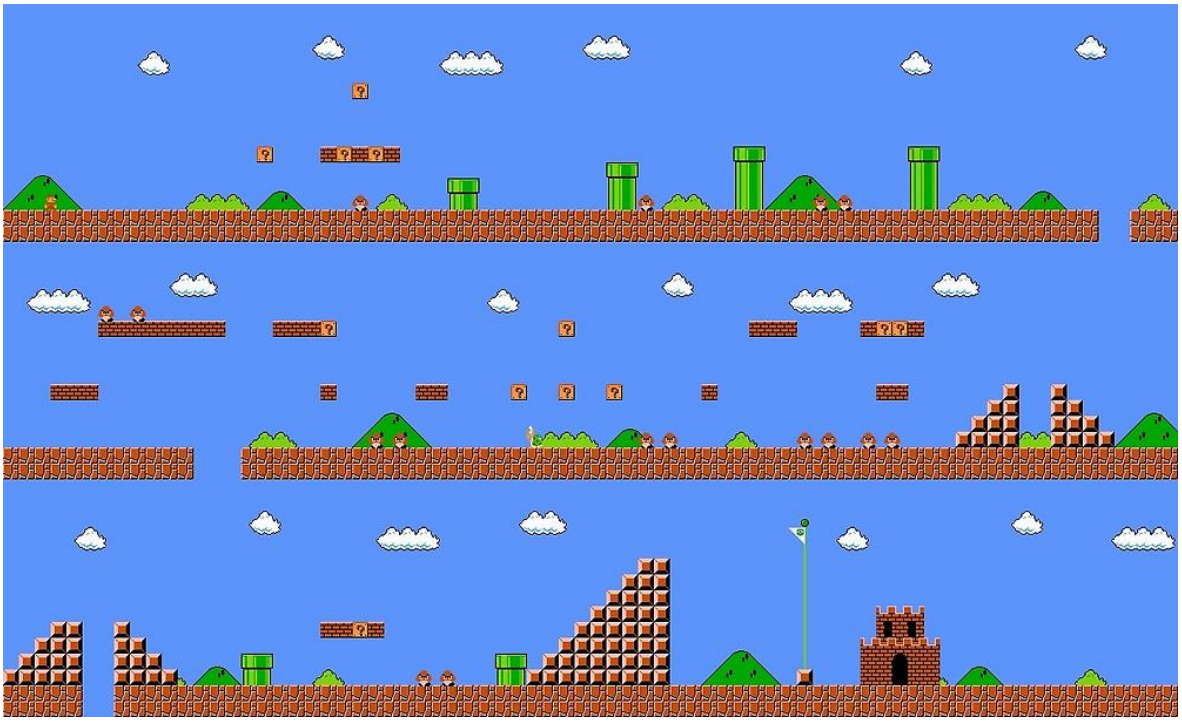

Figure 3.1: The map of World 1-1 of Super Mario Bros [11]

The train process will be on famous World 1-1 level of Super Mario Bros which can be seen in [11, Fig. 3.1]. As the most of other levels in the game, Mario has to move in a right direction to reach the end of the level while getting over pits, obstacles and enemies. It is the first level in the game and can be considered easier compared to other levels in the game. It is made for to be a tutorial for new players. Those properties make it a great level to train the agent.

There is a checkpoint location in the middle of the level. When Mario achieved to get there, game will respawn him on that point instead of the first starting point when he dies. This feature of the game is disabled to get stable results and to make finishing the first level more difficult for agent. Mario always starts over from the first starting point after he dies even he manages to pass the checkpoint.

3.1.3 State Data

The system model involves of an environment and an agent when using reinforcement learning paradigm. The environment in reinforcement learning methods returns two values to agent after agent makes an action; state and reward. In my project, this procedure keeps repeating in a loop in playing phase where the agent takes actions to play the game. It can be seen visually to get a better understanding in [12, Fig. 3.2].

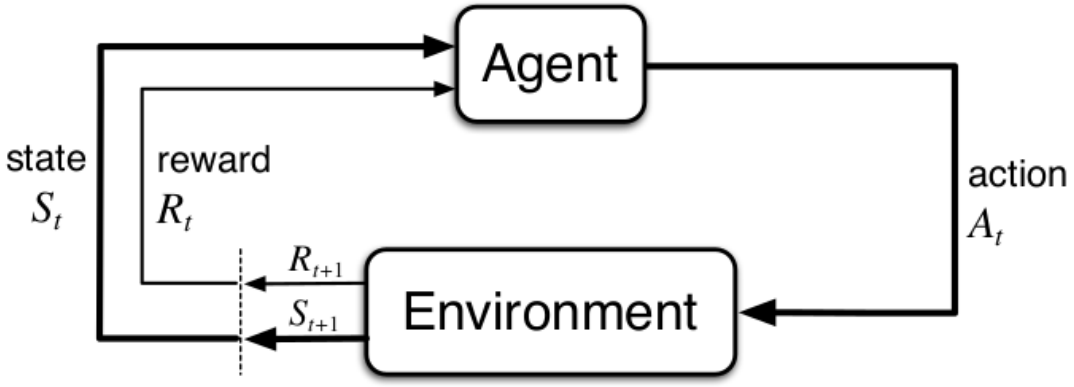

Figure 3.2: Relationship between agent and environment in reinforcement learning [12]

The first data of what an agent gets after taking an action is state. It is produced by the environment to give information to the agent about situation of the environment. The agent uses this information in two procedures. The first procedure can be called as playing phase. It is when the agent has to decide what action it should do based on the current state.

The other procedure can be called as learning phase. In the learning phase, the agent uses state, action, reward and next state information to change the weights of nodes in neural network. By manipulating nodes, the agent will make a better decision next time.

Gym Super Mario returns state data to the agent as a tile-based map. It is actually a two-dimensional integer matrix with a size of 16x13. Gym takes the categorical data from FCEUX, turns data to integers with integer encoding and returns the final matrix to the agent. Each integer in matrix can have 4 different values depending on what is inside of related tile. If there is nothing in that tile, integer value will be 0; if it is an object, the value will be 1; if it is an enemy, the value will be 2 and if it is Mario, the value will be 3.

Integer encoding told above is not making much sense for the agent. There is a better way to represent the state data that can make machine learning algorithms to do a better job in prediction. A binarization technique named one hot encoding is used to solve that problem. Using this method, categorical data converted into binary data. For example, if there is enemy in a tile, the value of that tile will be an array of 0010 instead of integer 3. The disadvantage of that approach is increased size of data. Now, the input layer of the neural network must be fed with 16 x 13 x 4 = 832 different inputs instead of 16 x 13 = 203 different values. At the last stage, those final 832 binary integers turned into an array to feed input layer of the neural network.

3.1.4 Reward Function

The other thing what environment sends to agent after an action is reward. Rewards are used in reinforcement learnings to make agent learn whether such behavior caused a pleasant situation such as completing a level. There can also be negative rewards to punish the agent if such action caused the environment to progress to an unpleasant state. Unlike state data, reward data is used by the agent only at training phase, not at playing phase.

Given rewards or punishments affects the action which will be taken later at the same state if at least one training phase is completed.

Rewards are given to agent with some different ways. The most frequently given reward is based on position change of Mario. If Mario moves to right direction and increases the score point, a positive reward is given to agent meanwhile if Mario moves to left direction, a negative reward is given. The amount of the reward is based on change of position of Mario. That means moving quickly by pressing sprinting button in the game will cause either a reward or a punishment with greater absolute value. The value of reward usually changes between -10 and 10 points depending on how fast Mario moved. However, a bit of punishment is added to the reward function, if Mario is standing still. Although, it should get 0 points normally since Mario is not moving at all, I found that giving -0.5 points as reward in that situation helps preventing Mario stuck in a place for the next time.

There are also rewards and punishments given to agent when something important happens on related state. When Mario dies by an enemy or falling into a pit, reward is given as -50 points as it is an undesired situation. However, if Mario dies due to running out of time, there is no punishment given. It is designed it in that way; because, otherwise the given punishment would not be related to state at that time. When the agent successfully completes a level, it will take 100 points as reward.

3.1.5 Action Space

The controller (gamepad) of NES console has total of 8 buttons on itself which can be seen in [13, Fig. 3.3]. It has a thumb-operated four-way directional control on its left side. On the right side, it has two action buttons to shoot, sprint or jump characters in games. Those action buttons are named B and A. On the middle side, it has start and select buttons which does not interfere the gameplay but works as pause or selection buttons. Since start and select buttons does not interfere the gameplay, we have 6 meaningful inputs which are up, right, down, left, A and B buttons. To take an action, the player must press at least one of these buttons, although the player also can press more than one buttons at the same time.

Figure 3.3: A Picture of a NES controller [13]

NES controller does not have analog buttons or sticks like in modern console controllers. The inputs given by pressing buttons on NES controller are pretty much like binary values and must be either true or false (1 or 0). The real challenge here is a player can press two or three buttons at the same time. Furthermore, that situation actually is a must in Super Mario Bros. The player has to press at least two buttons at the same time to finish the first level of the game successfully. For example, to jump over a cliff in level, the player has to press jump button and the right direction button at the same time.

The action space where more than one discrete value can be selected is called multi discrete action space. There can be more than one values which are true at the same time such as pressing two buttons at the same time. On the other hand, Q-learning algorithm is not compatible with multi discrete action sets due to its nature. It picks one action among all choices which gives maximum reward for state.

In brief, making output layer of the neural network to be controller buttons directly was not a good solution for this problem. Instead, multi discrete action space is converted into discrete action space where there has to be only 1 true value and a selected action among all choices. Meaningful actions on controller such as pressing left, pressing right, pressing right with sprint button, pressing left with jump button become possible action choices for the agent using a wrapper inside Gym Super Mario. Unmeaningful actions such as pressing left and right direction buttons at the same time are discarded. That way, Q-learning algorithm worked as intended.

To make learning process from actions more stable, repeated action method is used in this project. When agent takes an action, that action is repeated for n-1 times at the next frames. It allowed agent to learn outcomes of actions in a more stable way. The value of 2 is found as the best (choosing an action at every 2 frames), meanwhile the values more than 2 caused sluggish actions and made Mario to react slowly.

3.2 Agent Model

3.2.1 Q-Learning

Q-learning is a popular learning algorithm used in machine learning. It is a model-free reinforcement learning technique. Therefore, an information about the model of the environment is not required for the agent. Q-learning is an off-policy learning algorithm. As an off-policy method, behavior policy may be unrelated to the policy that is evaluated and improved. It is based on a simple hypothesis that the action should be chosen as the one which will have the largest positive impact in results in future. The algorithm uses Q values to determine that action. A high Q value means a high reward is expected if that action is taken.

Q-learning algorithm can be implemented using tables to store Q values. Each cell in the table represents a state and holds the Q value of related state. However, this method loses its strength quickly with increasing number of states. In my project, there are total of 2^{416} different states which makes using a look-up table impossible. Another option which is used in this project is function approximation using a deep neural network. This approach generalizes experienced situations and can approximate Q values for states never seen.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \lambda \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

$$\text{The New Action Value} = \text{The Old Value} + \text{The Learning Rate} \times (\text{The New Information} - \text{the Old Information})$$

Figure 3.4: Formula used to update Q values [14]

Q-function gets a state and an action as its parameters and returns a Q value which represents the expected total reward of doing that action at that state. The expected total reward is composed of two parts, immediate reward and discounted future reward. The agent needs to consider future rewards as well as immediate rewards to perform well in long term.

The value seen as λ is the discount rate. It is a floating-point number between 0 and 1 and determines how valuable future rewards are. A small value of discount rate indicates that future rewards are not that important compared to immediate rewards. Meanwhile, a large value of discount rate tells us future rewards are important as well as immediate rewards.

If we let our agent to make choices at first which will maximize future reward, it may cause our agent to learn a suboptimal policy. It will keep doing the same action whereas there are other actions which may give better rewards. However, if it acts completely randomly to find better ways, it may never learn a good policy. This is generally called as exploration-exploitation dilemma. To solve that dilemma, it is generally a good approach to make the agent do more exploration at first but allow it to exploit know solutions while time passes.

Epsilon-greedy method for exploration is selected in this project. This means that when an action has to be selected in acting phase, it is chosen either as an action with the largest q-value, or an action randomly selected. Choosing between these two is random and based on the value of a variable named epsilon which is also called as exploration rate. Initial value of the epsilon is 1 which means agent will act completely random at the beginning of the program. A function will decrease that value linearly at the end of each episode. After reaching that number, the program will stop decreasing it. With that way, exploration still continues by choosing random actions in a small amount at later episodes.

3.2.2 Network Structure

The most basic unit of computation in a neural network is a node. It is also called as a neuron and works in a similar way with neurons in human brain. All nodes except the nodes at input layer receive input from some other nodes. There is an associated weight for each input connection in a node. The values of weights are based on node's relative importance to other inputs. The outcomes from input nodes are multiplied with their related weights. Then, all those values are summed including bias. Bias is a constant

number which also affects the output of a node. That value can be changed at training phase just like weights. How a node functions can be seen in [15, Fig. 3.5].

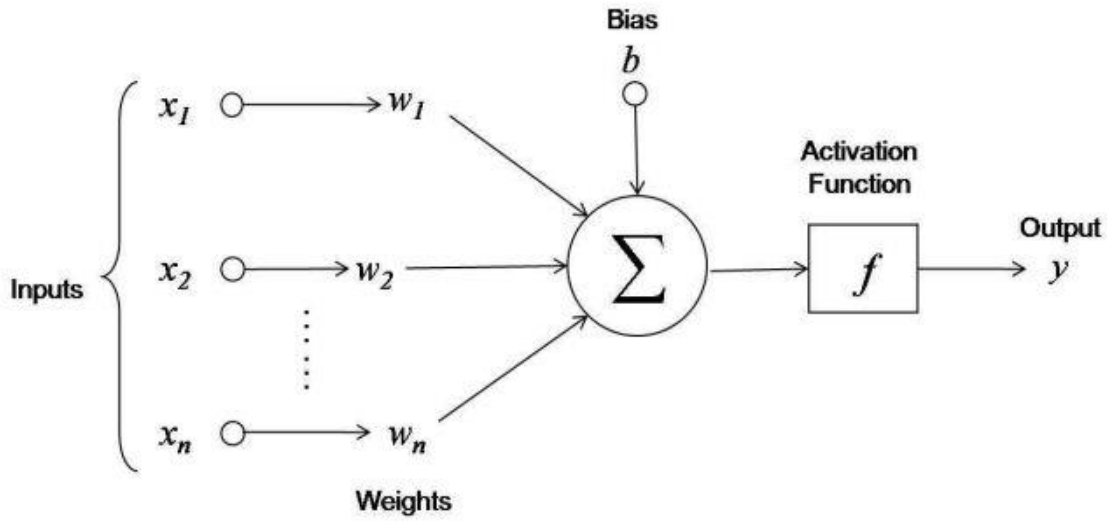

Figure 3.5: Graphical representation of a node in neural network [15]

After the final sum is achieved, calculated value is sent to a function called activation function. Those functions are relatively simple functions but highly affects behavior of a neural network. The most popular activation functions used in neural networks are sigmoid, tanh and ReLU functions. The returned result from activation function is the value of output of that node.

ReLU is selected as activation function in hidden layers. The biggest advantage of ReLU is it greatly accelerates the convergence of stochastic gradient descent compared to the sigmoid or tanh activation functions. It performed six times faster than an equivalent network with tanh activation function according to Krizhevsky et al [16]. Furthermore, there is no saturation problem as in sigmoid and tanh functions. However, ReLU is not an ultimate perfect solution as an activation function, because ReLU nodes are fragile and can die. That means ReLU nodes may output always the same value for any input and become unlikely to be recovered.

The neural network used in this project is a feedforward neural network. The output of nodes moves in only one direction and there is no connection between the nodes that would create a cycle. The collection of the nodes which are operating together at a specific depth is called layer in feedforward neural networks. In general, there are three different types of layers in a neural network. The first layer at the beginning of a neural network is called input layer. On the other hand, the last layer at the end of a neural network is called output layer. The layers which process and transmit information from input to output layer are called hidden layers.

My neural network has total of 5 layers in total. The first layer is the input layer and has 824 different nodes inside it. It receives state information from the environment. The reason why the number of 824 is the size of the input layer was explained in state data subtopic.

Second, third and fourth layers are hidden layers. They process the data which comes from the input layer and generate and output for the output layer. In each hidden layer, there are 256 nodes. Hidden layers are where the magic happens in deep learning. Their weights and bias values are starting to change in training phase which is what we call as saying "learning".

The fifth and the last layer of my network is output layer. It has total of 16 nodes inside it which equals the number of meaningful action choices in the game. The final outcome appears here with given state information in input layer. Agent choose to take an action with a highest value appeared in output layer. This value indicated the expected future reward of related action.

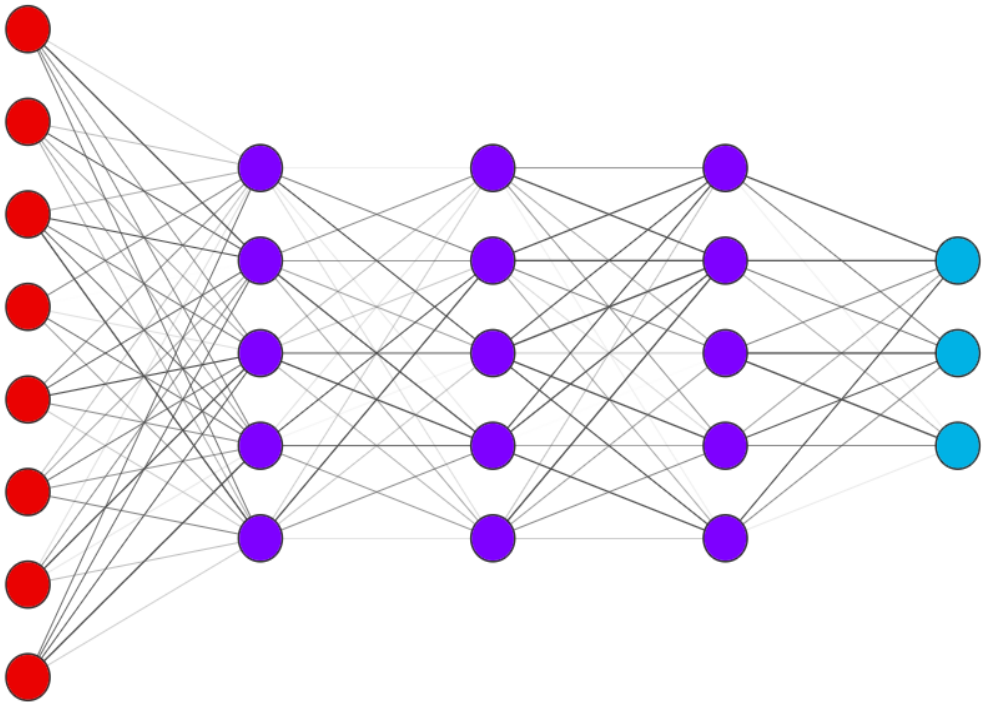

Figure 3.6: Analogy of the network used in this project. Red nodes represent input layer, purple nodes represent hidden layers and blue nodes represent output layer.

While initializing the nodes in neural networks, the weights of nodes do not have the same initial value for a good reason. Otherwise, the nodes would produce same output with same error values and the model would not learn anything. The most common solution was using random numbers within some parameters as initial weights. In 2015, a weight initialization technique has been published by He et al [17]. In this method, the weights which are actually still random are determined depending on the size of the previous layer. This approach helped loss function to be minimized in a faster way. This method is used in my project at weight initialization stage without having to choose any hyperparameter.

There is a regularization technique named dropout in neural networks used to prevent overfitting. Dropout works by ignoring randomly chosen nodes. At each training phase, nodes and their connections are either dropped out of the neural network with probability p or kept with probability 1-p. It is observed that better performance is achieved using this technique at many application areas such as object classification, speech recognition and

computational biology data by Srivastava et al [18]. In the light of this information, it can be said that dropout is a well-performed general method in classification problems. In Keras, dropout method can be implemented by adding it as an artificial layer to neural network. That layer will dropout nodes and connections on previous layer with selected p parameter.

3.2.3 Hyperparameters

There are parameters called hyperparameters in machine learning which has to be determined and set before the beginning of a learning process. Those parameters can affect the performance and the robustness of a network. Sprague published a great article and showed results of different parameter selections for deep q-learning networks [19]. The results of his research are found useful in this project to determine hyperparameters. Hyperparameter include variables such as node count, learning rate, discount rate and batch size. The hyperparameters in this project are mostly founded by investigating q-values, errors, visual output as well as using articles about this topic.

Node count in input layer and output layer are already determined according to state and action spaces. A general rule about determining the hidden node count is to keep number of nodes in a hidden layer to between input and output nodes. The number of 256 is selected as node count in each hidden layer. That number is big enough to perform and learn well, but small enough not to slowdown the learning process.

Learning rate which also called as step size is one of the most crucial hyperparameters of a neural network. It basically can be explained as how quickly a network abandons old experiences for new ones. It controls how much optimizer will adjust the weights of a network according to the loss gradient. A decay rate for learning rate is not used, as a result of Adam optimizer which used in this project handles this problem and decays learning rate itself [20]. This value varies depending on developed models.

The suggested dropout probability was 0.5 by Srivastava et al [18]. However, it is founded that the 0.5 value was causing underfitting too much. The agent was found to keep doing illogical actions and there was a huge variance between Q values. Degrading that value by 0.1 each step did not solve the problem. Even the final value 0.1 was a bit problematic. At last step, it is deprecated from the model. Dropout method might have worked well on classification problems, despite it does not seem working well on regression problems.

The value seen as λ is the discount rate. It is a floating-point number between 0 and 1 and determines how valuable future rewards are. A small value of discount rate indicates that future rewards are not that important compared to immediate rewards. Meanwhile, a large value of discount rate tells us future rewards are important as well as immediate rewards. The value of 0.9 created a nice balance between immediate rewards and future discounter rewards. Therefore, that value is used for discount rate.

There is another hyperparameter which is known as epsilon or learning rate. It starts from 1 which means it will act completely random at the first episode, meanwhile this value will be decreased to a minimum exploration rate. Generally, it is advised to determine a minimum exploration rate else than zero, because, at later episodes the agent will only

exploit the path it knows and will not try to do other actions which may yield higher reward. After observing the behavior of the agent, this value is kept at minimum 0.2 which allows a nice exploration chance even at later episodes.

Batch is a common term used in machine learning. It means a portion of whole available dataset. A crucial hyperparameter, batch size, refers to the size of a sample utilized in one iteration. To observe its effect on the system, it is included in a detailed experimentation. The experimentation is about when and how to train the agent to achieve the best results with three different models. The details about this experimentation is explained in the next chapters.

The final values of hyperparameter for this project can be seen as below:

- Hidden layer size: 3
- Node count in a hidden layer: 256
- Maximum exploration rate: 1
- Minimum exploration rate: 0.2
- Discount factor: 0.9
- Dropout probability: 0
- Learning rate: Depends on the model (0.000001, 0.0005, 0.00001)
- Batch size: Depends on model (1, 512, 32)

3.2.4 Training Models

Three different training models with different batch sizes, learning rates and training methods are developed to observe difference among them clearly. There are also comparative performance tests as the first experiment for these three models. Since any difference between models caused by number of processed frames in training process is not something desired, it is kept about at the same value for all models. They share the same hyperparameters if such hypermeter is not implied in model design. The model which will perform the best will be selected as the main model for this project.

3.2.4.1 Model A

The first model is developed with stochastic gradient descent with online training. Stochastic gradient descent is a paradigm which uses only a single experience to update the network in one go. On the other hand, online training is a model where learning happens on each input as they come. It is updating the weights once a new sequential data hence a state is acquired by the agent. There is no need to keep old experiences in memory. This property allows online learning to have minimum space complexity compared to other models. The main advantage of this property is it allows agent to be trained on systems with very small sized memories. The batch size can be considered as 1 as there is actually no batches. The main characteristics of developed program is stated as follows:

- Memoryless approach
- Training based on a single experience each time
- Keeps training while playing

- Changes weights frequently while playing
- Used with a low learning rate

```
for episode number N to 250 do
    while episode is not finished do
        Get state s_t from environment;
        if random value is smaller than ε then
            Take a random action a_t;
        else
            Take the action a_t which maximizes predicted Q(s_t,a_t) value;
        end
        Get the new state s_{t+1} and reward r_{t+1};
        Train the model using experience E(s_t,a_t,r_{t+1},s_{t+1});
    end
end
```

Figure 3.7: The pseudocode of Model A

3.2.4.2 Model B

The second of the developed model is based on offline training. In offline training, there is no training process and updating the weights of nodes while playing. Training phase and playing phase are separated completely. Training process is done at the end of each episode when the related episode is over. This approach needs a data container to store old experiences. A double-ended queue with size of 20,000 is used as experience memory in Model B. It can hold memories of average of last 50 episodes and automatically deletes the oldest member in memory, when it is full and a new element is inserted. As for gradient descent algorithm, a large batch gradient descent is selected for this model. After an episode is ended, the agent gets a sample from experience storage with a large sample size. The number of data samples taken from a dataset to train network in one time is called batch size in machine learning. The batch size is 512, because it is found that each episode lasts for nearly 400 frames. The number of processed frames is desired to remain same between models. The main characteristics of the developed model and program is stated as follows:

- Utilizes memory to save old experiences
- Training based on large batches
- Training and playing phases are separated
- Changes weights only at the end of the episode
- Used with a high learning rate

```
Initialize the experience memory M with size 20000;
for episode number N to 250 do
    while episode is not finished do
        Get state s_t from environment;
        if random value is smaller than ε then
            Take a random action a_t;
        else
            Take the action a_t which maximizes predicted Q(s_t,a_t) value;
        end
        Get the new state s_{t+1} and reward r_{t+1};
        Store experience E(s_t,a_t,r_{t+1},s_{t+1}) in memory M;
    end
    Get a random sample S with size of 1024 from memory M;
    Train the model using sample S as a batch;
end
```

Figure 3.8: The pseudocode of Model B

3.2.4.3 Model C

It is also possible to train a neural network while data still comes in and using batches at the same time. It is the approach used by Model C. It trains and update weights in the network as online training; but, the training is done after every 32 frames instead of every single frame. It uses memory and train in batches as Model B program does. However, its batch size is much smaller than Model B. It has a batch size of 32, so number of processed frames are almost same as other models. A batch which contains only small number of experiences to train the network is usually called as a mini-batch in machine learning. I tried to achieve a midpoint between two previous models when designing this model. The main characteristics of developed model and program is stated as follows:

- Utilizes memory to save old experiences
- Training based on small batches
- Keeps training while playing
- Changes weights seldom while playing
- Used with a medium learning rate

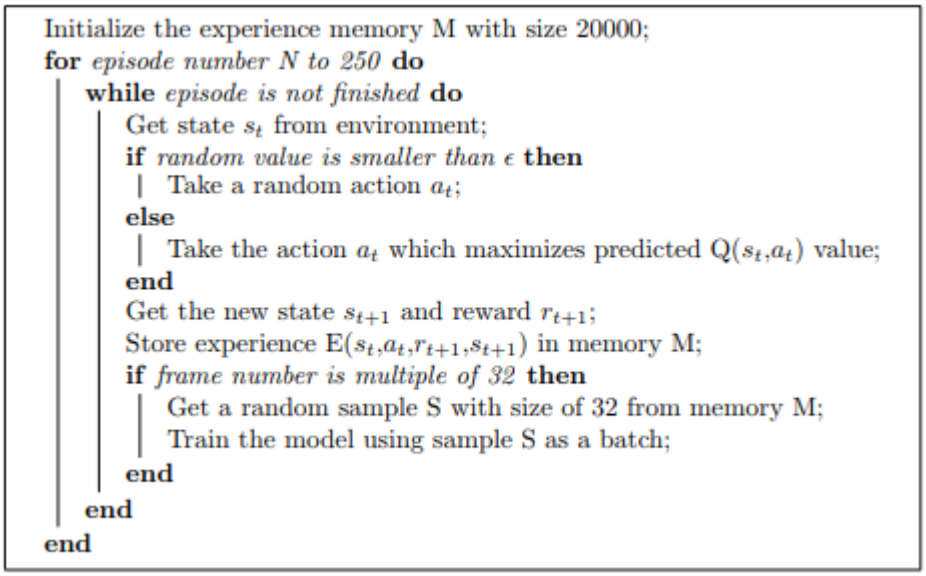

Figure 3.9: The pseudocode of Model C

4 Experimentation Environment and Experiment Design

4.1 System Environment

4.1.1 Software Specifications

The system run on my desktop computer. The applications which used to develop the final system in this project are stated as below:

- Primary neural network API: Keras 2.1.5
- Backend machine learning API: TensorFlow 1.6.0
- Operation System: Ubuntu 16.04.4 LTS
- Python IDE: PyCharm 2017.3
- NES Emulator: FCEUX 2.2.2
- Backend environment: OpenAI Gym 0.92
- Primary environment: Gym Super Mario 0.0.7

All of these components worked together in harmony. The only exception was the environment used to play Super Mario Bros. named Gym Super Mario. Unlike other software, Gym Super Mario is developed by a single person. It required an older version of OpenAI Gym to be installed, and there were few bugs in codes which needed to be fixed. Initialization of this environment was not an easy process either.

The final work environment can be seen in Figure 4.1. The emulator returns the regular pixel display and also the tile-based display which is the what the agent observes in this project. After each episode, metrics such as score, average reward, error (MAPE) are printed in terminal to evaluate the training process.

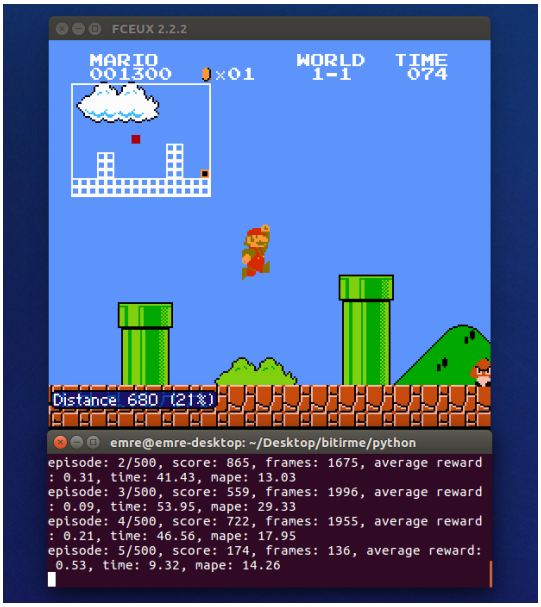

Figure 4.1: A screenshot of the work environment

4.1.2 Hardware Specifications

The system run on my desktop computer. The crucial parts of hardware are stated as below:

- CPU: Intel Core i7 2600k
- RAM: Kingston 16GB DDR3
- Storage: Samsung 850 EVO 256GB SSD

Using GPUs instead of CPUs to train deep neural networks became more and more popular due to advancement and nature of GPUs. GPUs started to offer not only graphics, but also calculation power which can be used in applications besides video games. GPUs offer much greater parallelization compared to CPUs. They have many small cores inside which make them a good choice to process floating point operations at the same time. CUDA, a closed source API invented by Nvidia, is supported by TensorFlow to utilize GPU power to train neural networks. However, CUDA can be used only on Nvidia branded graphics cards. Since, the graphic process unit of my computer is not Nvidia branded and does not support CUDA, I used only the CPU which is 7 years old to train neural network. That means results are achieved with a system which can be considered mediocre at its best. Using an SSD instead of HDD helped to prevent bottlenecks caused by I/O operations.

The CPU on my computer, which has 4 cores and 8 threads, also supports AVX instruction set. AVX instruction set is an extension to x86 architecture and inherits instructions about floating point operation. AVX started to be supported in processors in 2011 by both Intel and AMD. According to official performance guide of TensorFlow, AVX instruction set may yield up to 82% increase in performance compared to aged SSE3 instruction set [21]. Although, the performance was still not near of GPUs, it helped to achieve results in shorter time.

4.2 Experiments

4.2.1 Model Test

In this test, a detailed experimentation will be done between three different training models named Model A, Model B and Model C respectively. The details and pseudocodes about these models are mentioned at previous chapters. The model which gives the best results overall will be selected for the next experiments. There will be total of 250 episodes which equals to approximately 100,000 frames for each model. The programs for each model will generate a csv file at the end of their training including various metrics about training process. Three metrics will be used to determine the best overall performer among these models.

The first metric is loss or error function. It symbolizes the amount of forecasting error while predicting the final Q-values. It is found that Q values and errors are increasing in each episode with standard mean squared error loss function which is used to train network. The program discovers new states and expected total reward increases by time. The error function MSE which is used to train the network in Q learning is not a good indicator for an agent’s performance in deep q networks for that reason. The error value

founded by MSE may keep increasing over time. Instead, another error function named mean absolute percent error, also known as MAPE shortly, is used to evaluate error. The formula of MAPE can be seen in [22, Fig. 4.2]. It divides the absolute difference by the real value and acts as a better measurement for forecasting. It does not change the training process or algorithm of Q-learning, we will use MAPE just as a measurement. Low error rate indicates a better model unless it is overfitted to training data.

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i - \widehat{Y}_i|}{Y_i}$$

Figure 4.2: The formula of mean absolute percent error [22]

Another measurement will be on earned average reward per frame in episodes. This value is found as sum of rewards earned in an episode, divided by frames spent in that episode. It is a good measurement to find out how quickly agents learns to follow an optimal path. It is desired that this value to be high.

As the last of them, total time elapsed during the processes will be used as a performance metric. Time is a valuable source and efficiency is one of the main objectives in the engineering disciplines. The model which uses less time to get same results can be preferred over other models. It is the best if we can train our model in a time as little as possible.

4.2.2 Training Test

In this part, the objective is the winning the first level of the Super Mario Bros. using the best model from previous experiment. To achieve that goal, episode numbers will be increased gradually. If this approach is not helping enough, hyperparameters will be tweaked again. After, the model archives to win the first level a few times, the training process will be stopped. Because, training a model more than needed may cause overfitting which will make the same model perform poorly on the next test named generalization test.

It can be measured whether Mario successfully passed the first level by looking at the score from csv file. The maximum score the agent can get is 3250 points as it indicates the position of the end of the level.

4.2.3 Generalization Test

It is desired that a neural network can learn a general solution as its policy for all situations that can be countered in that problem. A good way to determine whether a model is learning a general policy and not overfitting is trying it with different dataset which is usually called test set in literature.

To do this, the weights of nodes of the trained model from previous stage will be saved on a file. Another python program which loads weights from a file will be implemented. Note

that, this and the last experiment does not involve any training process unlike previous experiments. The program will evaluate the loss to print MAPE error function, however it will not train the network, hence will not change the weights. What it does is simply playing the game while measuring.

There are two models and programs developed for this comparison. They will be tested on the World 3-2 of Super Mario Bros which can be seen in [23, Fig. 4.3]. The first model which is named random model can be considered as a control group. It will make a choice uniformly random among 14 different logical actions. The second model acts according to network trained on first level of Super Mario Bros. It will take actions according to result of neural network with 20% randomization. That randomization percent is breaking down the deterministic behavior of the agent by involving some stochastic elements. Otherwise, since there is no training involved, actions would be same at same states. It will be observed that how much a trained model can perform better than a random model.

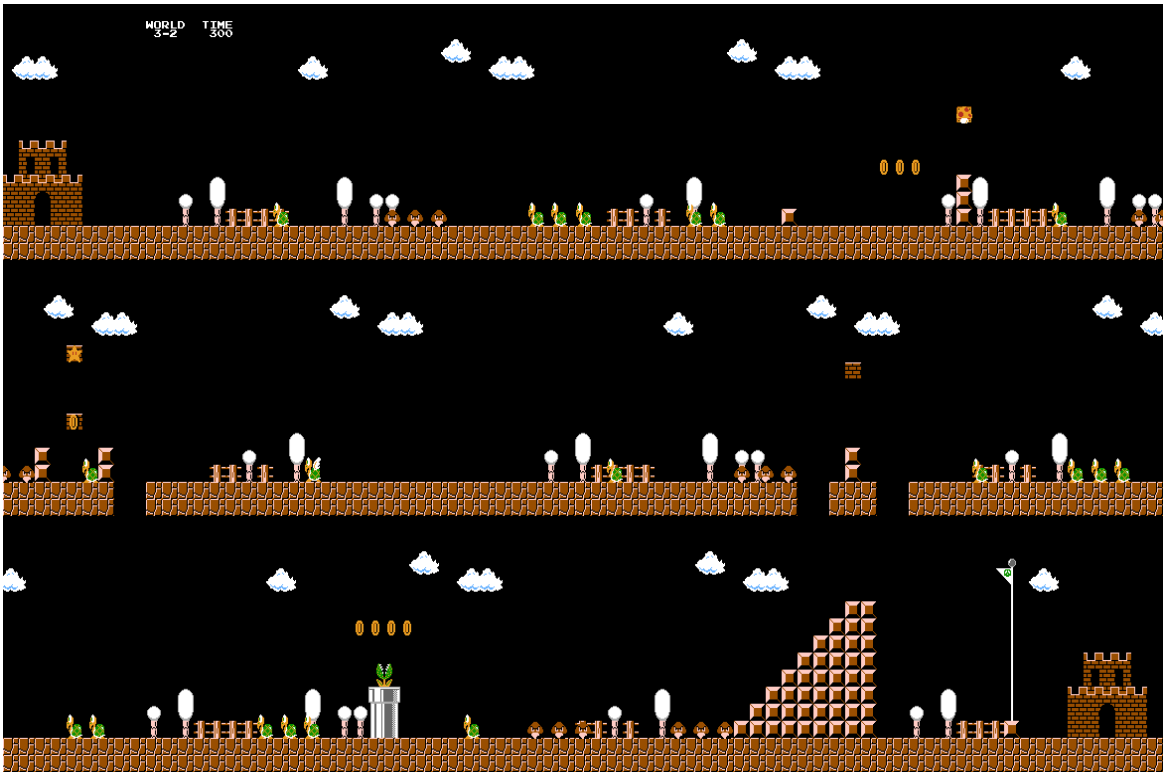

Figure 4.3: The map of World 3-2 of Super Mario Bros. [23]

5 Comparative Evaluation and Discussion

5.1 Experiment Results

5.1.1 Model Test Results

The csv files generated by python program of each model is used to plot following graphs. A small python program is taking csv files as its input and save figures in a png file. While plotting graphs, a filtering algorithm named Savitzky–Golay is used to smooth the lines.

Mean absolute percentage error (MAPE) is acquired as the first metric. It shows the error in predictions when determining Q values. Generally, a nice slope is expected via tuning learning rate. Both a really steep or almost horizontal linear line is considered as a bad model. In first situation error will converge quickly but remain high and will not decrease as expected; meanwhile, in second situation it will take a lot of time for error to converge.

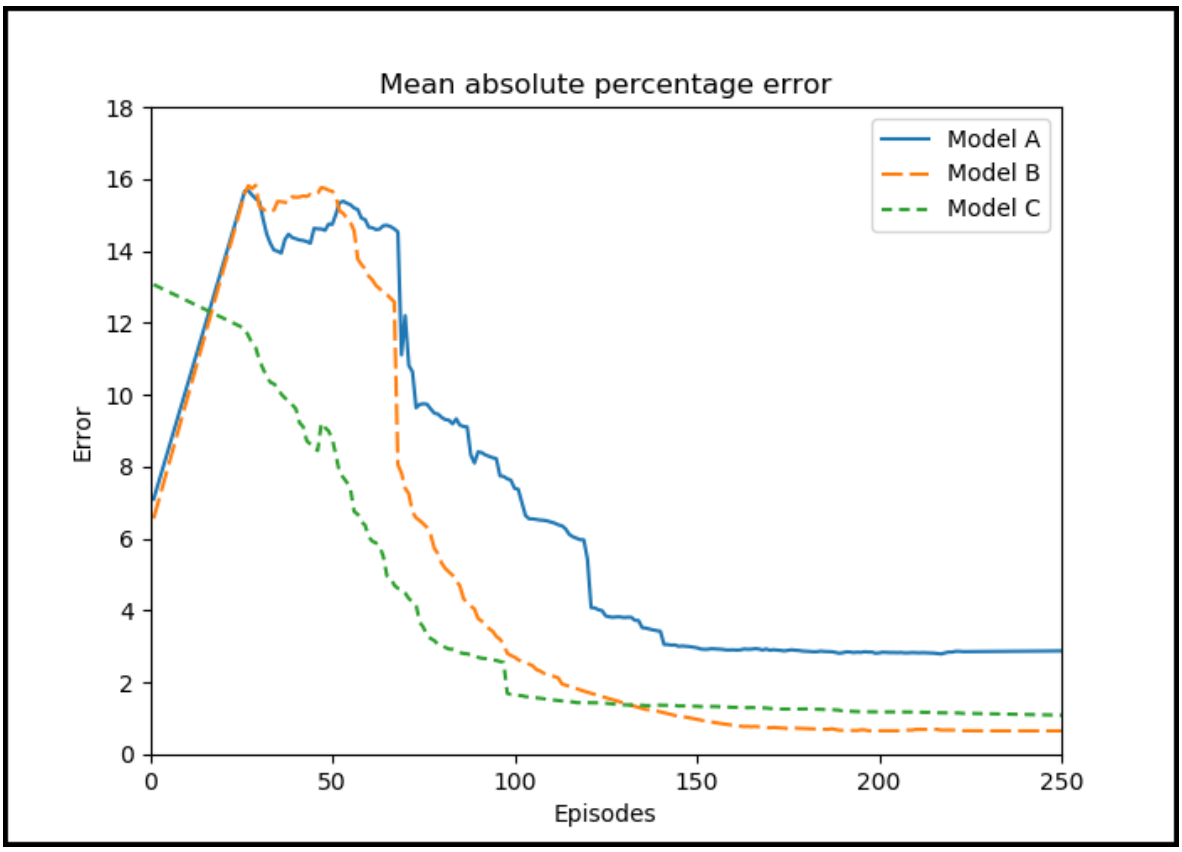

Figure 5.1: Means of absolute percentage error in episodes

After several times of tweaking learning rate, the graph which can be seen in Figure 5.1 is achieved. The graph tells us Model B was the best model at minimizing the error where Model C is the close second. Model A could not converge well and error remained over 3.0. It shows us Model B is a better model at predicting expected Q value.

Figure 5.2: Earned average reward per frame in each episode

As average reward per frame, Model A performed better than other models which can be seen in Figure 5.2. Changing and adapting weights of nodes frequently shows its benefit here. The second-best performer was the Model B while Model C follows it behind. It can be said that it is easier to find a policy hence learn faster with online learning method. The other methods required more episodes to trained on to earn same amount of rewards per frame in each episode.

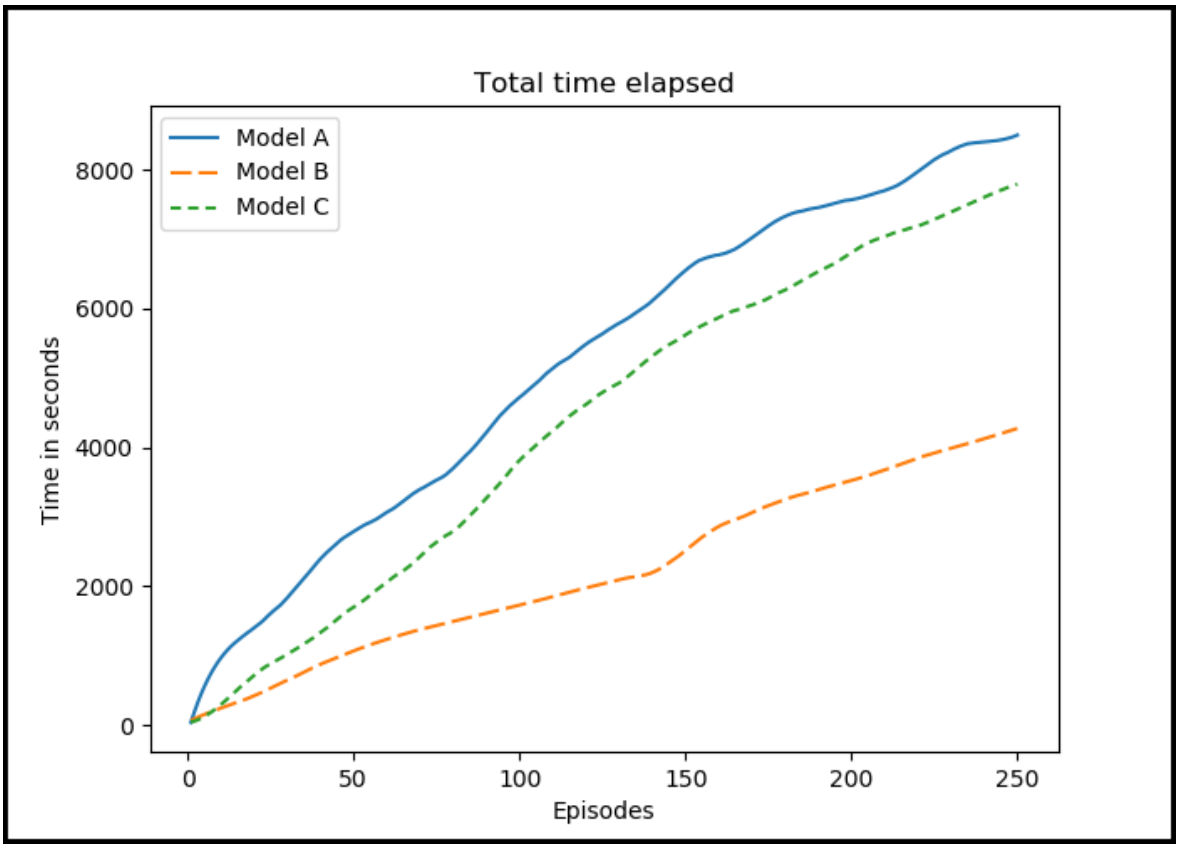

Figure 5.3: Total time elapsed during training

As the last measurement, we took a look at the elapsed time including both playing and training phases which can be seen in Figure 5.3. The total time elapsed was the highest in Model A. An episode lasted average of 33.93 seconds in this model. Model C achieved to finish an episode with 31.23 seconds which is a slight improvement over Model A. Model B, on the other hand, finished episodes in a much shorter time compared to other models. The average time of an episode was 17.06 seconds in this model. It is observed that time elapsed for training is not increasing at the same rate we increase the batch size. The count of training operation has more impact on elapsed time than the batch size. Training with large batch sizes with minimal weight updates lead to a greater efficient in time. Model B seems as a clear winner in this measurement. That means, it is a more efficient approach to train neural networks in time.

At last, we have to determine the best model overall to proceed. Model A started to learn faster than other models. However, it performed with a drawback which is a constant high error rate. As for efficiency, it performed the worst among all three developed models. Model B never performed the worst in any experiment. Despite the Model B earned a medium amount of reward, it has the lowest error and the time need to spent to train network. Model C, on the other hand, never performed as the best, and actually was the worst at the average reward per frame. In the lights of these information, Model B is selected as the main model to be used in next experiments.

5.1.2 Training Test Results

Model B is chosen as the primary model in this test due to its performance at previous stage. In this experiment, the aim is achieving to win the first level of Super Mario Bros. Firstly, number of episodes are increased from 250 to 500. The information about the score is investigated. The graph can be seen in Figure 5.4 is acquired.

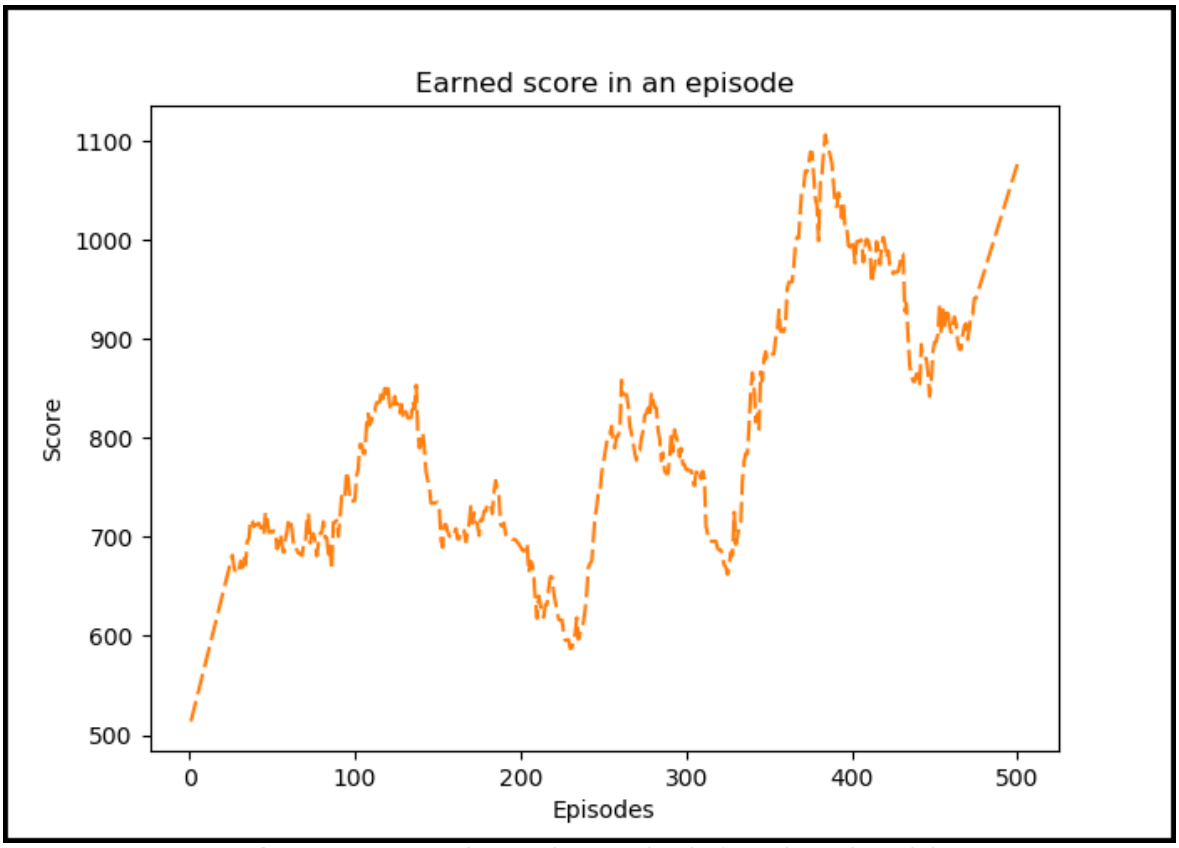

Figure 5.4: Earned score in an episode for selected model

The results were found fascinating. According to results saved in csv file, the agent managed to complete the first level for the first time at episode 370. Furthermore, it has successfully passed the first level two more times after that episode. Therefore, the agent was able to win the first level for 3 times in total.

From that point, episode number is not increased. There are two main reasons for that. The first of all, the agent was able to win the first level, actually 3 times, which was the main objective of this project. Since this goal is achieved, increasing episode number and even more tweaking seemed to be redundant.

The second and maybe the more significant reason is avoiding overfitting. When an agent keeps training on same dataset or level, it will make better predictions about that level; however, once it is tried with another dataset, it may show worse results. It is a good rule in machine learning to stop training after a point.

5.1.3 Generalization Test Results

As for the last experiment, the program is tested on a new level of Super Mario Bros. To observe if it will perform better than a random model and can make some logical actions, a comparison is made between two models. The achieved results can be seen in Table 5.1.

Table 5.1: Comparative results of models

	Average Reward	**Average Score**	**Highest Score**
Random Model	0.304	253.76	1177
Trained Model	6.269	561.6	2017

The average reward per frame earned in an episode showed a definite improvement. As the random model does not moved moves to maximize the reward, that value kept to remain very low. On the other hand, the result of the trained model showed us it was indeed making moves to get maximum reward possible.

The average and the highest score are approximately doubled. Although, the agent could not complete this level as it completed the first level, results tell us that Mario managed to go further by trained model.

The mean absolute percentage converged to approximately 0.5 in previous experiment. Since we use a new data which agent has never seen before in this experiment, the average of MAPE value is increased to 0.8. The increase on the error rate was not critical and shows that the new states did not cause a much trouble for the agent.

5.2 Discussion

The main objective stated in the interim report was passing the first level of the Super Mario Bros using a deep neural network. That objective is achieved successfully at the end of the project by designing a competent network with correct hyperparameters. Compared to other works in literature, the training process is realized using a much less frame count. For example, the number of episodes and frames used to train the agent was low compared to Deepmind where they trained the agent for total of 10 million frames [6]. The hardware used in this project did not have a great computation power either. Moreover, the game which agent tries to play was a more complicated and difficult game to play. However, the state information in this project was based on a tile-map meanwhile in the other projects they usually worked with raw pixel input with convolutional layers. Despite of all of this, agent still managed to complete first level of the game.

It is observed to found an optimal policy, although this policy can be suboptimal and not perfect. It was not making very clever choices and could not learn tricks such as collecting power-up items or taking a shortcut via pipes. It is observed that agent may need to train on a specific level for a lot of times to learn these tricks. It can be accomplished by lowering

learning rate and increasing frequency of training. However, it may need some advanced tweaks and optimizations to prevent overfitting in that case.

6 Conclusion and Future Work

In this project, the main goal was implementing a system which can play an arcade video game using a deep neural network. A video game from NES gaming console, Super Mario Bros, is selected as the video game which agent will play on. The goal is achieved by developing an intelligent agent using deep learning method.

Three training models are developed then deeply investigated. Training the neural network while separating the playing and training phases with large sized batches is founded as a good performer with the best efficiency among other solutions.

It is proved that deep learning can work quite well using mediocre computer hardware in real environment. Although, the problems in the real video games are more complex than synthetic ones, the system managed to win first level of Super Mario Bros.

The trained model showed impressive results when compared to model taking random actions on another level of Super Mario Bros. Although it could not manage to win that level, it showed us the model learned a general policy to progress in the game.

It is advised to use MAPE instead of MSE as the error metric to evaluate the system performance for people working on deep Q networks due to increasing Q values over time.

One of the things that can be done as a future work is changing state input from a tile-based map into raw pixels on the display. That can be achieved by adding convolution layers at the beginning of the neural network. However, the data which will come from raw pixels will not be in a good quality as tile-based input. Implementing this approach requires some knowledge about digital image processing. This approach also needs more time and refinement compared to approach used in this project.

The other thing that can be implemented as a future work is making a comparison between different learning algorithms. The learning algorithm used in this project was Q-learning. Although, it performed well and helped to achieve my goal, there are other algorithms such as SARSA (State-Action-Reward-State-Action) or gradient temporal difference which can be a better choice than Q-learning. According to Zhao et al, deep SARSA learning performed better than deep Q-learning when training the model to play two arcade games [24].

7 References

[1] C. Smith *et al*., “The History of Artificial Intelligence”, University of Washington, Dec., 2006. Available: https://courses.cs.washington.edu/courses/csep590/06au/projects/history-ai.pdf

[2] A. Munoz, “Machine Learning and Optimization”, Courant Institute of Mathematical Sciences, n.d. Available: https://cims.nyu.edu/~munoz/files/ml_optimization.pdf

[3] M. Bojarski *et al*., “End to End Learning for Self-Driving Cars,” *arXiv:1604.07316 [cs]*, Apr. 2016.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. Cambridge, Massachusetts: The MIT Press, 2016. Available: http://www.deeplearningbook.org

[5] G. Tesauro, "Temporal Difference Learning and TD-Gammon", *Communications of the ACM*, Vol. 38, No. 3, Mar. 1995.

[6] V. Mnih *et al*., “Playing Atari with Deep Reinforcement Learning,” *arXiv:1312.5602 [cs]*, Dec. 2013.

[7] A. Defazio and T. Graepel, “A Comparison of learning algorithms on the Arcade Learning Environment,” *arXiv:1410.8620 [cs]*, Oct. 2014.

[8] A. Gullì and S. Pal, *Deep learning with Keras: implement neural networks with Keras on Theano and TensorFlow*. Birmingham Mumbai: Packt, 2017.

[9] G. Pandey and A. Dukkipati, “To go deep or wide in learning?” *arXiv:1402.5634 [cs]*, Feb. 2014.

[10] I. Safran and O. Shamir, “Depth-Width Tradeoffs in Approximating Natural Functions with Neural Networks,” *arXiv:1610.09887 [cs, stat]*, Oct. 2016.

[11] Redbubble, Super Mario Bros World 1-1. [online] n.d. Available: https://www.redbubble.com/people/stowball/works/16243810-super-mario-bros-world-1-1 [Accessed May 21, 2018]

[12] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, Second edition. Cambridge, MA: The MIT Press, 2017.

[13] Wikipedia, List of Nintendo controllers. [online] n.d. Available: https://en.wikipedia.org/wiki/List_of_Nintendo_controllers [Accessed May 18, 2018].

[14] J. Bennett, Machine Learning, part III: The Q-learning algorithm, (May 2016). [online] Available: https://articles.wearepop.com/secret-formula-for-self-learning-computers [Accessed April 6, 2018].

[15]. Cornell University, Neural Networks and Machine Learning. [online] Available: https://blogs.cornell.edu/info2040/2015/09/08/neural-networks-and-machine-learning/ [Accessed April 25, 2018].

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[17] K. He, X. Zhang, S. Ren, and J. Sun, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”, *arXiv:1502.01852 [cs]*, Feb. 2015.

[18] N. Srivastava *et al*. “Dropout: a simple way to prevent neural networks from overfitting”, *The Journal of Machine Learning Research*, vol. 15, pp. 1929-1958, Jan. 2014.

[19] N. Sprague, “Parameter Selection for the Deep Q-Learning Algorithm”, 2015. Available: https://w3.cs.jmu.edu/spragunr/papers/rldm2015.pdf

[20] D. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization”, *arXiv:1412.6980 [cs]*, Dec. 2014.

[21] TensorFlow, *Performance Guide*. [online] Jul. 2018. Available: https://www.tensorflow.org/performance/performance_guide [Accessed Feb. 6, 2018].

[22] Veri Bilimcisi, *MSE, RMSE, MAE, MAPE ve Diğer Metrikler*. [online] July 2017. Available: https://veribilimcisi.com/2017/07/14/mse-rmse-mae-mape-metrikleri-nedir/ [Accessed May 13, 2018].

[23] Super Mario Wiki, *World 3-2 (Super Mario Bros.)*. [online] n.d. Available: https://www.mariowiki.com/World_3-2_(Super_Mario_Bros.) [Accessed July 14, 2018].

[24] D. Zhao, Haitao Wang, Kun Shao, and Y. Zhu, “Deep reinforcement learning with experience replay based on SARSA,” 2016, pp. 1–6.