CHAPTER 1

Introduction

1.1 Overview

There are three main learning strategies. The first one is called Supervised Learning [1] or as it is also known as learning with a teacher where the desired response is known. In supervised learning the system tries to fit the model with the given input-output pairs. It consists of an environment, a teacher who has a priori knowledge of the environment, while the learning system doesn't. The learning system tries to predict the value from the given input from the environment, the loss function takes the desired response and the actual response and outputs an error signal which is sent back to the learning system.The learning system then corrects its model, accounting for the error. Therefore after several iterations the actual response and the desired response will be similar in nature. The block diagram of a supervised learning system has been depicted in Fig.1.1(a).

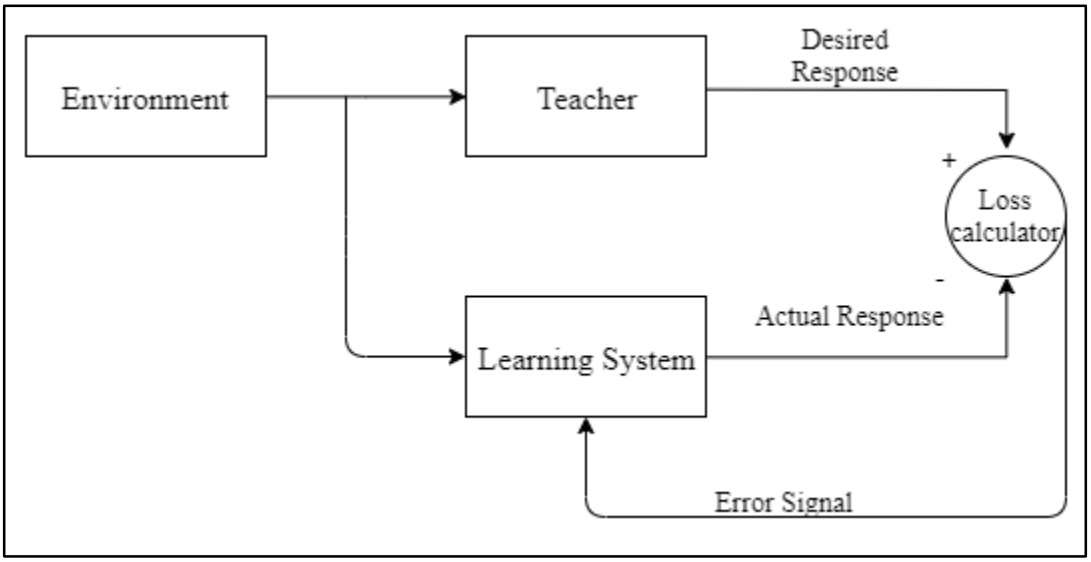

Fig.1.1(a) Block diagram of supervised learning system [1]

The second learning strategy is Unsupervised learning [1] which is also known as self-organized learning. In this strategy there is no teacher and hence no error signal is

generated. In unsupervised learning a task-independent measure is made to assess the quality of the learning and this is done prior to feeding of inputs to the learning system. By using this measure the learning system starts to learn the statistically relevant data from the input pairs and from this data it starts to create its own representation of the data and furthermore creates new classes automatically [2]. A block diagram has been pictorially represented in Fig.1.1(b).

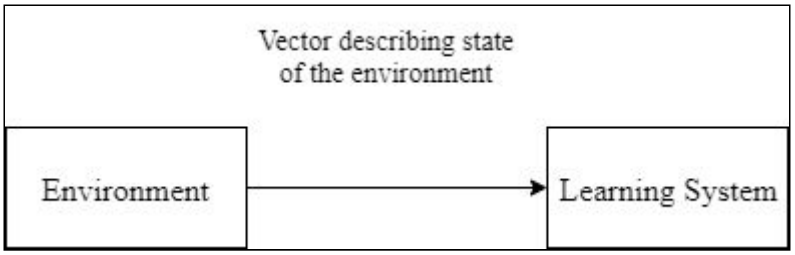

Fig.1.1(b) Block diagram of unsupervised learning [1].

The third and last learning strategy is Reinforcement learning(RL) [1] , in this learning strategy while there is a teacher, the purpose of the teacher is different from that of the teacher in supervised learning. In supervised learning an error signal is used by the learning system to update the model. In reinforcement learning the learning system interacts with the environment by generating actions which alter the state of the environment which leads to the environment generating a signal. The primary goal of the learning system is to optimise certain parameters of its model so that it can maximise its cumulative reward [1]. In reinforcement learning, there is also the presence of another entity besides the teacher called the critic. The main purpose of the critic is to convert the signals or outputs from the environment into signals which can be interpreted by the learning system such as rewards which can be both positive and negative in nature. While in other learning strategies the system learns from the error or from the statistical nature without any delay, however this is not the case when it comes to reinforcement learning, a delay exists as the system observes a temporal sequence of information from the environment via the critic. Reinforcement learning is difficult to use as due to the presence of a delay there is no teacher to provide a desired response at each step of the sequence, the entire reinforcement learning model can be seen in Fig.1.1(c). In the image the output from the critic is a heuristic reinforcement

signal and from the environment there are two outputs one is the state vectors and the others is the primary reinforcement learning signal [1].

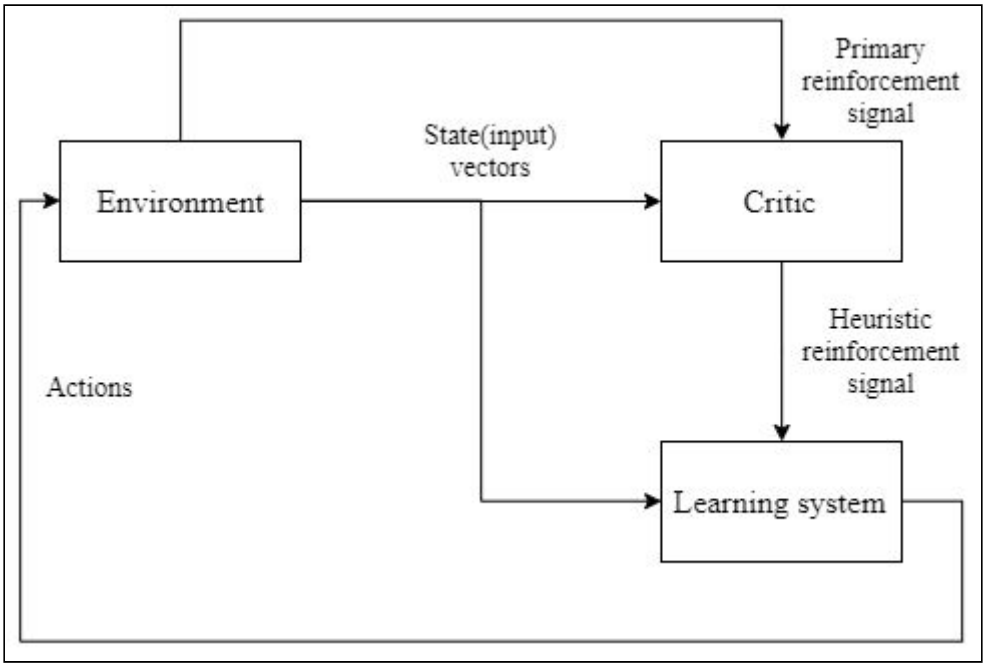

Fig.1.1(c) Block diagram of reinforcement learning system [1]

1.2 Basics of Machine Learning and Deep Learning

While there are several algorithms which were proposed over the years, such as Support Vector Machines [4], Decision tree based algorithms such as Gradient Boosted Tree [5] and Random Forest [6] and Neural networks [7] to name a few. The neural network approach has become on of the more popular approaches due to its ability to model nonlinear input-output mappings and generality as a neural network can be made to behave as a binary classifiers, multi-category classifiers, regressive models such as linear regression [8], polynomial regression [9] and logistic regression [10] . Neural networks can also be made to behave as generative models [11], sequential models [12]. A simple neural network with one hidden layer is pictorially represented in Fig.1.2(a).

The input vector x is a column vector of size $n_x (n_x \in N)$, the hidden layer values are the input vector transformed by a non-linear parameter function which is a matrix of

size $n_h \times n_x$ ($n_h \in N$) additionally there is also a bias term b_1 of size n_h .The bias term is added to the transformed input vector x .

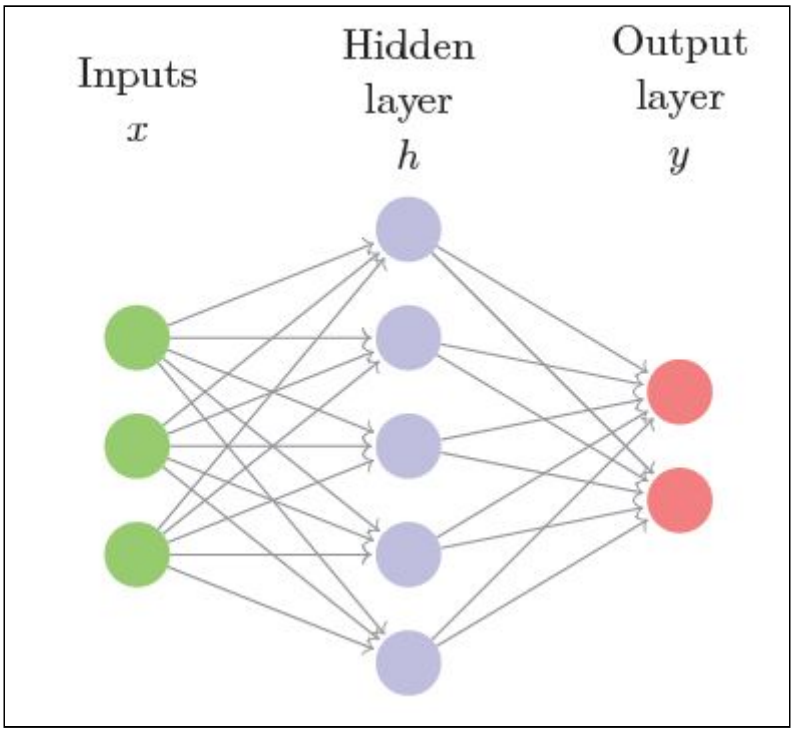

Fig.1.2(a) A Neural Network with one hidden layer [3]

$$h=A(W_1 \cdot x+b_1) \tag{1.1}$$

Where A is the activation function which can be either linear or non-linear in nature based on what the neural network is being applied for [1]. In sequential modelling activation functions such as tanh is preferred while in fields such as computer vision functions such as ReLU(Rectified Linear Units) [13], leaky ReLU [14] are used predominantly. The activation function helps in adding non-linearity to the neural network at each layer and furthermore makes it possible for neural networks to model nonlinear functions due to the availability of nonlinear activation functions. The output y is also another column vector y of size n_y($n_y \in N$) W_2 is of size $n_y \times n_h$ and the bias vector is also of size n_y. Several layers can be stacked on top of each other, they all would follow these equations. The bias term is optional and to further optimise it, in some places the bias term is constant and a weight is also applied to it [1].

$$y=A(W_2 \cdot h+b_2) \tag{1.2}$$

The parameters of the neural network are optimised to minimise the error, and the most basic formula would be to use stochastic gradient via backpropagation [1]. The rate the free parameters are modified is controlled by the learning rate η. A larger value would lead to system oscillating as it takes a large time to converge on the optimum value while a smaller value will take a long time to converge. Therefore it is recommended to start with a large value of η such as 0.6 and decrease it as time moves on. The error signal e is based upon the neural network's output y and the desired value d for supervised learning [1].

$$e = f(d, y) \tag{1.3}$$

The general formula for updating weights is given by the equation below where the parameters at $n+1$ iterations is equal to the parameters at the n^{th} changed by the error calculated which is then multiplied with the learning rate η [1] .

$$w(n+1) = w(n) - \eta \cdot \nabla e(w) \tag{1.4}$$

Another term which should be defined here is epoch and batch size, one epoch is when the neural network has iterated over the entire input dataset once and the number of epochs define the number of times the system will iterate over the input dataset [1]. The input dataset can be iterated over it by passing one vector at a time and then calculating the error and updating the system. An alternative to this would be to pass a batch of input vectors calculate the error for the batch and then update the system, batch size is defined as the number of input vectors which are passed before the system is updated , it is recommended for the batch size to be a power of two such as sixty-four , thirty-two one hundred and twenty-eight, for efficient memory allocation [1].

While in machine learning, handcrafted features are derived from the input dataset which are then fed to the learning system, taking an example in a speech recognition system, the features from the acoustic speech would be its Mel Frequency Cepstral Coefficients (MFCC)[15] . In deep learning there are several hidden layers, furthermore no handcrafted feature is fed to the system. The input dataset is used as it is and the learning system derives its own handcrafted feature, in other words in machine learning feature selection is explicit in nature while in deep learning the feature selection is implicit in nature [1]. A deep learning network used for object detection is depicted in Fig.1.2(b).

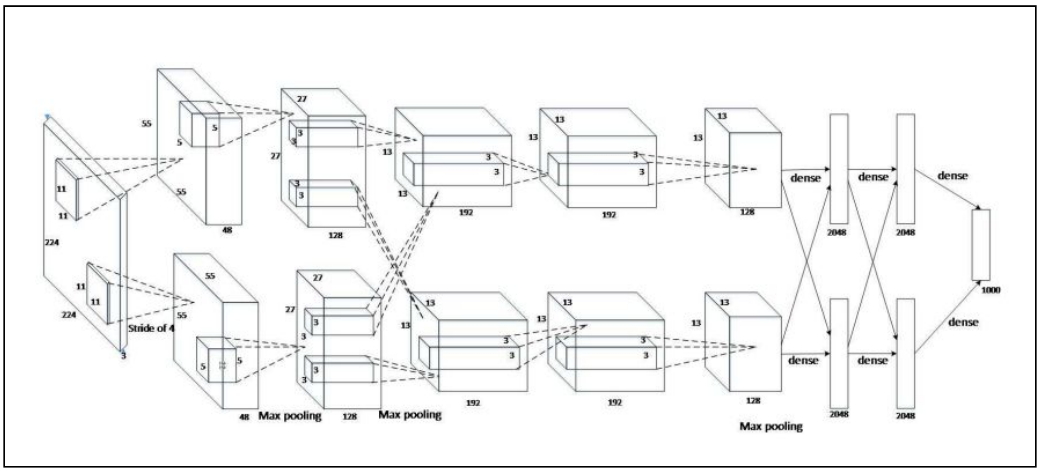

Fig.1.2(b) A Deep learning network used for object detection [16].

1.3 Basics of Reinforcement Learning

While machine learning and deep learning are specifications of the network topology and the way the input dataset is fed, as explained before reinforcement learning is a learning strategy that dictates how the learning system will update its free parameters [1]. In reinforcement learning there is an agent which interacts with the environment. The way the learning system learns to figure out which actions the agent should enact to minimise its loss and hence maximise reward is through trial and error [3].

There are two broad categories to divide them into. Offline learning which is also called Batch Reinforcement learning is when the learning system has a priori knowledge of the environment prior to learning [3].

Before explaining on-line learning it is necessary to talk about the exploration/exploitation dilemma [17]. It is a tradeoff in RL systems where exploration is about obtaining new information from the environment while exploitation is about maximising the cumulative reward from the information it has currently gathered from the environment. The tradeoff is about either learning new information about the environment or maximising the reward from the current information it has. Therefore using this dilemma to further explain on-line RL, where there is no a priori knowledge

of the environment and the learning system and it acquires data on the go, however in on-line learning since there is no a priori knowledge it has the possibility of varying the amount of exploration/ exploitation from the environment [3].

Before discussing about RL systems any further, another problem RL systems experience is the temporal credit assignment problem [3] . It is because the RL system experience delays because the system observes a temporal sequence of events, therefore the system is unable to accurately put the blame on one such an event it has enacted in the past [3]

Moving onto the solving the RL problem, usually it is broken up into four variables, which are s_t the state transition $s_t \in S$ which describes the state transitions of the environment [3]. The reward is denoted by $r_t \in R$ and the observations obtained by the agent are denoted by ω_t where $\omega_t \in \Omega$, and the actions taken by the agent are denoted by a_t where $a_t \in A$. From these four variables a RL system can be modelled and has been depicted in Fig.1.3(a) [3].

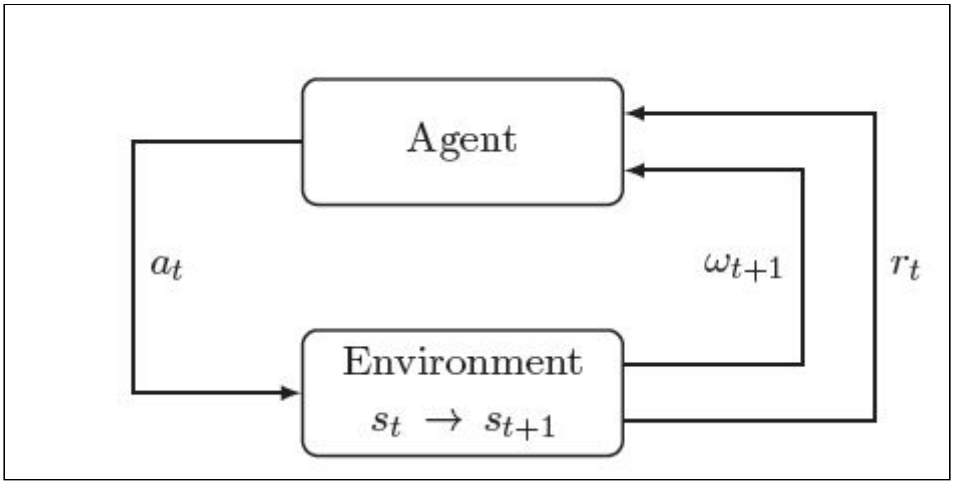

Fig.1.3(a) Agent-environment interaction in RL[3].

1.4 Motivation

While Reinforcement learning is a well established field, Deep Reinforcement learning is an up and coming field and has found applications in domains ranging from the

financial sector the self driving cars. In the march towards creating a general artificially intelligent system, reinforcement learning has its part to play. As several physical tasks such as walking, grabbing an item , playing video games all of which are acquired via trial and error with the prospect of accumulating a reward hence the reason for choosing reinforcement learning for this manuscript is precisely this. To become familiar with the concepts of reinforcement learning and how it has been applied to solve real world problems.

1.5 Objectives of Report

1. To understand the basic concepts of machine learning, deep learning and explain their differences
2. To understand the basics of reinforcement learning and the framework defined
3. A study of the several algorithms present in this field in a chronological order
4. To understand the best performing algorithms present in reinforcement learning
5. To study the applications where reinforcement learning has been applied

1.6 Outline

In this chapter the three different learning mechanics were covered and the basics of machine learning, deep learning their differences and an introduction to neural networks was covered.

In Chapter 2 the basic machine learning and deep learning concepts will be covered from the beginning starting from regression to decision trees then moving onto neural network based layers such as fully dense, convolutional layers and finally recurrent neural networks.

In Chapter 3 the basics of reinforcement learning and the framework used will be defined, followed by a brief literature survey encompassing the three different learning methods and the best performing algorithms from this chapter are discussed in the subsequent chapter which is Chapter 4.

In Chapter 5, assuming the reader now has a basic knowledge from the previous 3 chapters, the applications where reinforcement learning will be covered with respect to the algorithm used and the network configuration used.

CHAPTER 2

Machine Learning and Deep Learning Models

2.1 Overview

While the difference between both machine learning and deep learning has been covered in the previous chapter, however for the sake of continuity it will be reiterated here as well. In machine learning systems the input dataset is not fed as is, instead several features which are derived from the input data are then fed instead to the learning system. In trying to predict the timing of earthquakes[18], the acoustic data was measured via a piezoceramic sensor, A moving time window is applied to the input and several features are extracted from each window such as mean, variance, kurtosis, autocorrelation and skew to name a few, in total 100 features were used which are then used in the random forest, the predictions are averaged over 1,000 decision trees [18].

While in Deep Learning, there is no explicit feeding of features to the learning system. While this field is relatively new, as for several years there was a lack of computational power. However with the advent of graphic processing units(GPU) deep learning simulations were possible, also the interest in deep learning picked up around 2012 when a deep learning model [19] surpassed all other previous machine learning models in the Imagenet dataset [20]

2.2 Regression

Regression is one of the many models used in both machine learning and deep learning systems. Regression can only be used in a supervised learning setup as there is a requirement that there is an input-output pair as the model tries to fit the values of x to y.

There are several types of regression models but a few will be covered here.

1. Linear Regression

2. Polynomial Regression
3. Logistic Regression

Starting with linear regression,the learning system assumes that there is a linear relationship that exists between the input x and output y [8]. Where m is the slope of the graph and c is the biasing term also called y-intercept is a constant term which is added to the input [8].

$$y = mx + c \tag{2.1}$$

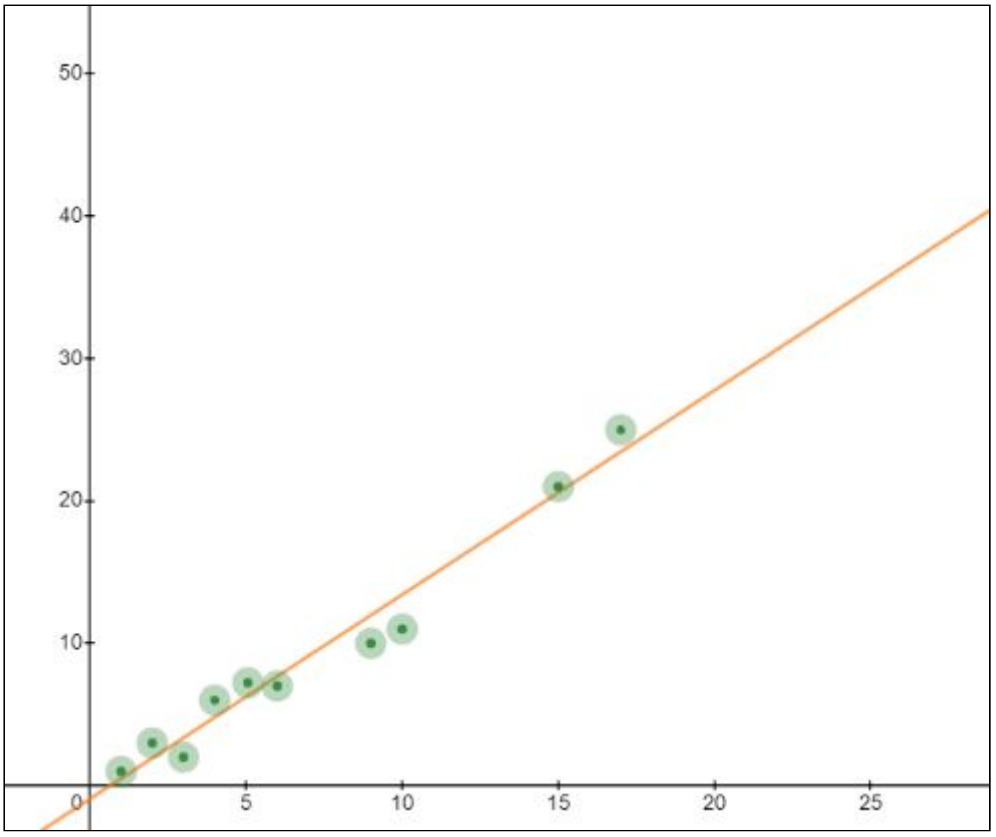

Fig.2.2(a) An example of linear regression [21]

An example of single variable linear regression has been depicted in Fig.2.2(a) where the line is $y = 1.43551x - 1.02129$ [8]. In all regression models the curve is fitted in such a way that there is minimum error between the input-output dataset, in linear regression, this is called the line of best fit [8].

However linear regression is simple in nature and easy to implement, it is extremely difficult to model polynomial systems with linear regression therefore there is another branch of regression called polynomial regression [9].

$$y = ax^n + bx + c \tag{2.2}$$

It is similar to linear regression however the learning system fits the input-output data using a polynomial function where a is the slope of the graph and c is the biasing term, or in other words the y-intercept [9]. An example of polynomial regression has been depicted in Fig.2.2(b) with $y = 0.929x^2 - 0.271x + 0.8$ as the line of best fit [9].

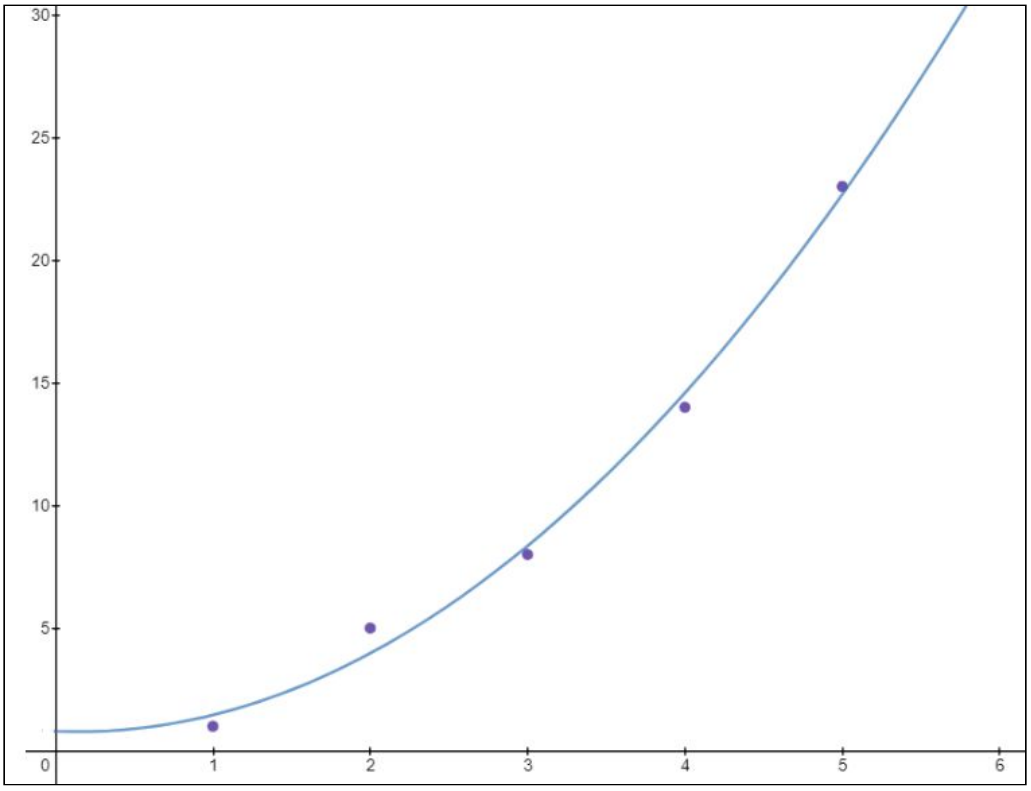

Fig.2.2(b) An example of polynomial regression [21]

Logistic regression is a type of regression which is modelled on the logistic function for binary dependent variables, it can be used for binary classification furthermore it can be used for predicting the probability of a class with an output one means class 1 and an output zero means class 2 [10].

$$f(x) = e^{a+bx} \tag{2.3}$$

$$y = f(x)/(1 + f(x)) \tag{2.4}$$

2.3 Decision Trees

Decision trees are another model used which are generally used for binary,multilabel classification and when used with the gradient boosting setting can also do tasks such as regression [5]. When used for classification, the output data should be have a finite number of states, or labels in other words. To further expand on there are two types of trees which will be discussed here which are listed below.An example of a decision tree has been shown in Fig.2.3(a)

1. Random Forests
2. Gradient Boosted Tree

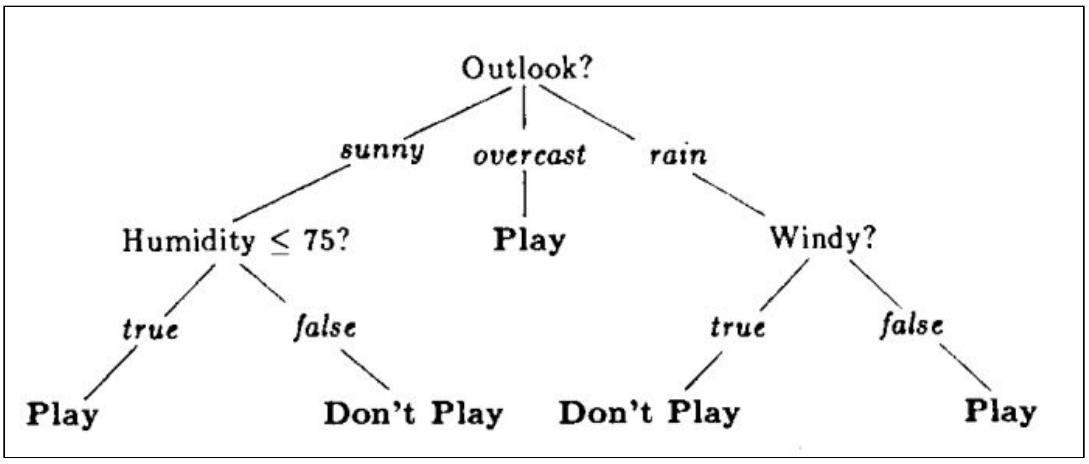

Fig.2.3(a) A decision tree for binary classification [22]

Starting with random forests [23], they are formed from tree predictors and each tree depends on a set of randomly sampled input vectors. The tree then inspects the error of generalisation and further decides if a branch in the tree needs to be split further, this is based on the confidence interval set. If the error is greater than the confidence interval more samples are randomly sampled and the tree is split further, on the contrary, if the error is less than the confidence interval the branch will not split. This is done over several iterations and the results are average this is a case of ensemble learning [24]. A sample random forest is depicted in Fig.2.3(b).

Another powerful algorithm used in gradient boosted trees [5], which are different from random forests as the input data is not sampled randomly. In gradient boosted trees, it scans all the data points once before deciding the number of split points in the tree [5].

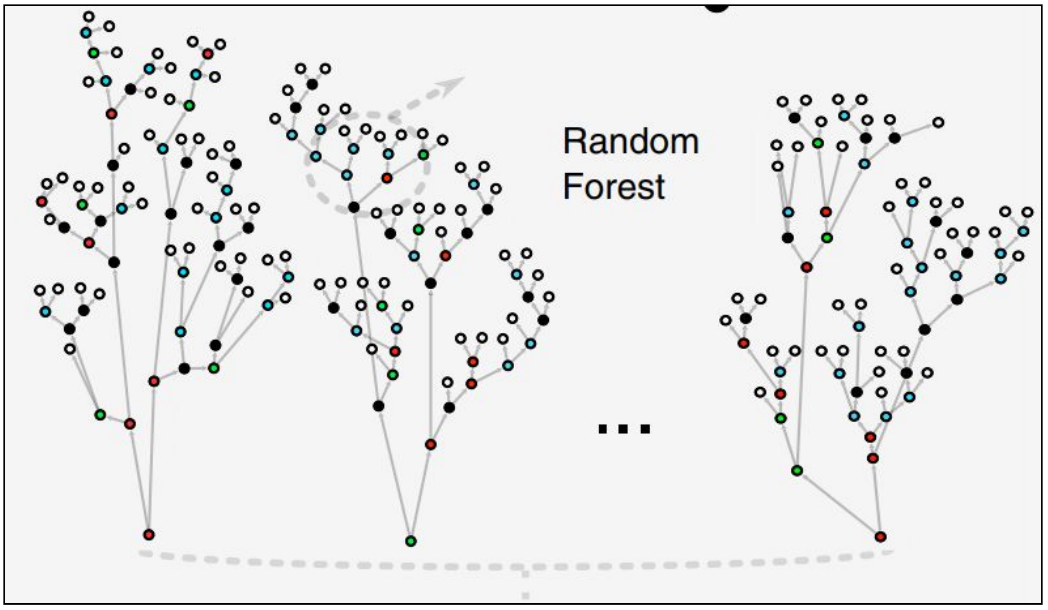

Fig.2.3(b) Random Forest shown with ensemble learning [18].

In each iteration the gradient boosted trees, calculates the remaining error which is also called the residual error and the algorithm which is usually used for finding the split points is the histogram, which is calculated upon the quantile information of the data [5]. Gradient descent is used to minimise the error at each stage, this would mean that each node on the tree has a separate weight assigned to it [5].

The predictions in gradient boosted tree are a sum of all the predictions made by a branch of the tree, this is due to the fact the error is considered additive in nature when minimised [5].

2.4 Artificial Neural Networks

While the basics of neural networks were covered in the previous chapter, in this chapter it will be discussed more in detail here. The neural network model is modelled based on the human brain. One of the first neuron models which were proposed was the

Mcculloch-Pitts neuron model which behaved in a binary manner [1]. Where the output v is a linear sum of the weights w multiplied with their corresponding inputs. The output y is binary in nature. If $v \geq 0$ then $y = 1$ else $y = 0$. A Mcculloch-Pitts model has been shown Fig.2.4(a).

$$v = \sum_{0}^{j} w_i \cdot x_i \tag{2.5}$$

$$y = f(v) \tag{2.6}$$

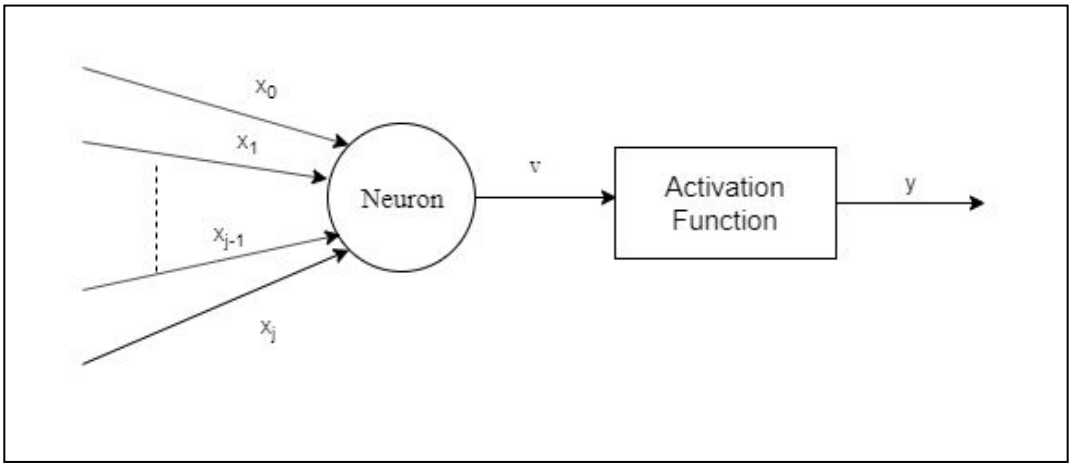

Fig.2.4(a) Mcculloch-Pitts neuron model [1]

While in the earlier model, the activation function was a binary function, later on another activation function was proposed where the output y would just be a linear sum of the inputs x multiplied with their weights w, however this lead to a few problems due to the fact that this lead to some neurons having large outputs which could be both positive and negative in nature and to further add to this it was virtually impossible with a linear or binary activation function to model nonlinear systems [1].

To model nonlinear systems, more activation functions were added which were nonlinear in nature such as sigmoid, tanh to name a few. Where the constant c determines the steepness of the sigmoid curve [1]. The graph of three sigmoid curves for $c = 1,2,3$ is shown in Fig.2.4(b).

$$y = 1/(1 + exp(cx)) \tag{2.7}$$

One of the problems faced with the sigmoid curve is that for large values of x and for small values of x it is possible for the output to saturate at values one and zero.Therefore the tanh activation function is used in places where such a property of the output being saturated is not desired [1].

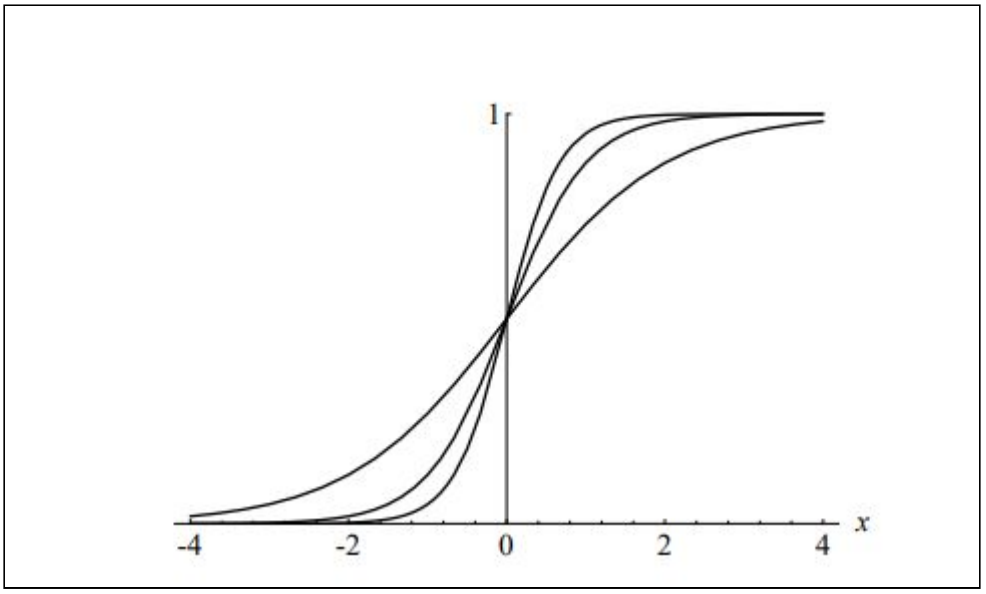

Fig.2.4(b) Sigmoid activation function graph for three different slopes [1]

The tanh activation function has been depicted in Fig.2.4(c), unlike the sigmoid function the output y can attain two values ,a positive one and a negative one as well [1].This makes it a more suitable candidate compared to the sigmoid activation function in certain applications, especially in sequential modelling [1].

The desirable properties an activation function should have:

1. Continuous at all points
2. Should be differentiable
3. No discontinuities
4. Output should not quickly saturate at certain levels
5. Should not have any points with a high inflection

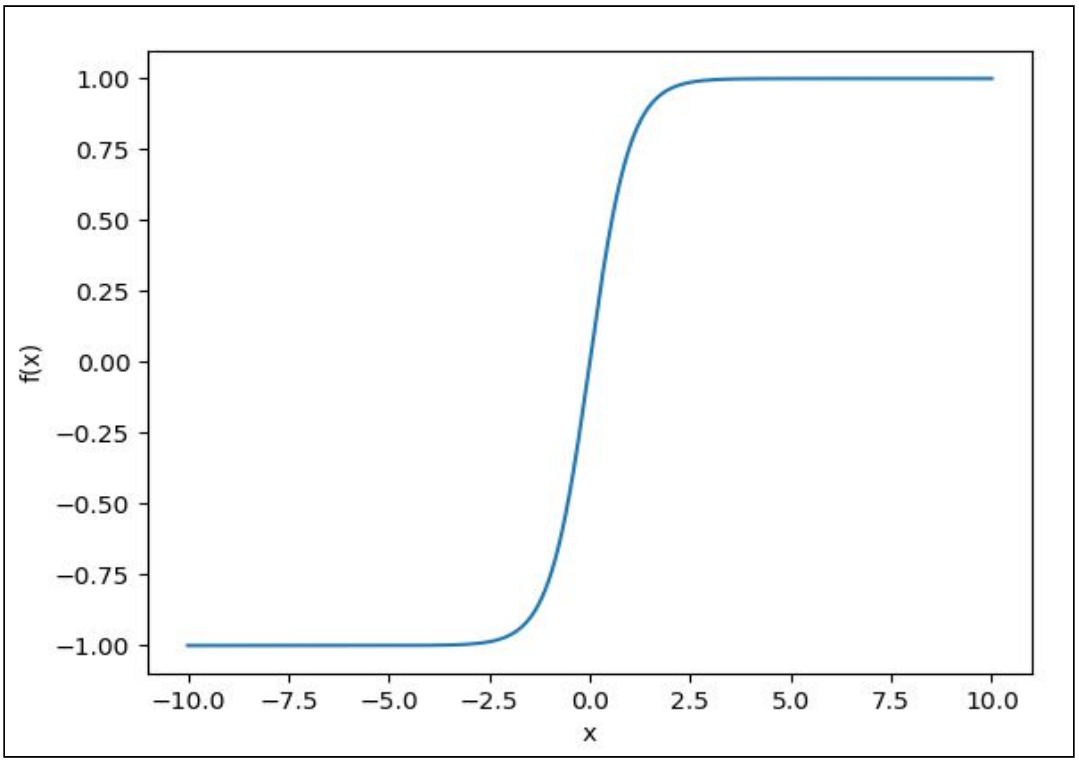

Fig.2.4(c) Tanh activation function [1]

Besides the activation functions used in the learning system, there are other parameters or requirements needed to pay attention to [1]. One such thing is the learning rate η which defines how fast or slow the system learns. If it is too large, the system will oscillate about a point taking a long time to converge and similarly if the learning rate is very small, while it won't oscillate it will also take a long time to converge due to the miniscule incremental steps it takes towards the local minima. Another note about the error surface is that it is never purely concave in nature [1]. A typical error surface is shown in Fig.2.4(d)

After the error is calculated between, by using backpropagation [25] which takes the error and distributes the error systematically distributed between the output layer, several hidden layers and the input layer.

While the backpropagation algorithm distributes the error to all the layers, the optimization algorithms such as the Adam optimizer [26], Stochastic Gradient Descent

optimization [27] decide how the parameters are updated from the back propagated error.

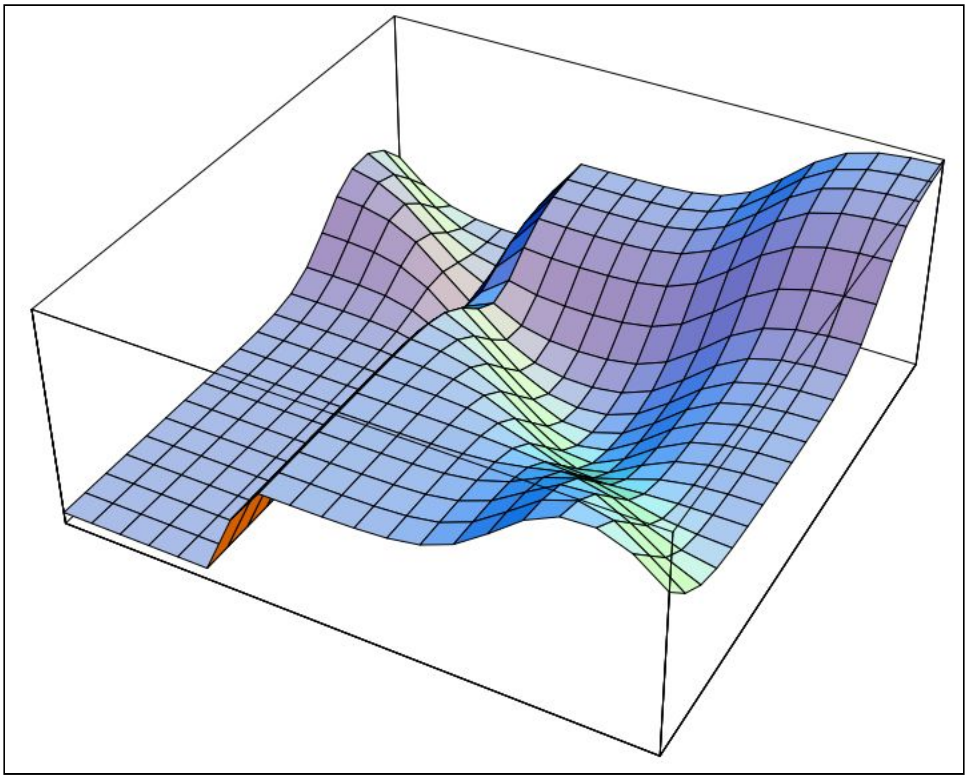

Fig.2.4(d) Typical error surface [26]

2.5 Convolutional Neural Networks

While ANN's can be used for singular dimension data, for input datasets which contain higher dimensional data such as images, videos where there are three channels [28]. A neural network is used which is more suited for images, this adaptation is called the convolutional neural network.A convolutional neural network is made of kernels whose size can be set. The stride can also be varied [28].

The kernel size decides how much of the image should be looked at in one instant while the stride decides by how much should the kernel slide over the image after each instant [28]. At each instant the kernel applies a transformation on the input image, by extracting its own features from the image such as the horizontal or vertical edges of the images. The number of filters also decided the number of kernels which should be applied to the image [28].

The dot product of the kernel with the image is then either passed to another convolutional layer or it can be passed through an activation function to add non-linearity to it. Some of the activation functions used are ReLU and Leaky ReLU. The process of a kernel working on an image is seen in Fig.2.5(a) and a convolutional neural network based system is seen in Fig.2.5(b).It can be seen that in Fig.2.5(b) that each layer has a different kernel value which is highlighted by the red indicator.

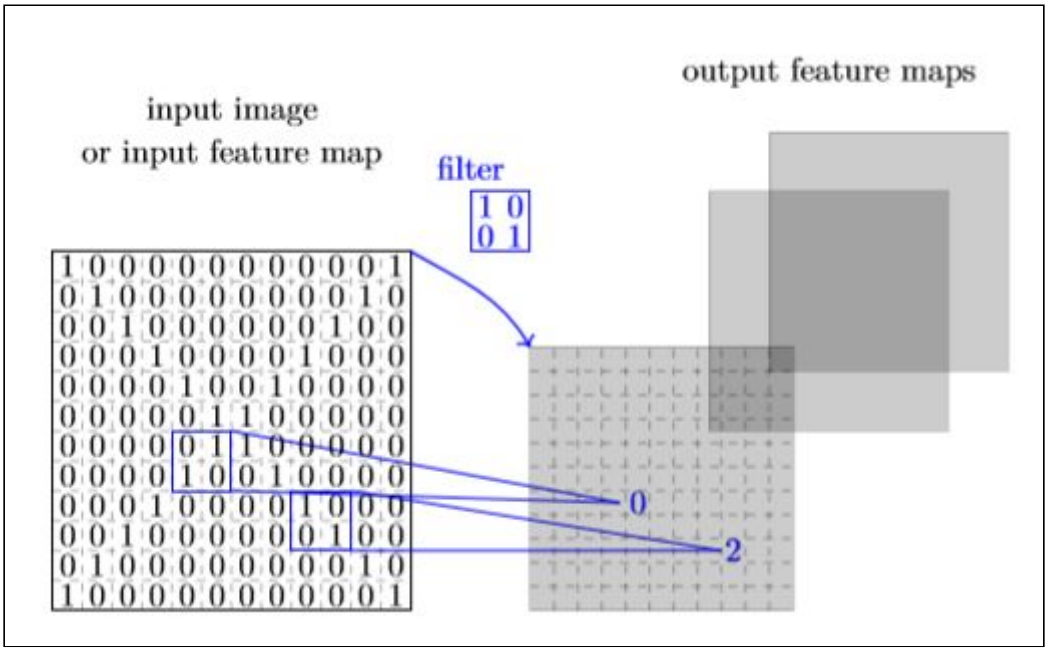

Fig.2.5(a) A kernel working on an image [3].

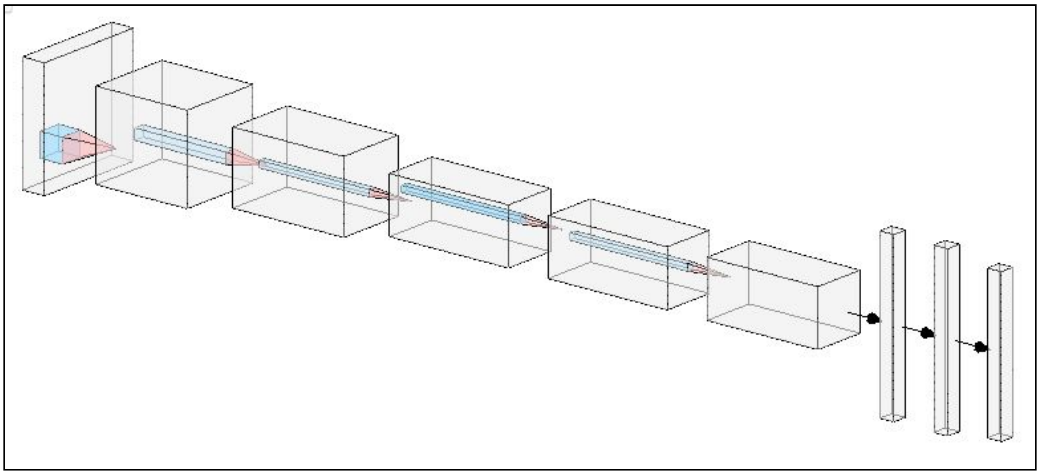

Fig.2.5(b) An example of a Convolutional Neural Network [28]

To further add to this the output of a convolutional layer can be interfaced with other layers such as fully dense layer which is also the other name for ANN's, this can be achieved by flattening the multi-dimensional output as seen in Fig.2.5(b) [28].

2.6 Recurrent Neural Networks

While fully dense layers can handle single dimensional data and convolutional neural networks can handle multi-dimensional data such as images and videos [1]. However these layers are not optimised or built for sequential data such as audio or any data which is time dependent therefore there is another section of neural networks optimised for sequential modelling [1].

In sequential modelling the output at time step t y_t is based on the current input x_t and the previous output or outputs as well y_{t-1} [1]. This gives the ability to model any recurrent system as a function of time t as well and on previous outputs as well [1].

$$y(t) = ax(t) + by(t-1) + cy(t-2) + ... \quad (2.8)$$

While theoretically a recurrent neural network should be able to remember information for an infinite duration of time however practically this isn't the case [29]. The recurrent neural network it can remember something only for a short duration of time. An example of a recurrent neural network system is depicted in Fig.2.6(a) where the left image depicts how the output is recursively fed back to the input and the other two images depict how the network looks like when it is unrolled in time [29].

To solve this problem of the recurrent neural network not being able to remember and to solve the problem of exploding/vanishing gradients where due to the presence of several layers when the error is back propagated either the error becomes almost zero by the time it reaches the first layer or the error gains a very large value [29].

The exploding/vanishing gradient causes the parameters of the learning system to change either too rapidly or not change at all causing the learning system to take more time to converge on a minima. The improvement on the recurrent neural network was the LSTM(Long Short Term Memory) which had three gates, an input gate, a forget

gate and a memory gate which decided how much of the input should be let in, how much of the memory should be forgotten and how much of the output should be memorised respectively [29].

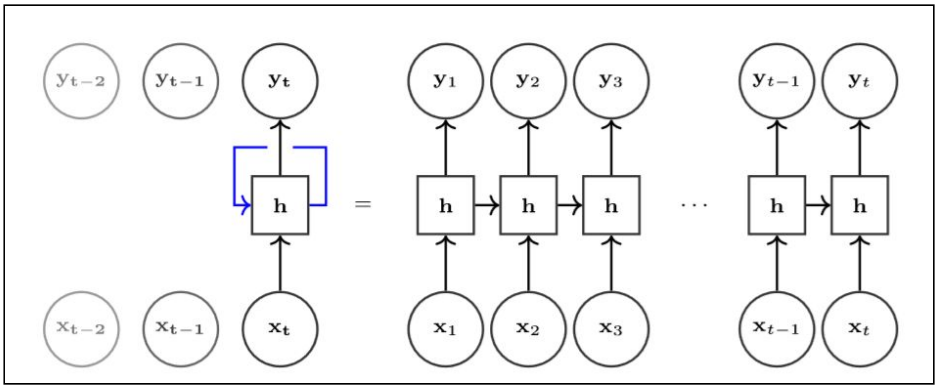

Fig.2.6(a) Recurrent Neural Network [3]

A LSTM model is pictorially presented in Fig.2.6(b) where the three gates can be seen and it can be seen that the previous states of the LSTM are also fed to the current state to determine the current output [29].

Similar to convolutional neural networks the output of a recurrent neural network can be interfaced with fully dense network and or a convolutional neural network [3].

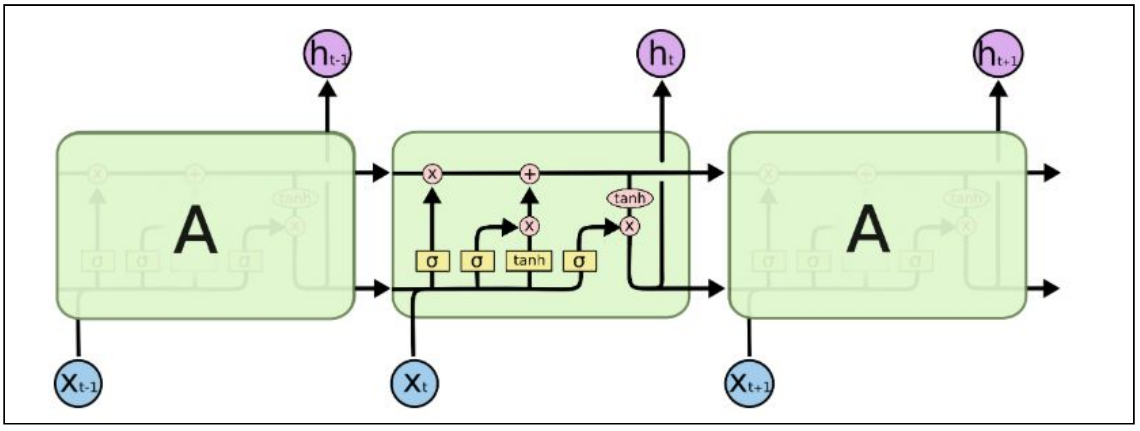

Fig.2.6(b) A LSTM architecture example [29]

2.7 Summary

In this chapter the different machine learning and deep learning models such as fully dense layers, gradient boosted trees,convolutional neural networks which are used to define the architecture for reinforcement learning systems to use.In the following chapter the formal definition of reinforcement learning and the several algorithms will be discussed.

CHAPTER 3

Reinforcement Learning

3.1 Overview

Reinforcement learning is one of three machine learning strategies [1]. It deals with making decisions on a sequential basis. A reinforcement learning system consists of several entities such as an agent which selects an action to carry out to interact with the environment, a critic which converts the signals received from the environment into signals which are interpretable by the learning system. Over time the learning system learns a policy of good-behaviour to maximise its cumulative reward [3].

While a policy dictates how the agent selects an action. The policy is learnt over several iterations. There are three different learning styles through which the policy is decided and defined by the learning system [3].

1. Value based reinforcement learning
2. Policy gradient reinforcement learning
3. Model based reinforcement learning

3.2 Reinforcement Learning Framework

A RL system can be fully modelled by a Markov Decision Process(MDP) [3] with a 5 state tuple (S,A,T,R,γ) where

1. The state space is represented by S
2. The action space is represented by A
3. The transition function $T: S \times A \times S \rightarrow [0,1]$ which represent the set of conditional probabilities between the states
4. The reward function $R: S \times A \times S \rightarrow R$ where R is a continuous set of possible rewards in range $R \in Rmax \rightarrow [0, Rmax]$
5. The discount factor $\gamma \in [0,1]$

While some of the variables were defined in the first chapter, they will be redefined here for the sake of continuity.

1. State of the environment at time step t is s_t where $s_t \in S$
2. Action taken by the agent at time step t is a_t where $a_t \in A$
3. Reward obtained by the agent at time step t is r_t where $r_t \in R$
4. Observation made by the agent at time step t is ω_t where $\omega_t \in \Omega$

The RL system is a fully observable Markov Decision Process and it can further be simplified as the observations made by the agent is equivalent to the state of the environment at time step t Therefore the probability of moving to s_{t+1} is given by the transition function $T(s_t, a_t, s_{t+1})$ [3] .

$$\omega_t = s_t \tag{3.1}$$

The bounded reward function is given by $R(s_t, a_t, s_{t+1})$. Both the transition function and the reward function are both depicted in Fig.3.2(a).

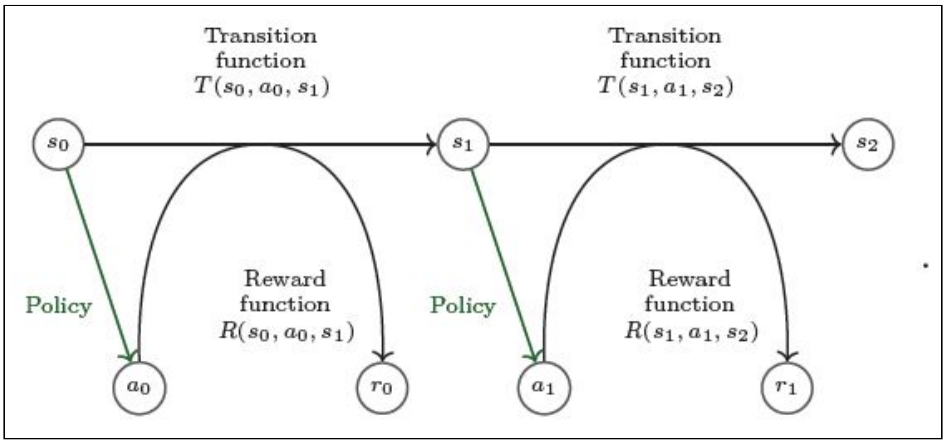

Fig.3.2(a) Markov Decision Process of a RL system [3]

As iterated before the policy is what decides how the agent decides how to interact with the environment. There are two types of policies that can be developed by the RL system which are listed below [3].

1. Deterministic policy $\pi(s) : S \rightarrow A$
2. Stochastic policy $\pi(s, a) : S \times A \rightarrow [0, 1]$ where $\pi(s, a)$ talks about the probability of that the agent selects action a during state s.

Coming back to discussing about the discount factor γ is related to the reward function [3].The discount factor lies in between zero and one . As said before the reinforcement learning system attempts to maximise the cumulative reward via selecting the most optimum actions at a given state due to an efficient policy which dictates the agent, however the time duration over which the RL is given time to learn theoretically is infinity. This has led to problems as it is difficult to simulate models which require infinite time and furthermore the reward function will not converge over time duration *t* [3].

Therefore to convert the infinite time duration to finite time duration and to make the reward function converge, the discount factor [3] was added. If the discount factor is equal to zero, the RL system experiences 'short-sightedness' and only selects actions which have immediate gains on the other hand if the discount factor is equal to one, it selects its actions in such a way that at the final time step *t* it will have the maximum reward [3].

Besides the Markov Decision process state variables which define how a RL system work, there are other components[3] required by the learning system which are listed below :

1. Function Approximator
2. Replay Memory
3. Learning Algorithms
4. Controllers
5. Policies

The function approximator is a network or model which can fit the data between the input-output pairs and as explained before in Chapter 2, the most efficient function approximator is the neural network approach as it can model high-dimensional data and can add extra layers or dimensions to the data with ease and without the exponential increase in data [3]. Furthermore neural networks can be trained iteratively and new

data can be added to the system when required [3]. A general RL scheme with all the components required is depicted in Fig.3.2(b).

The learning algorithm decides how the policy is designed and as can be seen in Fig3.2(b) the learning algorithms have a direct influence on the policies which form [3].The controller block controls the hyperparameters or the parameters which affect the overall learning pace of the system such as the learning rate, batch size, training-validation data split.

The replay memory [35] is an important part of the RL system as whatever experience the system has acquired can be stored here which can be reprocessed at a later time [3]. Replay memory is generally used with an on-line setting where the learning system does not have a priori information about the environment [3].

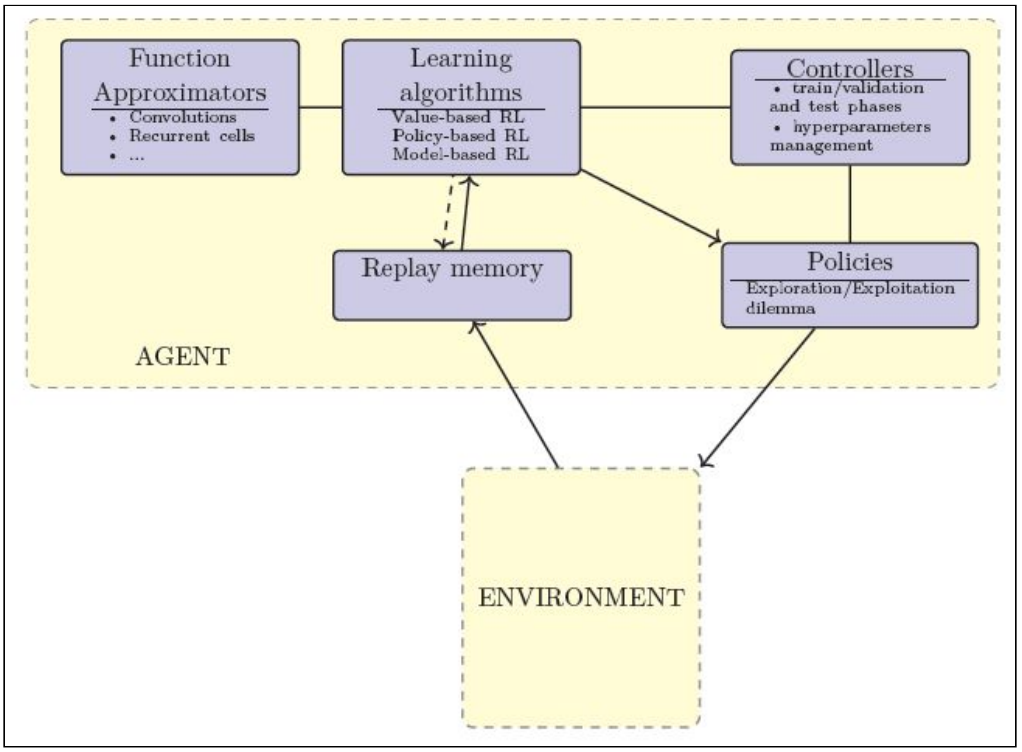

Fig.3.2(b) General RL scheme with all the components [3]

There have been proposals by *Sutton and Barto(2018)* [36] that on-policy methods are used to evaluate and improve the method while off-policy methods are used to improve

policies disjoint from the policies used to generate the data. This is possible as off-policy methods use a trajectory from which could be obtained from another policy *B(s,a)* where $\pi(s,a)$. On-policy methods however introduce a bias into the policy as the trajectory is usually acquired from another policy π which could have negative consequences if care is not taken[36]. Therefore off-policies methods are usually more efficient [3].

3.3 Literature Survey on Value Based Methods

In value based method, importance is given to the value function from which the policy is derived.Importance is given to the Q-value which can be solved recursively using Bellman's equation this is the assumption that the system is a markov decision process.In equation 3.3 *s'* is the next state of the environment s_{t+1} [3].

$$Q^{\tau}(s,a) = E[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} | s_t=s, a_t=a, \pi] \tag{3.2}$$

$$Q^{\tau}(s,a) = \sum_{s' \in S} T(s,a,s')(R(s,a,s') + \gamma\, Q^{\tau}(s', a = (s'))) \tag{3.3}$$

As it can be seen from the Q value equation that the reward is measured at every time step *t* for a given state and action therefore in Q-learning based methods there is a lookup table which has one entry for every state-action pair, besides this there are a few requirements to find the optimal value function. The requirements are listed below:

1. State-action pairs are represented discrete manner
2. All actions are repeatedly samples at every state to ensure that there is enough exploration happening which therefore puts away the transition model.

The first algorithm put forward is the Monte carlo algorithm proposed by *Sutton et.al(1988)* [36] is where the learning system takes a trip to Monte Carlo or in other words, several simulations are done from *s* using a policy π the problem with this algorithm is that it is computationally inefficient, however it is easy to implement as there is no requirement of learning any value function as it directly looks at the returns from the environment [3].

Q-learning proposed by *Watkins et.al (1992)* [30] is based on the Q-value, where it implements a look-up table for every state-action pair and the optimal policy is found by using the Bellman's equation of the Q-function, this algorithm serves the basis for several other important algorithms. It is not possible to use this implementation [3] in higher dimensions and possibly in a continuously environment. The solution is to use another parameter θ that has to be defined it is used to define a few parameters of the Q-values such that the Q-function becomes $Q(s,a;\ \theta)$, which is then called fitted-Q learning [3].

Another brief algorithm proposed by *Rummery et.al (1994)* [37] is an offshoot of Q-learning is called the SARSA(State-Action-Reward-State-Action).In this algorithm the value function is updated with the policy π taken by the system while in Q-learning the value function is updated towards the optimum global policy of Q-values [3].

When fitted Q-learning is applied to neural networks such an algorithm is called a neural-fitted learning proposed by *Reidmiller et.al(2005)* [31] and in this setup the states of the environment can be directly fed to the neural network and the output can be expected to be different for each of the possible state actions [3]. One of the benefits of a neural network implementation is that it can calculate $max\ Q(s',a';\theta_k)$ which is the maximum Q-value for a given state *s'* and *a'* given under a parameter θ_k in a single forward pass making it computationally efficient and to further add to this the parameter θ_k is updated iteratively using an optimisation algorithm such as stochastic gradient descent and the mean square loss is minimised as this ensures that there is no intentional bias being introduced into the system, this helps in a quick convergence of the policy towards the ideal policy. One of the drawbacks is that it can be unstable during training and the error can be unevenly distributed in between the several state-action pairs and the Q-value tends to be an overestimate of the true value [3].

Drawing inspiration from Neural Fitting Q networks, Deep Q networks is configured for an online setting, it has two ways it mitigates the instabilities in the training procedure experienced by Neural Fitting Q networks Deep Q networks proposed by *Mnih et.al(2015)* [32] is another algorithm present which have become very efficient in

video games by inferring what action the agent should select from the raw pixel values it is different from Q-learning as it clips the reward values and changes the parameters. Deep Q networks also require some data preprocessing such as the pixels of the image being normalised to the range of positive one and negative one [3]. Another recommendation is to use the RMSProp optimisation algorithm which was proposed by *Tieleman et.al (2012)* [33] for parameter updates.

Another network architecture was proposed by *Wang et.al (2015)* [38] which was a dueling network architecture where there are two parallel networks. One of the dueling networks estimates the value function while the other network estimates the advantage function. There are three parameters $\theta^1,\theta^2,\theta^3$ all of which contribute to the Q-function.The dueling network when compared to the others is highly stable during training. When compared to Deep Q network is computationally more expensive, however there is greater control over the policy due to there being three parameters [3].

While Deep Q networks tend to be over optimistic and thus lead to an overestimate of the value function this is particularly not a desirable property as this means that there is a presence of an upward bias in the system [3]. The value function should be independent of the stochasticity of the environment, non-stationary sources. The Double Deep Q network was proposed by *Van Hasselt (2016)* [39] where there are two estimates for each variable [3].

After comparing the several value based function algorithms here the Deep Q network based algorithm is the best performing algorithm so far due to its simplicity in nature when compared to other networks such as Dueling networks and to further add to it, it is possible to model nonlinear state action spaces due to the advent of deep learning in reinforcement learning.

3.4 Literature Survey on Policy Gradient Methods

In this section of reinforcement learning algorithms, unlike the value function which are centric around developing the value function to estimate the policy [3]. In policy

gradient learning the learning system is tasked with optimising a performance objective which is usually the cumulative reward received from the environment [3]. In this section the learning depends on the sampling instantiations of the policy parameters and the policy is developed in such a way that the system can acquire maximum returns [3].

This learning algorithm requires a policy evaluation and policy improvement setting. The former attempts to estimate the $Q^{'}(s,a)$ this is done by estimating the gradients which is done by replacing the Q-function with the cumulative return from entire trajectories [3]. In Monte-carlo policy gradient it is estimated from the rollouts on the environment while following a policy π' however, this approach requires an on-policy and there is a high variance in between each rollout therefore several such rollouts are required to be done to get a more accurate estimation of the rollout. This is computationally expensive and an alternate method would be to borrow an idea from value based models such as estimating the value function [3].

To prevent a policy from being deterministic in nature, which would mean that it would stop exploring the environment.This is what is described by the exploitation/exploration dilemma. To prevent such a thing from happening an entropy regularizer is added to the gradient.

The actor-critic method which was proposed by *Konda et.al (2001)* [42] which has two entities. The actor's function is to refer to the policy while it is the critic's function to estimate the value function. Both the actor and the critic can be represented by linear and nonlinear neural networks [3] . The actor by observing the gradients adjusts the policy parameters such as *w* while the critic estimates the value of θ for a current policy such that it follows such a relationship as described below. The critic-actor model is simple in nature, computationally efficient. On the other hand it is unstable during training and the rewards are slowly back propagated [3].

$$Q(s,a;\theta) \simeq Q^{\pi}(s,a) \tag{3.4}$$

Natural policy gradients was another algorithm which was proposed around the same time as the actor-critic model[3] . This algorithm was proposed by *Kakade.S (2001)* [43] where the steepest gradient is used for learning by employing the Fisher information metric, however this is usually practically impossible due the large matrices formed and it is computationally expensive to compute its inverse, therefore natural policy gradient scheme learning is not usually used in reinforcement learning [3].

While a policy gradient learning system should not have a deterministic policy , on the contrary the policy gradient learning system can be applied to deterministic policies such as Neural Fitted Q Iteration with Continuous Action (NFQCA) proposed by *Hafner et.al(2011)* [40] and Deep Deterministic Policy Gradient (DDPG) which was proposed by *Lillicrap et.al(2015)* [41] this is done so as in policy gradient learning the state-action pair do not need to be discrete in nature, which allows for a continuous state-action space which is sometimes the case when it comes to domains such as motor control in robotics where the action the agent selects will be analog in nature and therefore cannot be discrete [3] .

The last algorithm in this subsection is the trust region optimisation where the policy is improved in a controlled manner. It is a constrained optimisation problem where the changes made to the policy are restricted by the KL divergence between the action space distributions. It measures how dissimilar two action spaces distributions are and from this it restricts the policy changes.It was first implemented in TRPO which was proposed by *Schulman et.al (2015)* [44] where the advantage function estimation is used to perform the update[48].

A variant of the TRPO called Proximal Policy Optimisation (PPO) which was proposed by *Schulman et.al (2017)* [45] uses a penalty as the constraint instead of using the KL divergence.

3.5 Literature Survey on Model Based Methods

In the previous two methods the algorithms which were discussed are considered model-free [3]. This is the case as the learning system does not have a priori information about the environment. Here function approximators which are neural networks are used to implement model based networks as they model high dimensional input while possibly being partially observable.

The first algorithm in this section is the lookahead search which a pure-model based method and uses a decision tree implementation [3] . It was proposed by *Brugman (1993)* [46]. The current state is the current node and promising trajectories or policies are given extra attention. The reward is stored in the nodes and the cumulative rewards would be the linear submission all the parent nodes in a selected tree branch.While the lookahead search is simple to implement it can only work on discrete spaces [3].

Trajectory optimisation is another algorithm present is used for continuous state-action spaces [3].In an implementation done by *Deisenroth et.al (2011)* [47] a gaussian process is used to model the probabilistic model which can then later on be used for policy planning and evaluation. One of the drawbacks of this algorithm is that it is not reliable for higher order dimensions [3].

To solve the higher order dimension is by using deep learning. *Wahlstrohm (2015)* [48] implemented an auto-encoder was used where the model was used with a latent space. Following which the policy can be found by solving a finite space horizontal optimal control on the latent space [3].

While comparing the two algorithms the lookahead search has its drawback of being only applicable to discrete spaces it has found more applications in video games and is easy to implement [3].

3.6 Summary

The basics of reinforcement learning were looked at in this chapter following the basic framework for all reinforcement learning was also covered to supplement this the three different methods by which the policy is determined were discussed and it was concluded that Deep Q network , trust region optimisation and look ahead search were one of the best performing algorithms which will be delved into deeper in the next chapter.

CHAPTER 4

Review of the Best Performing RL Algorithms

4.1 Overview

In the previous chapter the basics of reinforcement learning were covered, the framework of reinforcement was covered and the three methods the policy is created by the learning system were discussed. In this chapter the best performing RL algorithms from each of the methods will be discussed more in detail.

4.2 Deep Q Networks

Deep Q networks are a value based method implementation, where the value function is modelled prior to deciding the policy [3]. Furthermore Deep Q networks borrow ideas from another previous algorithm called Neural Fitting Q-learning (NFQ) where the states were directly fed into the neural network [3].

Experience is gathered in a dataset which contains a tuple of the current state, current action which the agent has selected, the reward received from the environment and the next state of the environment [3]. The next state of the environment *s'* is calculated from the transition function $T(s,a,\cdot)$ and the reward r is calculated from the reward function $R(s,a,s')$. In this algorithm, the initial parameter θ_0 is randomly initialised indicating that it starts with a random policy with a constraint that the Q-values should be in the locality of zero. At the k^{th} iteration the parameter θ_k is updated and is defined as below, with the constraint that $a' \in A$ [3].

One of the benefits of a neural network implementation is that it can calculate $max\ Q(s',a';\theta_k)$ in a single forward pass making it computationally efficient and to further add to this the parameter θ_k is updated iteratively using an optimisation algorithm such as stochastic gradient descent and the mean square loss is minimised as this ensures

that there is no intentional bias being introduced into the system, this helps in a quick convergence of the policy towards the ideal policy [3].

$$Y_k^Q = r + \gamma \, max \, Q(s', a'; \theta_k) \tag{4.1}$$

$$Loss = (Q(s, a; \theta_k) - Y_k^Q)^2 \tag{4.2}$$

As explained in the previous chapter the loss is then back propagated towards the input layer and the parameter θ_k are updated in a similar fashion to the fashion to the way the free parameters of the network are updated, which is partially controlled by the learning rate η [3]. Due to the neural networks being a general purpose algorithm, there could be an uneven distribution of error in the state-action space and this could lead to the convergence being slower than predicted and further could make the learning unstable in nature another drawback of using neural networks is that Q-values tend to be a more optimistic prediction an overestimation when compared to the true value this is due to the neural network *max* operator [3].

Deep Q networks is configured for an online setting, it has two ways it mitigates the instabilities in the training procedure experienced by Neural Fitting Q networks. Some of the remedies proposed [32] are as follows:

1. The rewards are clipped to have a maximum value of one and a minimum value of negative one. $r \rightarrow [-1 \ +1]$
2. By introducing a new parameter θ_k^- where the parameter is kept constant for C iteration where $C \in N$, and at the C^{th} iteration $\theta_k^- = \theta_k$.
   This prevents the Y_k^Q from being updated every iteration, this prevents the instabilities present in the training from propagating quickly
3. In an online setting instead of keeping the tuple data for the entirety of the time, the tuples for the previous I iterations are kept where $I \in N$. From the tuples a mini-batch of such tuples are randomly selected this allows the learning system to explore the state-action space and this leads to lower variance in between several mini-batch updates leading to quicker convergence [3].

A block diagram of a Deep Q network has been depicted in Fig.4.1(a)

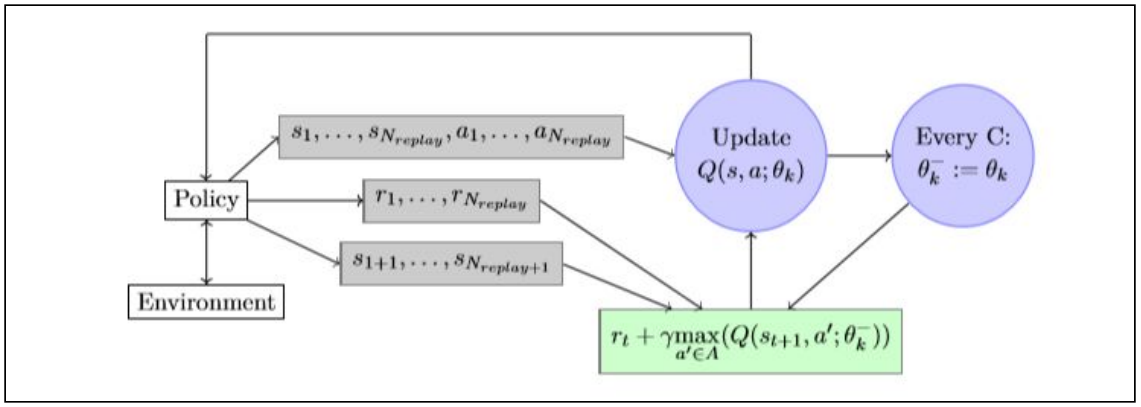

Fig.4.1(a) Block Diagram of a Deep Q network [3]

4.3 Trust Region Policy Optimisation

Trust region policy optimisation is a policy gradient method based learning algorithm that can be used with a continuous state-action space diagram [3]. This is especially useful in the field of robotics when it is used for torque control or in robotic kinematics to name a few [49].

The algorithm is based off the natural policy gradient which uses the Fisher information matrix to calculate the steepest gradient, the drawback of this method is that it is computationally extremely expensive due to the calculation of inverses for several large matrices, following which by adapting the idea and by constraining the optimisation of the policy gradient it has become computationally cheaper.

Trust region optimization(TRPO) uses constrained updates and the estimation of the advantage function to update the parameters of the learning system [3].The advantage function has been described below.

$$V^{\pi}(s_0) = \int_S p^{\pi}(s) \int_A \pi(s,a) R'(s,a) da ds \tag{4.3}$$

$$R'(s,a) = \int_{s'} T(s,a,s') R(s,a,s') \tag{4.4}$$

$$p^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^{t} \cdot Pr(s_t = s | s_0 \pi) \tag{4.5}$$

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{4.6}$$

The first three equations 4.3 and 4.4 describe how to calculate the value function and then from the policy derived from the learning system the advantage function can be calculated from the fourth equation.

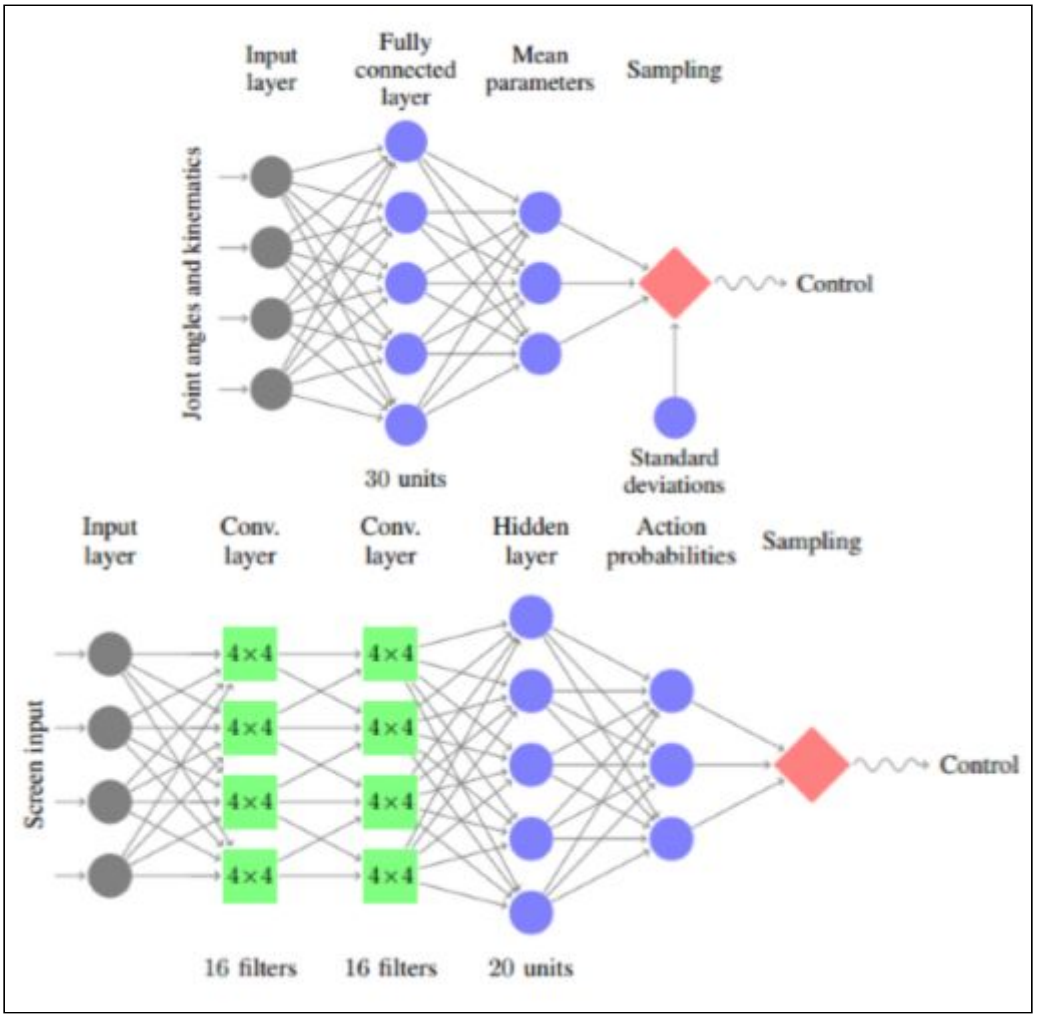

Fig.4.3(a) A neural network depicting a TRPO policy.[49]

Furthermore TRPO uses the KL diverge between the several action spaces distributions, therefore by bounding the size of the policy update, there is a constraint placed on the state distributions therefore leading to a better policy [3]. Furthermore due to KL divergence acting as the loss function, it prevents large-scale changes from

happening to the policy distribution and therefore helps in learning system converging much quicker [49]. A network which represents a TRPO policy has been shown in Fig.4.3(a).

4.4 Lookahead Search

The lookahead search is a model based method where the learning system has a priori information about the environment [3]. It is a fully observable markov decision process which builds a decision tree in which the current state is the root node. The rewards are stored at the node and attention is given to the trajectories which look promising. One of the problems experienced in this model is that a number of trajectories have to be sampled so that the system can sufficiently explore the environment [3].

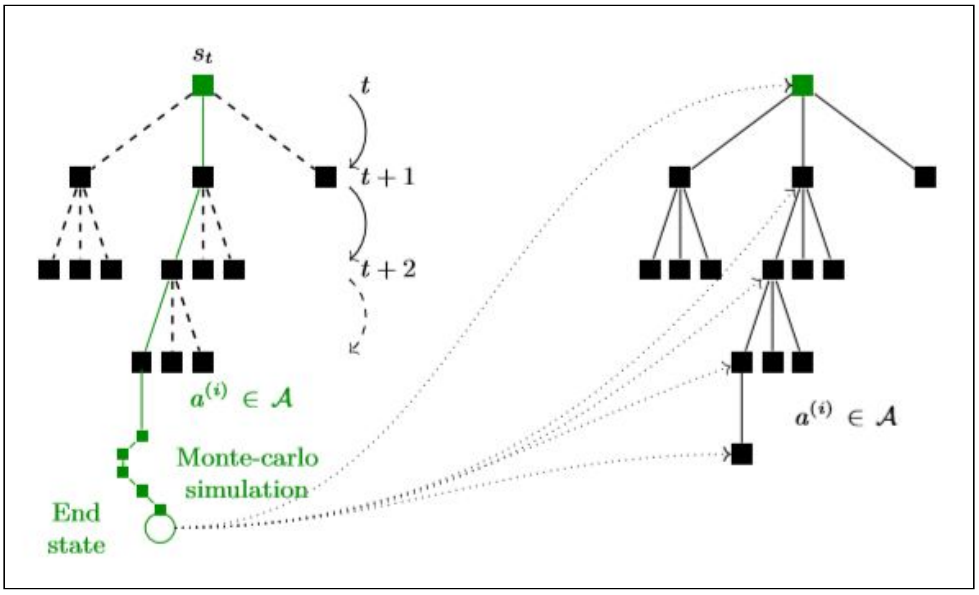

Fig.4.4(a) A MCTS tree [3]

A special variant of the lookahead search algorithm is the Monte-Carlo Tree Search(MCTS) which has been used extensively in video games [3]. In this algorithm several trajectories are sampled from the current state until a terminal condition has occurred such as the maximum depth of the tree has already been achieved [3].

Once the terminal condition has arrived, the MCTS traverses the tree and then recommends which action the agent should select by viewing the reward stored at the

node and the cumulative reward by taking the sum of all the parent node rewards from the terminal node. A MCTS diagram has been depicted in Fig.4.4(a).

4.5 Summary

In this chapter the best performing algorithms for each one of the policy learning methods were looked at the first one was the Deep Q Network model , the second one was the Trust Region Policy Optimisation and last but not least the lookahead search. The first and last model are used either in tandem with each other or as stand alone networks when training RL in video games. The second one is suited for continuous state-action space problems and is used in areas dealing with robotics. In the next chapter the applications of reinforcement learning will be dealt with.

CHAPTER 5

Applications of Reinforcement Learning

5.1 Overview

Reinforcement learning has been applied to several domains such as the financial sector and the video games sector to name a few. Reinforcement Learning especially Deep RL algorithms have performed exceptionally well in real life problems with high-dimensional inputs. In this chapter the focus will be on some of the domains RL has been applied to moreover their implementation will also be touched upon.

5.2 Financial Sector

One of the more recent implementations that used a Deep RL implementation was proposed by *Deng et.al (2016)* [50], which was the first such algorithm to be implemented for real time trading systems. Usually for predicting stock prices several hand crafted features are extracted from the data and which are then fed into the learning system .

However one of the main issues that lie around this is the concept of generality as the financial market is not deterministic in nature to start with, it is stochastic and has a noise-like signal. Features selected for one problem may not work in another, for example by taking a moving average [50] of the data is a good indicator the trend however this would not be a good indicator of a mean-revision market.

A Deep learning approach with a fuzzy logic interface [50] was used where the RNN's were used in the deep learning section. RNN or its variants such as LSTM work well with predicting sequential data and hence was used in this implementation. The activation function used was tanh and to increase their accuracy the input end of the neural network was interfaced with a fuzzy logic system [51], which used a gaussian membership function to map the input financial trading data to fuzzy output.

Fuzzy logic which was first proposed by *A.Zadeh* (1968) [51], where the real world data is converted into fuzzy values which are then used by fuzzy systems to make a decision and in this case, fuzzy logic was used due to the uncertainty revolving around the financial market and its sensitivity to global politics , news and even rumours give the financial market its stochastic quality [50].

To mitigate this impact of randomness in the stock market the input was interfaced with a fuzzy logic system which then converted it into fuzzy values, these fuzzy values are then fed into the neural network and the output of the neural network is the ultimate reward of profit of the trading market [50]. The block diagram of the system is shown in Fig.5.2(a) where W represents the predicted value.

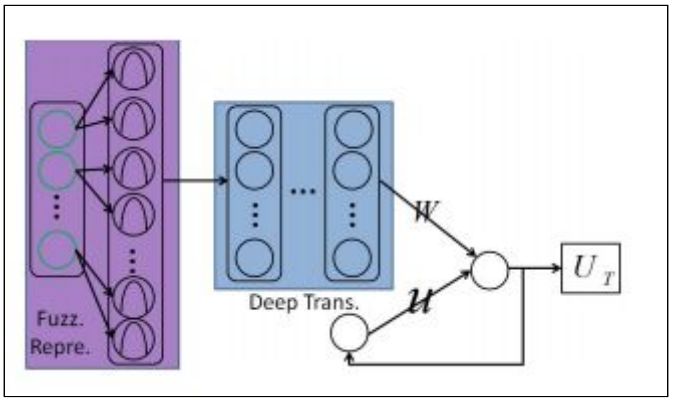

Fig.5.2(a) Block diagram of a Deep RL system used in the financial market [50]

The reinforcement learning method which was used was a value based method with an online setting as there is no a priori information about the environment to further add to this, the learning system was self-taught and no explicit signalling was given to it [50].The value function was built upon maximising the profit statement of a single share of an asset in a real time financial scenario.

5.3 Video Games Sector

With the advent of powerful graphical processing units, video games have become more complex and have started to mimic real life events and in the trend to design a

general artificially intelligent system reinforcement learning has been applied to video games in hopes of studying how a RL agent would interact with the real world. By applying a Deep Q Network [52] which is a value based method, reinforcement learning has attained superhuman performance in a series of Atari 2600 games which is a benchmark in reinforcement learning [52].

The data was the pixels from the screen of the game which had dimensions 210x160, several preprocessing actions were done on the screen such as extracting the Y channel from the game screen and then rescaling it to 84x84 [52] .Encoding a single frame with the maximum value of the present screen and previous frame, this was done to reduce the amount of flickering as certain gaming characters would appear only in even frames and others in odd frames. The output of the DQN network was the action the agent should select to maximise its reward[52].

The optimiser used was RMSProp, with ReLU being used as the activation function used after each convolutional layer. To maintain uniformity across several of the Atari 2600 games the same network was used and the rewards were clipped to be between -1 and +1 [52]. Since in many of the games there are a fixed number of lives given to the player it has been modelled as a terminal state when given to the learning system and in this implementation the number of lives left is also fed to the learning system [52]. The network architecture used is shown in Fig.5.2(a).

In Fig.5.2(a) the snake like pattern seen at the beginning is the kernel being passed over the entire image and the resulting vector being passed on further.

The first convolutional layer has a kernel size of 8 with 32 filters with a stride of 1, being applied to the input with image of size 84x84x4 [52]. The output is then passed through the ReLU function and the second convolutional layer has 64 filters with kernel size of 4 and stride 2 which is then passed through the same activation function and the third layer with 64 filters with kernel size 3 and stride 1 passed through the same activation function. The following output is then sent to a fully dense layer of 512 units which is passed through a ReLU activation function and the output is passed

through another fully dense layer whose outputs range from 4 to 18 based on the number of controls in that game [52].

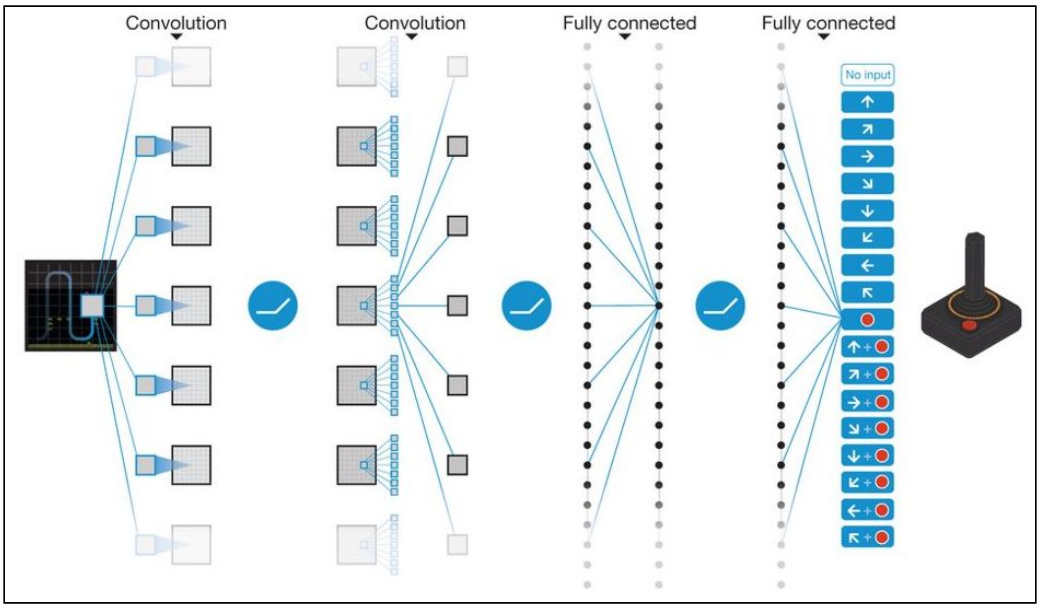

Fig.5.2(a) Deep Q Network used for playing Atari 2600 games [52].

5.4 Robotics Sector

Robotics are an area where reinforcement learning has successfully applied to, where it generally uses a policy gradient approach due to there being several continuous variables such as motor torque and angle of rotation. What has become of more interest is visual motor control where the input is either still images or a series of images.

Making a reinforcement learning system learn hand-eye coordination as discussed by *Levine et.al (2018)* [53], where the input were still images of a desk which were cluttered with several objects and a robotic arm was tasked with picking up an object in real time, to add control over what object should be picked up, the selected object would be coloured green for the system to recognise and pick up [53].

The grasping model was broken down into two sub models the first network was a large CNN based model which would take in monocular images [53]. The output of CNN would be a series of probability of whether the grasping robot would be able to successfully execute such an action [53].

The second network was a servoing function which would convert the predicted probability in servo commands to give to the robot which was continuously controlled.

It was trained on 80,000 labelled images and it was configured for on-line learning as it gives the system a chance to explore the environment. A total of 17 convolutional layers are used in the first half of the network to predict the probability of a successful grasp and it used a policy gradient method [53].

The CNN takes two images which are stacked on top of each other both are of size 512x512x3, they are randomly cropped to 472x472x3 this is done to make the system invariant to translation[53].

The first image is the current image and the previous image is the original image without the gripper robot [53]. This image is passed through a convolutional layer with kernel size 6 and with stride 2, following which it passes through a max pooling layer with a kernel size of 3 and following this it passes through 5 convolutional layers which contain batch normalisation layers[54], where the kernel size is of size 5 and the output is passed through a max pooling layer with kernel size 3 [53].

Briefly talking about batch normalisation [54] , it is a layer added usually in between the output of a convolution layer and the input of another , it is useful as it reduces the training time and reduces the covariant shift of the data in between layers.

At the output of the 5^{th} layer the servoing function is passed through, to match the vectors it is passed through a fully dense layer which is then reshaped to match the dimensions of the 5^{th} convolutional layer after max pooling [53]. Following this it is passed through several convolutional layers and pooling layers which is then passed through a fully dense layer which outputs a series of probability values which is then passed on to the second part of the model [53]. The CNN network for the first half of the network is shown in Fig.5.4(a).

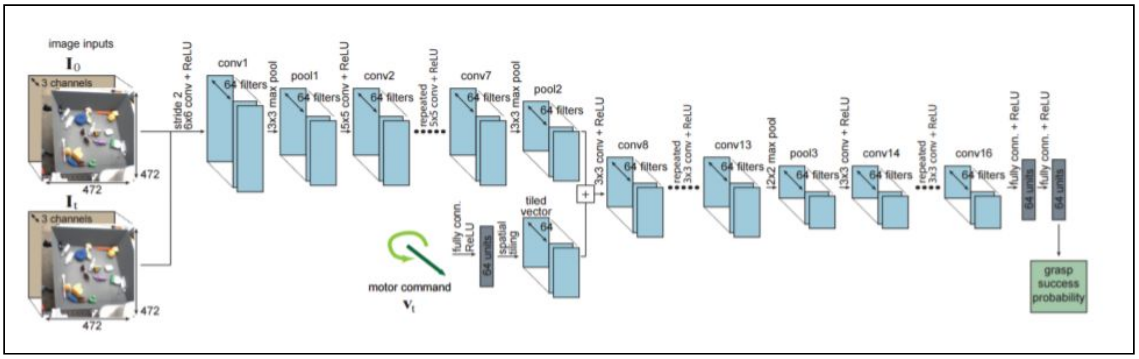

Fig.5.4(a) CNN network used for the grasping robot [53]

5.5 Self Driving Cars

The idea behind self driving cars is for the car to sense its surroundings and the car will automatically make a decision such as go straight, take a left turn or go straight and faster. Most of the research done in this field is supervised in nature [55]. Where several such scenarios are fed to the learning system and the output will be an action and the error from the difference between the predicted and desired is back propagated and this is how a supervised autonomous system learns. The problem with such a learning strategy is an extremely large set of labelled data ,with human effort is required to train such a system [55].

Therefore to reduce human effort and to train the system on fewer images a reinforcement learning system is proposed by *Pan et.al (2017)* [55] where a reinforcement learning system using the actor-critic method which is a policy gradient method is used [55]. In RL the agent interacts with the environment on a trial and error basis which is not feasible as a car can't be left alone on a road to learn will end up in the car damaging itself, the surrounding cars and the environment. Therefore the RL system is setup in a simulator to learn [55].

There are two networks present in this, the first one takes virtual images from the simulator and by applying image to image translation outputs a segmented image which is then passed through another network which generates realistic images which are

different from the simulator images, and are still realistic. This ensures that the images used by the simulator are realistic [55]. The image to image translation is a generative adversarial network implementation (GAN) [11], the network implementation can be seen in Fig.5.5(a).

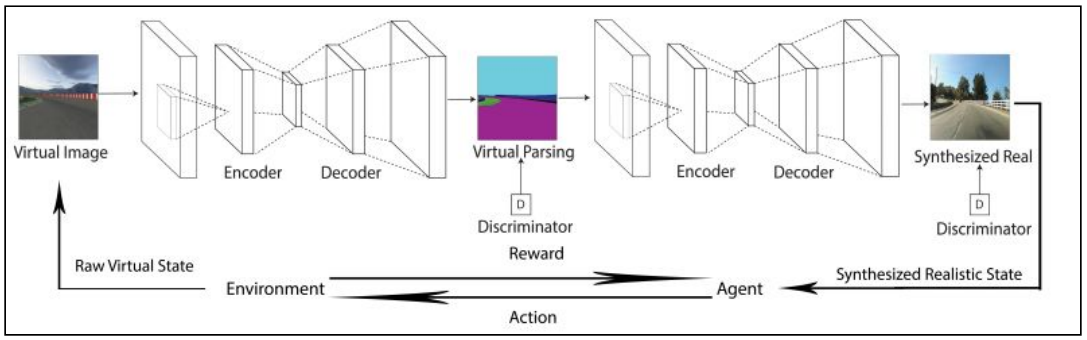

Fig.5.5(a) GAN model used for image to image translation [55]

As seen from the Fig.5.5(a) the agent interacts with the synthesised images and selects an action to interact with the environment which is interacting with the real life environment and thus the RL system is trained to autonomously handle the car as if it was in a real life situation [55].

Both the virtual image to segmented image and the segmented image to a realistic image using an encoder decoder architecture [56] which uses a U-net architecture with skip connections. The input image is of size 256x256x3 which is then passed through several convolutional layers with kernel size of 4 and stride 2 using batch normalization and a leaky ReLU activation function with a slope of 0.2. In the decoder architecture the same convolutional layer setup is used but the activation function is ReLU and the last stage uses a tanh activation function. The output size is same as that of the input size [55].

The RL network used for interacting with the environment takes a synthesised realistic image and outputs an action such as “go straight”, “go left” , “go right” and their combinations with “acceleration” and “brake”[55]. There are a total of 9 such actions possible. The network consists of 4 convolutional layers which use ReLU as an

activation function and is optimised on RMSprop. 4 consecutive frames are taken in and it outputs an action for the agent to select. The RL network architecture is shown in Fig.5.5(b).

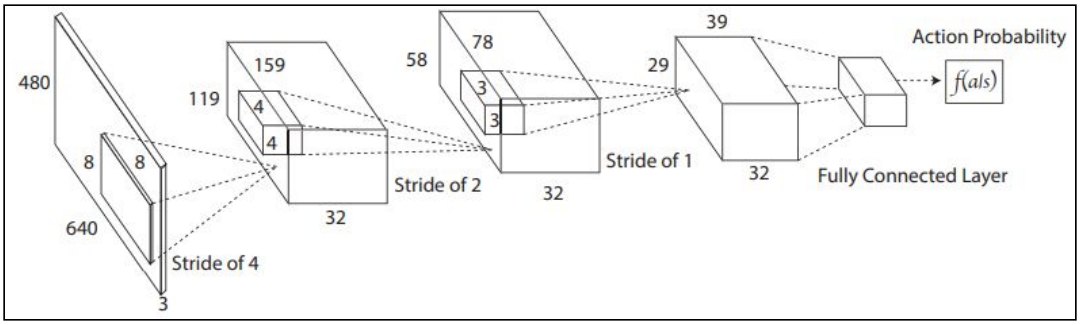

Fig.5.5(b) RL network used to interact with the environment [55].

5.6 Summary

In this chapter a few applications of RL in fields such as in stock trading where it can be seen that by interfacing fuzzy logic with a RNN the predictions are more accurate. In video games on the Atari 2600 games benchmark the DQN was able to achieve superhuman performance by using a CNN network which outputs an action for the agent to select, the first two applications were implementations using a value based method. In the robotics sector to control a grasping hand a policy gradient method was used with a large CNN network which outputs the probability of a successful grip which is passed onto a second network which translates it to a servoing function. Lastly in self driving cars a RL system using an actor - critic method which is a policy gradient method is used where the input are 4 consecutive frames and the output is an action such as “go left and brake”. For this to happen the virtual images are converted to realistic images for the RL system to use.

CHAPTER 6

Summary and Future Scope

In this manuscript the first chapter covered the basics such as the three types of learning mechanism and briefly talked about neural networks and mentioned about the difference between machine learning and deep learning. In the following chapter the basics of machine learning and deep learning were covered starting from regression and ending with recurrent neural networks.

In the subsequent two chapters the focus shifted from the basics onto more detailed discussions such as in third chapter where the basics of reinforcement learning were talked about with the introduction of the framework. In the same chapter a brief literature survey was done on the three main learning methods present in reinforcement learning. In the fourth chapter a review of the best algorithms from each one of the methods were talked about in detail with their mathematical equations.

Subsequently in the last chapter, now having knowledge of what components are needed to implement a reinforcement learning system with knowledge of which reinforcement learning algorithm should be applied to the learning system to get satisfactorily results few of the applications where reinforcement learning has been successfully applied to were covered.

Reinforcement learning has shown to work brilliantly in fields such as video games, and robotics while the performance of reinforcement learning in self-driving cars is better than the baseline model, it is still comparatively poorer than the supervised version. In the future improvements can be made in areas such as self-driving cars,financial trading by using a mix of value based or policy based with a model based definition as this would add constraints to the environment.

References

[1] Haykin, Simon O. *Neural Networks and Learning Machines*. Pearson Higher Ed, 2011.

[2] Becker, Suzanna. "Unsupervised learning procedures for neural networks." *International Journal of Neural Systems* vol.2, no. 01 no.02 ,pp. 17-33, 1991.

[3] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare and Joelle Pineau (2018), "An Introduction to Deep Reinforcement Learning", Foundations and Trends® in Machine Learning: Vol. 11: No. 3-4, pp 219-354. http://dx.doi.org/10.1561/2200000071

[4] Scholkopf, Bernhard, and Alexander J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[5] Ke, Guolin, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. "Lightgbm: A highly efficient gradient boosting decision tree." *In Proceedings of the 30th Conference on Advances in Neural Information Processing Systems*, 2017, pp. 3146-3154.

[6] Saffari, Amir, Christian Leistner, Jakob Santner, Martin Godec, and Horst Bischof. "On-line random forests." In *Proceedings of Ieee 12th international conference on computer vision workshops, iccv workshops*, IEEE, 2009,pp. 1393-1400.

[7] Zhang, Yuwen, X. Ding, Y. Liu, and P. J. Griffin. "An artificial neural network approach to transformer fault diagnosis." *IEEE Transactions on Power Delivery*,vol.11, no. 4, pp:1836-1841,1996.

[8] Seber, George AF, and Alan J. Lee. *Linear regression analysis*. vol.329. John Wiley & Sons, 2012.

[9] Gallant. "Nonlinear regression." *The American Statistician*, vol.29, no. 2 (1975): 73-81.

[10] Kleinbaum, David G., K. Dietz, M. Gail, Mitchel Klein, and Mitchell Klein. *Logistic regression*. New York: Springer-Verlag, 2002.

[11] Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets." In *Proceedings of the 25th conference on Advances in neural information processing systems*, 2014, pp. 2672-2680.

[12] You, Quanzeng, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo. "Image captioning with semantic attention." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4651-4659.

[13] Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." In *Proceedings of the 27th international conference on machine learning (ICML-10)*,2010 pp. 807-814.

[14] Zhang, Xiaohu, Yuexian Zou, and Wei Shi. "Dilated convolution neural network with LeakyReLU for environmental sound classification." In *2017 22nd International Conference on Digital Signal Processing (DSP)*, pp. 1-5. IEEE, 2017.

[15] Han, Wei, Cheong-Fat Chan, Chiu-Sing Choy, and Kong-Pang Pun. "An efficient MFCC extraction method in speech recognition." In *Proceedings of 2006 IEEE international symposium on circuits and systems*, 2006, pp. 4-4..

[16] Gu, Shenshen, Lu Ding, Yue Yang, and Xinyi Chen. "A new deep learning method based on AlexNet model and SSD model for tennis ball recognition." In *2017 IEEE 10th Int Workshop on Computational Intelligence and Applications (IWCIA)*, 2017, pp. 159-164.

[17] Thrun, Sebastian B. "The role of exploration in learning control." (1992).

[18] Rouet-Leduc, B., Hulbert, C., Lubbers, N., Barros, K., Humphreys, C. J., and Johnson, P. A. Machine learning predicts laboratory earthquakes. In *Geophysical Research Letters*, vol. 44 , 2017. pp.9276–9282. https://doi.org/10.1002/2017GL074677

[19] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In *Proceedings of the 11th conference on Advances in neural information processing systems*,2012 pp. 1097-1105.

[20] Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In *2009 IEEE conference on computer vision and pattern recognition*,2009, pp. 248-255.

[21] Statistics: Linear Regression," *Desmos Graphing Calculator*. [Online]. Available: https://www.desmos.com/calculator/jwquvmikhr. [Accessed: 19-Oct-2019].

[22] Quinlan, J. Ross. "Decision trees and decision-making." *IEEE Transactions on Systems, Man, and Cybernetics*,vol.20, no. 2 (1990),pp. 339-346.

[23] Breiman, Leo. "Random forests." *Machine learning*, vol.45,no. 1, pp.5-32, 2001.

[24] Oza, Nikunj Chandrakant, and Stuart Russell. *Online ensemble learning*. University of California, Berkeley, 2001.

[25] Rumelhart D.E., Hinton G.E. and Williams M.J."Learning Internal Representations by Backpropagation of Errors",Nature vol.323, pp. 533-536, 1986.

[26] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).

[27] Robbins, Herbert, and David Siegmund. "A convergence theorem for non negative almost supermartingales and some applications." In *Optimizing methods in statistics*, Academic Press, 1971,pp. 233-257.

[28] LeCun, Yann, Corinna Cortes, and Christopher JC Burges. "The MNIST database of handwritten digits, 1998." *URL http://yann. lecun. com/exdb/mnist* 10 (1998): 34.

[29] Hochreiter, Sepp, and Jürgen Schmidhuber. "LSTM can solve hard long time lag problems." In *Proceedings of the 3rd conference on Advances in neural information processing systems*, 1997,pp. 473-479.

[30] Watkins, Christopher JCH, and Peter Dayan. "Q-learning." *Machine learning* vol.8, no. 3-4,pp.279-292,1992.

[31] Riedmiller, Martin, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. "Learning by playing-solving sparse reward tasks from scratch." *arXiv preprint arXiv:1802.10567* (2018).

[32] Mnih, Volodymyr, Adria Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous methods for deep reinforcement learning." In *International conference on machine learning(ICML)*,2016, pp. 1928-1937.

[33] Tieleman, H. 2012. “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude”. COURSERA: Neural Networks for Machine Learning.

[34] Lin, L.-J. 1992. “Self-improving reactive agents based on reinforcement learning, planning and teaching”. *Machine learning*,vol.8,no.(3-4): pp.293–321. Lipton, Z. C., J. Gao, L. Li, X. Li, F. Ahmed, and L. Deng. 2016.

[35] Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[36] Sutton, Richard S. "Learning to predict by the methods of temporal differences." *Machine learning,*vol.3, no. 1,pp.9-44,1988

[37] Rummery, Gavin A., and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. Cambridge, England: University of Cambridge, Department of Engineering, 1994.

[38] Wang, Ziyu, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. "Dueling network architectures for deep reinforcement learning." *arXiv preprint arXiv:1511.06581* (2015).

[39] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[40] Hafner, Roland, and Martin Riedmiller. "Reinforcement learning in feedback control." *Machine learning*,vol.84, no. 1-2,pp.137-169,2011

[41] Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971* (2015).

[42] Konda, Vijay R., and John N. Tsitsiklis. "Actor-critic algorithms." In *Proceedings of the 11th conference on Advances in Neural information processing systems*, 2000,pp. 1008-1014.

[43] Kakade, S. 2001. “A Natural Policy Gradient”. In *Proceedings of the 16th International conference on Advances in Neural Information Processing Systems,Natural and Synthetic*,Vancouver, British Columbia, Canada,December 3-8, 2001, pp.1531–1538.

[44] Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization." In *Proceedings of the 32nd International Conference on Machine Learning(ICML)*, 2015,pp. 1889-1897.

[45] Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms." *arXiv preprint arXiv:1707.06347* (2017).

[46] Brügmann, Bernd. *Monte carlo go*. Vol. 44. Syracuse, NY: Technical report, Physics Department, Syracuse University, 1993.

[47] Deisenroth, Marc, and Carl E. Rasmussen. "PILCO: A model-based and data-efficient approach to policy search." In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011, pp. 465-472.

[48] Wahlström, Niklas, Thomas B. Schön, and Marc Peter Deisenroth. "From pixels to torques: Policy learning with deep dynamical models." *arXiv preprint arXiv:1502.02251* (2015).

[49] Amarjyoti, Smruti. "Deep reinforcement learning for robotic manipulation-the state of the art." *arXiv preprint arXiv:1701.08878* (2017).

[50] Deng, Yue, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. "Deep direct reinforcement learning for financial signal representation and trading." *IEEE transactions on neural networks and learning systems,*vol.28, no. 3,pp.653-664, 2016.

[51] Zadeh, Lotfi Asker. "Fuzzy logic." *Computer,*vol.21, no. 4,pp.83-93,1988.

[52] Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves et al. "Human-level control through deep reinforcement learning." *Nature,*vol.518, no. 7540,pp.529-529,2015

[53] Levine, Sergey, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection." *The International Journal of Robotics Research* vol.37, no. 4-5, pp.421-436, 2018.

[54] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *arXiv preprint arXiv:1502.03167* (2015).

[55] Pan, Xinlei, Yurong You, Ziyan Wang, and Cewu Lu. "Virtual to real reinforcement learning for autonomous driving." *arXiv preprint arXiv:1704.03952* (2017).

[56] Isola, Phillip, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. "Image-to-image translation with conditional adversarial networks." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017,pp. 1125-1134.

Acronyms

Acronyms	Description
ANN	Artificial Neural Network
CNN	Convolutional Neural Networks
DDPG	Deep Deterministic Policy Gradient
DL	Deep Learning
KL	Kullback-Leibler
Leaky ReLU	Leaky Rectified Linear Units
LSTM	Long Short Term Memory
MCTS	Monte-Carlo Tree Search
MDP	Markov Decision Process
MFCC	Mel Frequency Cepstral Coefficients
ML	Machine Learning
NFQ	Neural Fitting Q-learning
NFQCA	Neural Fitted Q Iteration with Continuous Action
PPO	Proximal Policy Optimisation
ReLU	Rectified Linear Units
RL	Reinforcement Learning
RMSProp	Root Mean Square Propagation

RNN	Recurrent Neural Network
SARSA	State-Action-Reward-State-Action
TRPO	Trust Region Policy Optimisation