
SRH HOCHSCHULE HEIDELBERG

MASTER THESIS

# Optimis: A framework for optimal control of a message in social media using Deep Reinforcement Learning

*Author:*
Deborah Zenobia Rachael Menezes , 11012497

*Supervisors:*
Prof. Dr. Ajinkya Prabhune
Prof. Dr. -Ing Gerd Moeckel

*A thesis submitted in fulfillment of the requirements for the degree of Masters in Big Data and Business Analytics*

*in the*

Department of Information, Media and Design

March 23, 2021

# Declaration of Authorship

I, Deborah Zenobia Rachael Menezes, declare that this thesis titled, "Optimis: A framework for optimal control of a message in social media using Deep Reinforcement Learning" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

# Affidavit

Herewith I declare:

- that I have composed the chapters for the Master Thesis for which I am named as the author independently,
- that I did not use any other sources or additives than the ones specified,
- that I did not submit this work at any other examination procedure.

Heidelberg:
_____(date)

Signature:
_____

# *Abstract*

Humans communicate with a dynamic world in a broad range of applications through asynchronous stochastic discrete events through continuous-time, also known as temporal point processes. Examples of temporal point processes range from social media posts, trading signals to student activity on learning platforms. There is a significant amount of potential in modeling decision-making behavior based on point processes. Reinforcement learning is an ideal framework for constructing sequential decision-making models. However, research on sequential decision-making on temporal point processes is quite scarce. In this thesis, the Optimis framework is developed with the objective of 1) enabling social media users to find the optimal broadcasting time in online social environments and 2) assisting researchers in building reinforcement-learning algorithms on discrete temporal settings that encompass a broader set of use cases. Optimis achieves this by implementing 1) an RL-based agent that makes optimal actions in point process environments and 2) a customizable environment that supports a wide array of discrete time-domain applications. For validation on a real-world scenario, the Optimis framework is benchmarked on a Twitter dataset against a state-of-the-art parametric model. In order to showcase the generalizability of Optimis, a synthetic dataset is chosen for evaluation. On both datasets, Optimis achieves significantly higher results than the chosen parametric model. The Optimis framework's environment enables scalability to several discrete temporal applications such as viral marketing, spaced repetition and quantitative trading and fake news mitigation.

**Keywords:** *Deep reinforcement learning, Hawkes process, Influence maximization, Temporal point processes, Viral marketing, Social media post control, Smart broadcasting, fake news*

# *Zusammenfassung*

Menschen kommunizieren mit einer dynamischen Welt in einem breiten Spektrum von Anwendungen durch asynchrone stochastische diskrete Ereignisse durch zeitkontinuierliche, auch als zeitliche Punktprozesse bezeichnete Prozesse. Beispiele für zeitliche Punktprozesse reichen von Social-Media-Posts über Handelssignale bis hin zu Schüleraktivitäten auf Lernplattformen. Die Modellierung des Entscheidungsverhaltens auf der Grundlage von Punktprozessen birgt ein erhebliches Potenzial. Reinforcement Learning ist ein idealer Rahmen für die Erstellung sequentieller Entscheidungsmodelle. Die Forschung zur sequentiellen Entscheidungsfindung über zeitliche Punktprozesse ist jedoch recht selten. In dieser Arbeit wird das Optimis-Framework mit dem Ziel entwickelt, 1) Social-Media-Nutzern die Möglichkeit zu geben, die optimale Sendezeit in sozialen Online-Umgebungen zu finden, und 2) Forschern dabei zu helfen, Algorithmen für das Verstärkungslernen auf diskreten zeitlichen Einstellungen zu erstellen, die einen breiteren Satz von umfassen Anwendungsfälle. Optimis erreicht dies durch die Implementierung von 1) einem RL-basierten Agenten, der optimale Aktionen in Punktprozessumgebungen ausführt, und 2) einer anpassbaren Umgebung, die eine Vielzahl diskreter Zeitbereichsanwendungen unterstützt. Zur Validierung in einem realen Szenario wird das Optimis-Framework anhand eines Twitter-Datensatzes mit einem hochmodernen parametrischen Modell verglichen. Um die Generalisierbarkeit von Optimis zu demonstrieren, wird ein synthetischer Datensatz zur Auswertung ausgewählt. In beiden Datensätzen erzielt Optimis signifikant höhere Ergebnisse als das ausgewählte parametrische Modell. Die Umgebung des Optimis-Frameworks ermöglicht die Skalierbarkeit für verschiedene diskrete zeitliche Anwendungen wie virales Marketing, räumliche Wiederholung und quantitativen Handel sowie die Reduzierung gefälschter Nachrichten.

# *Acknowledgements*

First and foremost, I want to thank my research supervisors, Prof. Dr. Ajinkya Prabhune and Prof. Dr.-Ing Gerd Moeckel. Without their assistance and dedicated involvement throughout the process and development, this thesis would have never been accomplished. I would like to thank the faculty of SRH for all the unconditional support during my Master's Thesis.

I would like to show my utmost gratitude to Prof. Dr. Ajinkya Prabhune, who supervised me and allowed me to do this thesis under his guidance. As well as for his immense knowledge to build the framework to support influence maximization. In addition to the constant push towards research and development, which gave me a broader view of how the research community works and how it has helped the world see some of the best implementations in AI. I want to express my heartfelt gratitude to all authors in the research community for sharing their knowledge and findings.

For the friendships that I made at the university, they have listened to me as well as tolerated me over the past two years. For getting through my dissertations required more than academic support. John Joy Kurian, Avinash Ronanki and Sandeep Krishnaprasad have been unwavering in their personal and professional support during my time at the University. I gained knowledge in several technologies and worked in such a committed and dedicated team. And also to Mr. Ashish Chouhan for giving constant feedback and guidance during the case study project.

Most importantly, I thank my family, it would be an understatement to say that, we have experienced some ups and downs in the past couple of years. They never let me quit and has been a constant support and encouragement throughout my years of study.

# List of Abbreviations

| | |
|---|---|
| **MTTP** | **M**arked **T**emporal **P**oint **P**rocess |
| **RMTPP** | **R**ecurrent **M**marked **T**emporal **P**oint **P**rocess |
| **SDE** | **S**tochastic **D**ifferential **E**quations |
| **MDP** | **M**arkov **D**ecision **P**rocess |
| **RL** | **R**einforcement **L**earning |
| **ML** | **M**achine **L**earning |
| **DRL** | **D**eep **R**einforcement **L**earning |
| **HP** | **H**awkes **P**rocess |
| **DL** | **D**eep **L**earning |
| **SAHP** | **S**elf - **E**xciting **H**awkes **P**rocess |
| **NHP** | **N**eural **H**awkes **P**rocess |
| **CIP** | **C**onditional **I**ntensity **F**unction |
| **RNN** | **R**ecuurent **N**eural **N**etwork |
| **GRU** | **G**ated **R**ecuurent **U**units |
| **LSTM** | **L**ong **S**hort **T**erm **M**emory |
| **THP** | **T**ransformer **H**awkes **P**rocess |
| **NLP** | **N**atural **L**anguage **P**rocessing |
| **AI** | **A**rtificial **I**ntelligence |
| **TCN** | **T**emporal **C**onvolutional **N**etwork |
| **DQN** | **D**eep **Q** **N**etwork |
| **SGD** | **S**tochastic **P**olicy **G**radient |
| **DPG** | **D**eterministic **P**olicy **G**radient |
| **DDPG** | **D**eep **D**eterministic **P**olicy **G**radient |
| **TRPO** | **T**rust **R**egion **P**olicy **O**ptimization |
| **ACKTR** | **A**ctor-**C**ritic using **K**ronecker-**F**actored **T**rust **R**egion |
| **DPG** | **D**eterministic **P**olicy **G**radient |
| **PPO** | **P**roximal **P**olicy **O**ptimization |
| **NAF** | **N**ormalized **A**dvantage **F**unction |
| **BFTT** | **B**ack **P**ropogation **T**hrough **T**ime |
| **ARIMA** | **A**uto**R**egressive **I**ntegrated **M**oving **A**verage |
| **AR** | **A**ugmented **R**eality |
| **TD** | **T**emporal **D**ifference |
| **SARSA** | **S**tate-**A**ction-**R**eward-**S**tate-**A**ction |

# Contents

**Declaration of Authorship** i

**Affidavit** ii

**Abstract** iii

**Zusammenfassung** iv

**Acknowledgements** v

**1 Introduction** 1
- 1.1 Objective . . . 1
- 1.2 Motivation . . . 1
- 1.3 High level problem definition . . . 2
- 1.4 Background . . . 5
  - 1.4.1 Discrete vs continuous time series . . . 5
  - 1.4.2 Marked temporal point processes . . . 5
  - 1.4.3 A strong parametrization . . . 6
  - 1.4.4 Non-parametrization of the problem . . . 7
  - 1.4.5 Key contributions of Optimis . . . 8
  - 1.4.6 Thesis outline . . . 9

**2 Literature Review** 10
- 2.1 Point Process . . . 10
  - 2.1.1 Theory of point process . . . 11
  - 2.1.2 Poisson Process . . . 11
  - 2.1.3 Hawkes process . . . 13
  - Limitations . . . 14
- 2.2 Deep Learning . . . 14
  - 2.2.1 Review of popular deep neural learning architectures . . . 15
  - 2.2.2 Recurrent Neural Networks . . . 15
  - Long-Short Term Memory Networks . . . 16
  - Convolutional Neural Networks . . . 17

Temporal Convolutional Networks . . . . . 19
2.2.3 Recurrent Marked Temporal point process . . . . . 20
Architecture of RMTPP . . . . . 21
Limitations . . . . . 22
2.2.4 Neural Hawkes Process . . . . . 22
Limitations . . . . . 24
2.2.5 Self-Exciting Hawkes Process . . . . . 24
Model Architecture of SAHP . . . . . 25
Limitations . . . . . 26
2.2.6 Transformer Hawkes process . . . . . 26
Transformer neural network . . . . . 26
Model of THP . . . . . 28
2.3 Reinforcement Learning . . . . . 29
2.3.1 Introduction to RL . . . . . 29
2.3.2 Types of RL . . . . . 32
Model free methods . . . . . 32
Model based methods . . . . . 33
2.3.3 Limitations . . . . . 34

**3 Research Methodology** **35**
3.1 Problem formulation . . . . . 35
3.1.1 Preliminaries . . . . . 36
Previous work . . . . . 36
3.2 A Deep Reinforcement Learning Approach . . . . . 37
3.2.1 DRL Algorithms . . . . . 38
Continuous action space based Algorithms . . . . . 38
Discrete action space based Algorithms . . . . . 39
3.3 Dataset . . . . . 44
3.3.1 Description of the Dataset . . . . . 44
Synthetic Dataset . . . . . 44
Real-world Twitter Dataset . . . . . 46
3.3.2 Data Acquisition . . . . . 46
3.4 Implementation . . . . . 47
3.4.1 Optimis framework . . . . . 47
Training workflow . . . . . 48
Prediction workflow . . . . . 48
3.4.2 Data Preprocessor . . . . . 49
Converting from discrete to a continuous domain: . . . 49

3.4.3 LSTM Forecaster . . . . . 50
3.4.4 Custom Gym Environment . . . . . 51
Custom gym environment . . . . . 52
Defining the state, action and reward . . . . . 52
Illustration of the generalisability . . . . . 55
3.4.5 Optimis Agent Implementation . . . . . 55
3.4.6 Running example . . . . . 57
Summary . . . . . 59

**4 Evaluation** **60**
4.1 Control parameters . . . . . 60
4.1.1 Algorithms considered . . . . . 60
4.1.2 Action Penalty . . . . . 60
4.1.3 Evaluation metric . . . . . 61
4.2 Evaluating on the Synthetic dataset . . . . . 61
4.2.1 The synthetic data . . . . . 61
The baseline metrics . . . . . 61
4.2.2 The results . . . . . 62
4.3 Evaluating on the Twitter dataset . . . . . 62
4.4 Evaluating on running example . . . . . 64
4.4.1 The results . . . . . 64

**5 Conclusion** **66**
5.1 Future scope . . . . . 67

**Bibliography** **68**

# List of Figures


1.1 Four stages of fact-checking . . . . . 2
1.2 High level architecture of Optimis . . . . . 4
1.3 Discrete time series . . . . . 5

2.1 Poisson Process . . . . . 12
2.2 Poisson Process . . . . . 13
2.3 Hawkes Process . . . . . 13
2.4 RNN architecture . . . . . 16
2.5 LSTM Architecture . . . . . 16
2.6 Standard Multilayer perceptrons . . . . . 18
2.7 Model architecture of CNN . . . . . 18
2.8 Encoder-Decoder of Temporal Convolutional Network . . . . 19
2.9 Temporal Convolutional Network Model . . . . . 20
2.10 Model Architecture of RMTPP . . . . . 21
2.11 Deterministic finite state automation . . . . . 23
2.12 An event stream from a neural Hawkes process . . . . . 23
2.13 Model architecture of self-attentive Hawkes Process . . . . . 25
2.14 Transformer Architecture . . . . . 27
2.15 Multi-Head Attention - Attention layer . . . . . 27
2.16 Model of Transformer Hawkes process . . . . . 28
2.17 Comparison of the event sequences in various models . . . . . 29
2.18 Illustration of event sequences on a graph . . . . . 29
2.19 Reinforcement Learning cycle . . . . . 30
2.20 Actor-Critic Architecture . . . . . 33

3.1 Schematic structure of DRL agent . . . . . 38
3.2 Sample efficiency for DRL . . . . . 39
3.3 Comparison of several DRL algorithms . . . . . 41
3.4 Architecture of PPO . . . . . 41
3.5 Simulated data using hawkeslib . . . . . 45
3.6 Simulated data using tick . . . . . 45
3.7 Data acquisition workflow . . . . . 46
3.8 Framework of Optimis . . . . . 48

3.9 Training model architecture of Optimis . . . . . 49
3.10 Prediction model architecture of Optimis . . . . . 49
3.11 Domain Example OpenAI . . . . . 53
3.12 Illustration of a running example . . . . . 58
3.13 Graph for the running example . . . . . 58

4.1 Simulated data using hawkeslib . . . . . 62
4.2 Twitter highlight data . . . . . 63
4.3 Evaluation for running example . . . . . 64

# List of Tables

3.1 gym.spaces modules . . . . . 51
3.2 Features of SB3 . . . . . 55
3.3 Features of RLlib . . . . . 56

4.1 Evaluation results on synthetic data . . . . . 63
4.2 Evaluation results on Twitter data . . . . . 65

# Chapter 1

# Introduction

## 1.1 Objective

The main idea of the thesis is Influence maximization (IM), It encompasses of a broad class of applications: effective broadcasting, ad placement, viral marketing, etc. IM can be thought of as a decision making mechanism on point process where the broadcaster decides when to post to maximize her post visibility. Reinforcement learning is a good candidate for modeling on decision making processes. That works well for temporal data. The objective are as follows:

1. A custom RL environment that can work with temporal point processes, and can accommodate the needs of various influence maximization strategies - y adding a layer of customization over the API like action penalty, max allowed action and time limit, many high-level posting strategies are possible

2. The RL environment enables the use of simple policy models in discrete domain so that they can be implemented with ease by application developers  researchers - By implementing an environment with preprocessing techniques that make point processes suitable for simple in-built agents like PPO, DQN, and A2C.

## 1.2 Motivation

As a part of a case study done at the university, the topic of fact verification was explored in depth, and an implementation of a state-of-the-art fake news detection framework was achieved. As a guideline for building the

fake news detection system, inspiration was taken from a prominement non-profit fact checking organisation, *fullfact* [1]. According to FullFact, there are fours stages to a successful automated fact-checking pipeline[1] shown in figure 1.1.

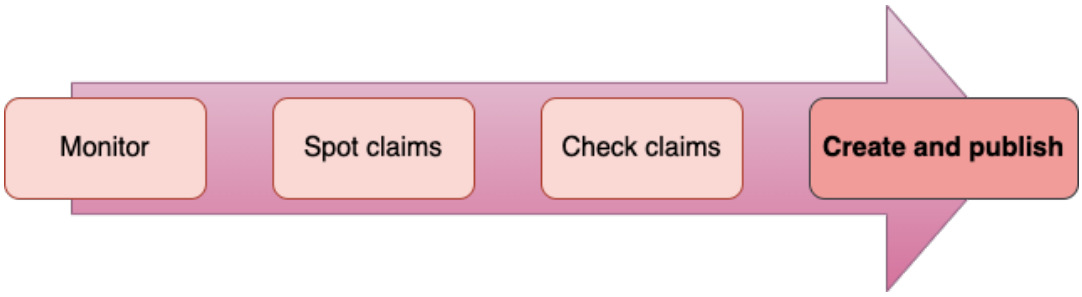


FIGURE 1.1: Four stages of fact-checking [1]

The scope of the case study was limited to claim-checking. According to full-fact, a significant step in the process of tackling fake news is to counteract the spread of fake news using the truth. This requires effectively broadcasting the truth into a community which is already affected by misinformation. This objective initially led to the motivation behind choosing the scope of the thesis. After a few iterations of the concept, an area of novelty and broad appeal was identified. The objective of the framework subsequently expanded to form a more generic use case where the effective broadcasting of information in social media was considered.

## 1.3 High level problem definition

Many large-scale online social networking sites, such as Facebook, Twitter, Instagram, and LinkedIn, have recently been popular because they are beneficial platforms for linking people and putting them together in small and isolated offline social networks. Besides, they are becoming a significant forum for marketing and promotion, enabling a broad audience to be influenced by information and ideas in a brief period of time. However, several obstacles have to be faced with leveraging these social networks as marketing and information dissemination channels. Trying to reach the right audience in an online social network has become increasingly popular and works for many individuals who want to share their views or market a product. For example,

[1]https://fullfact.org/media/uploads/full_fact-the_state_of_automated_factchecking_aug_2016.pdf

a marketer trying to push a product or a politician trying to send a message to the community.

Let's consider a hypothetical scenario where a small company is trying to develop an online shopping application. The company has a limited marketing budget but wants to reach the maximum right audience over a network. To find out the initial users who will market the online application can be promoted by influencers who will not only love the application but also use it. The influencers will start to spread and promote the application to their friends and followers, therefore reaching their target audience. It is essential to choose the appropriate group or community which will have the greatest impact on the largest number of people.

In this digital era where almost all everything is marketed on the web. Many individuals want to promote their ideas and views to the community but cannot find the right target audience on social media. This is known as influence maximization (IM), where IM can be thought of as a decision making mechanism on point process where the broadcaster decides when to post to maximize her post visibility. IM is a small subset of nodes in a social network can increase the spread of influence [2]. The social networks are vast and have complex structures, which makes it very challenging; therefore, building a model that is able to solve this problem could be beneficial.

Figure 1.2 which shows a high-level view of the problem and the solution which uses the the framework Optimis to identify the optimal time for posting. I present this thesis study to resolve one of the challenges, namely identifying the trends in social media and to effectively broadcast information across an online community.

The problem essentially boils down to the challenge of making optimal decisions in a discrete environment. This can be much more broader than the influence maximization problem that is mentioned above. It can accommodate other use cases such as:

1. **Mitigating fake news**, where the objective becomes knowing when the right time to post the truth is in a misinformation-affected community so that the truth spreads fast.

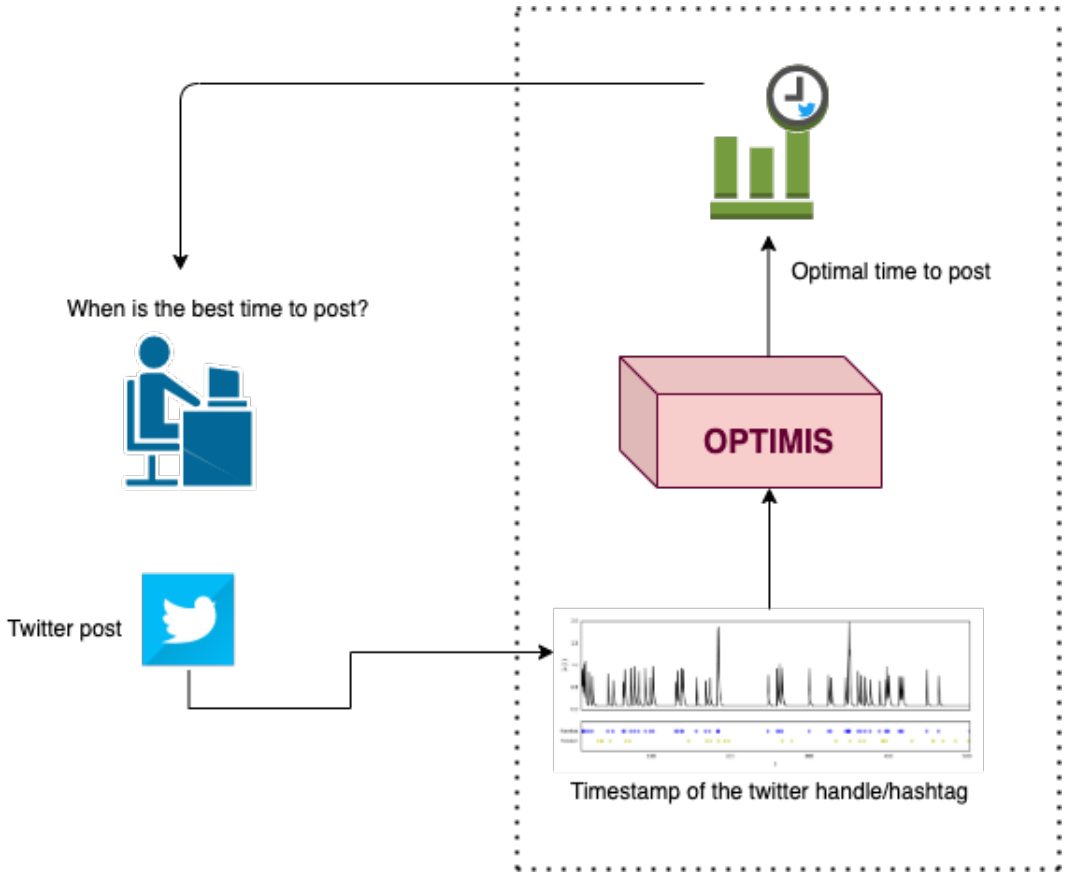


FIGURE 1.2: High level architecture of Optimis

2. **A trading environment** where the user is observed discrete signal of highs and lows and she is expected to making an optimal decision that maximizes profit. Making appropriate reward functions is introduced, which adds a negative value to each trade action taken so that the agent will be more judicious while trying to bull and sell the stock. Negative reward can is assigned for not not making a trade.

3. **A personalized memorization application** that helps the user in remembering information by identifying the right time to let the user know when to recall a particular information.

## 1.4 Background

### 1.4.1 Discrete vs continuous time series

Time series that are observed at regular spaced intervals of time are discrete time series, essentially here the time is described as discrete [3], but the underlying data source is always continuous in nature. We can get discrete events by sampling, binning and aggregating such event data. A continuous time series are observed a irregular spaced interval of time. But in some domains are not feasible especially in time series data analysis. Models such as ARIMA, AR and other regression models along with state-spaced models. This thesis deals discrete events in continuous time represented as a point on a time line as shown in figure 1.3.

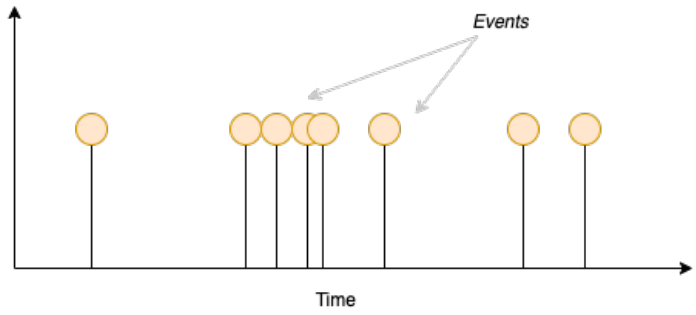


FIGURE 1.3: Discrete time series [3]

### 1.4.2 Marked temporal point processes

A probability distribution of points over time is referred to as a temporal point method [4]. The concept of Marked Temporal Point Processes (MTPPs) has gained popularity in recent years. For modeling asynchronous events in continuous time has become increasingly common. There are wide application domains, from social domains to financial and even health domains. In a social network, the events represent the users' posts or likes; in finance, the events can represent selling and buying of stocks; in health, they may represent when a patient exhibits symptoms of the current epidemic. In most scenarios, the problem would be in designing the model that can calculate the appropriate conditional intensity of the event and the distributions of the corresponding mark. Equation 1.1 [5] refers to the intensity function of a marked temporal point processes. This type of equation is particularly used for earthquakes and magnitudes occuring at time 't' and the $\lambda$ function here denotes the ground tensity function of the marked process. When calculating

the magnitude of the earthquake the intensity of the event type is taken into consideration and if the events are dependent of past events, the distribution of these intensities are calculated and we can find the peak of the intensity and the potential event of the earthquake can be calculated.

$$\lambda^*(t, \kappa) = \lambda^*(t) f^*(\kappa \mid t) \tag{1.1}$$

For a given specific user, we can find the visibility measure from the position of the most recent posts on the followers' or communities' feed over time [6]. More specifically, by using the framework of MTPPs, we can represent each point in time as the users' post and feeds, which thus characterizes using conditional intensity functions.It is a function that expects the number of occurrences at a specific time of all the events upto now. It represents the point behaviour analysis of the MTPPs. In order to solve the "when to post" problem, the need to find the optimal intensity function by examining an alternative view of marked temporal point processes as Stochastic Differential Equations (SDEs).

The technical process for modelling event sequences (MTPPs) are usually a family of point processes such as the Poisson and Hawkes process which have been increasingly used to model user activities on social media [7]. Poisson process is not a good approximation of the underlying dynamics as it doesn't capture its true essence i.e. Events are not dependent on each other and the failure of one event does not affect another event. This leads to unexplainable errors. The main limitation doesn't capture complex dynamics.

### 1.4.3 A strong parametrization

In objective when it comes to machine learning is to find the function $f$ that maps the input variables (x). Generally, this approach can be divided into 2 sets of models: parametric and non-parametric. Parametric model assumes that the function is in a particular form which creates a bias. The model assumes that the data is linear in nature in the case of linear regression. This assumption (or bias) does have certain advantages such as explainability and simplicity of the model [8], but a major disadvantage is the bias or the inflexibility of the model.

Classical Hawkes process model is a parametric model that is point process analysis [9]. It can raise the probability of future events taking into consideration the past events and exponentially decaying with time, it tries to fit into a concrete equation whereas most events cannot be fitted into a perfect equation therefore leading to less flexibility in its prediction.

### 1.4.4 Non-parametrization of the problem

Non Parametric models assume that they do not make assumptions about the mapping function of the input variables to the output variables [8]. The parameters are adjustable and are prone to change and have a higher model complexity. Feature engineering is not as important compared to parametric models [10]. ML algorithms such as k-nearest neighbors, decision trees, Neural Hawkes processes and self-attention Hawkes processes are some of the non parametric models.

Deep neural network models make less or no assumptions about the data. Non-parametric models require a large volume of data to learn a function from the input to produce a desired output and avoid over-fitting, they are rather flexible as they minimize the assumptions they make about the data. Since we are working with large twitter data, it only is feasible to work with non-parametric models [11].

In this thesis, I will be introducing a Deep Reinforcement Learning(DRL) approach to the "when to post" problem. RL refers to both sub fields of machine learning and problems of learning, which has become increasingly popular and always on the news lately. RL-based systems like the *alpha-go* have now defeated Go's world champions, and while the RL agent learned by playing itself, it helped the agent master the game. A lot of RL based systems that were developed especially in the gaming domain like Atari games. Many more positive findings are being seen by the research community.

RL is a part of machine learning where it consists of agents who perform the actions and the observation or the environment sends the feedback based on the action along with its rewards as to maximize the returns. The environment is in the form of a Markov decision process (MDP) [12] because of its dynamic programming techniques. The main objective of RL is to maximize the reward it receives which is "expected" when following the parametrized

policy itself. All MDP have at least one optimal policy which means it returns the maximum rewards. There are challenges in performing reinforcement learning approach but the end results are better than most neural network techniques. According to the above mentioned optimal control problem, the desired solution would be to use a Deep Reinforcement Learning approach to find the optimal time to post in social media.

### 1.4.5 Key contributions of Optimis

In this thesis, a reinforcement learning based-framework is designed with the purpose of:

1. Enabling social media users to find the optimal time to post in an online social environment like Twitter or Facebook so that their messages are broadcast widely and effectively.

2. Assisting researchers in building reinforcement-learning algorithms on discrete temporal settings by providing a framework which is scalable to other related domains.

The framework achieves these contributions by implementing the following features:

1. **An RL-based agent** that is capable of identifying temporal patterns in a non-parametric manner and optimising actions in order to maximize post survival time.

2. **An Open AI Gym environment** that supports discrete time domain problems like post time maximization. In theory, the framework can be extendable to application domains such as financial trading, viral marketing and spaced repetition.

### 1.4.6 Thesis outline

The remaining part of the thesis is structured as follows:

*Chapter 2* reviews the previous literature in the field of smart broadcasting on temporal data. This chapter briefly discusses the various types of point processes used in the field today and the theory behind it. Provides a formal introduction to machine learning and reinforcement learning by explaining the preliminaries associated with it. It outlines various open-source frameworks and libraries that are in use today, including the definition of scope addressed by the thesis in detail. Finally, the chapter provides an overview of various statistical, machine and deep learning models that are popular choices in temporal point process analysis.

*Chapter 3* discusses the problem formulation and the limitation of previous work, then introduces DRL and explains its different components. Introduction of a few DRL algorithms and provides motivation for choosing a particular algorithm for this thesis. Description of the dataset and how it is been used during implementation. Also in explanation of the optimis framework with the custom environment and policy network of stable baseline and RLlib.

*Chapter 4* evaluates the performance of the proposed framework over the datasets on various DRL algorithms and also explains the advantages as well as the limitations of this approach.

*Chapter 5* presents a summary and conclusion of the work done in the thesis and provides directions for future work.

# Chapter 2

# Literature Review

In this chapter, previous work on implementing the various approaches to solve similar problems and data involved in temporal point process analysis will be illustrated in brief. A section on the definition of a point process and various types of point processes along with their limitations. Providing a formal introduction to machine learning, deep neural network, and reinforcement learning by explaining the preliminaries associated with it. Along with some of the related work and models that have been implemented along with their limitations have been discussed.

## 2.1 Point Process

In a world where many events occur, the trends of these events are likely to follow a pattern. A point is known as event which is stochastic is nature and the occurrence of events in time and space [13]. Typical point processes are temporal and spatial. Temporal point processes are events that can be defined as a point in time. The likelihood of a new earthquake increases in a place where an earthquake has already occurred or rise in selling or buying of stocks in one country could cause a more similar event in a different country and even user activities on social media. The probability of similar events can be decreased by understanding the patterns of these sequences. If the probability of an event occurring increases due to previous similar events then they are categorized as excited or self-excited but if the probability of an event occurring decreases due to previous similar events then they are categorized as inhibited or self-regulating.

### 2.1.1 Theory of point process

1. Counting Process: Consider a line that represents event times where $t$ = $t1$, $t2$, $t3$....tn. We can count the number of events upto the current time$(t)$. Instead of modelling these events, we can describe it as counting process N$(t)$.
   Ex: N(80)=140 ; years=80, no of earthquakes=140

2. Conditional Intensity function(CIF): CIF is a function that gives the expectation of event occurrences at time$(t)$ and it denoted as $\lambda^*(t)$ which represents the rate at which the events occurs around that time, refer equation 2.1 [13]. This equation can be used to find the rate of intensity of the such as point or events, which could be the a patient who is having a viral infection, this could be dependant on another event, since hte patient might have gotten thte virus through another person therefore causing a dependency effect. This is conditional to the prior history H$(\mu)$ of the events that have occurred upto time$(t)$. Therefore by finding the no of events or patients that have occurred upto time from the beginning of time can be known

$$\lambda_{k(t)} = \mu^k + \sum_{n:t_h<t} \alpha_{k_j,k} \exp^{\left(\delta_{k_h k}(t-t_n)\right)} \tag{2.1}$$

When we talk about self-exciting or self-regulating, we can take into consideration the conditional intensity function whose history events cause the increase in the probability of an event which is defined as a self-exciting or Hawkes process. But if the history events does not affect the probability of an event to occur it is stated as a self-regulating or Poisson point process.

### 2.1.2 Poisson Process

A Poisson point process is a self-regulating process which means that the historical events do not influence future events, each event is treated as an independent event [14]. Eg: Spread of a virus in a geographic location. Figure 2.1 shows the points which are independently placed in a geographical location which is called spatial point processes.

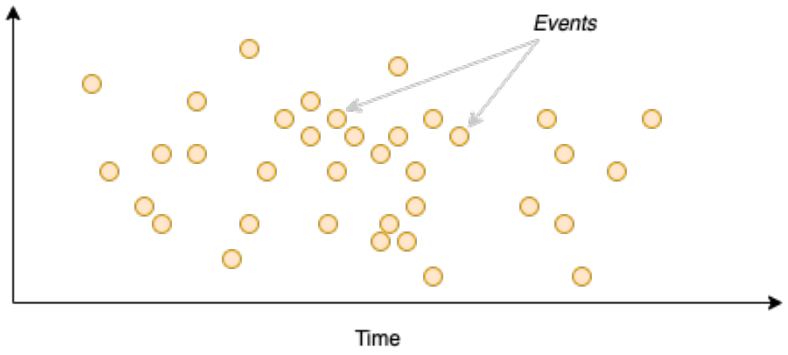


FIGURE 2.1: Poisson Process [14]

A Poisson process is a sequence of discrete events known as points in a line, with the average time between events known but the exact time between two consecutive events unknown. The current event at time $t$ are not influenced by the past times, each event is considered to be independent to each other. The Poisson process is a stochastic model which includes an infinite distribution like to Markov Process.

The Poisson process has a list of criteria it holds, first the events are considered to be independent of each other, the average time between the events are constant and lastly two events cannot occur at the same time. In reality, many of these fail to reach these criteria. Visitors to a website, stock price movements, and even consumers calling a service center are all examples of Poisson processes. Poisson process refer equation 2.2 [14]. Refer the figure 2.2 which shows a website, the site crashes on average once in 60 days, but this failure doesn't affect the probability of the next event.

$$\lambda^*(t) = \frac{f\left(t \mid \mathcal{H}_{t_n}\right)}{1 - F\left(t \mid \mathcal{H}_{t_n}\right)} \tag{2.2}$$

As a limit of the binomial distribution, the Poisson distribution appears. The Poisson Method is a model for explaining random events that isn't especially useful on its own. The task of performing probability distribution in which the number of events on a given time period before the upcoming events occur. The number of events seen can be adjusted by using the parameter $\lambda$. The probability mass function is illustrated in the graph 2.2 above. Using this function [14], observation of the probability is known. The waiting time between events can be known as well.


FIGURE 2.2: Example of a Poisson Process[14]

### 2.1.3 Hawkes process

The Hawkes process is a special type of point process that have numerous applications from earthquake to finance. If the probability of an event occurring increases which are dependent if the past or history events then such processes are to be called self-excited or Hawkes processes [15]. For a classical hawkes is called a univariate self-exciting temporal point process N whose conditional intensity function is linear, referred to as linear hawkes process and lambda is non linear then it is referred to as non-linear hawkes process. The two alternatives of univariate self-exciting temporal point processes, one is called the intensity based hawkes process which has a non-negative exponentially decaying factor and the other is called cluster based hawkes process which has a constant decaying factor.

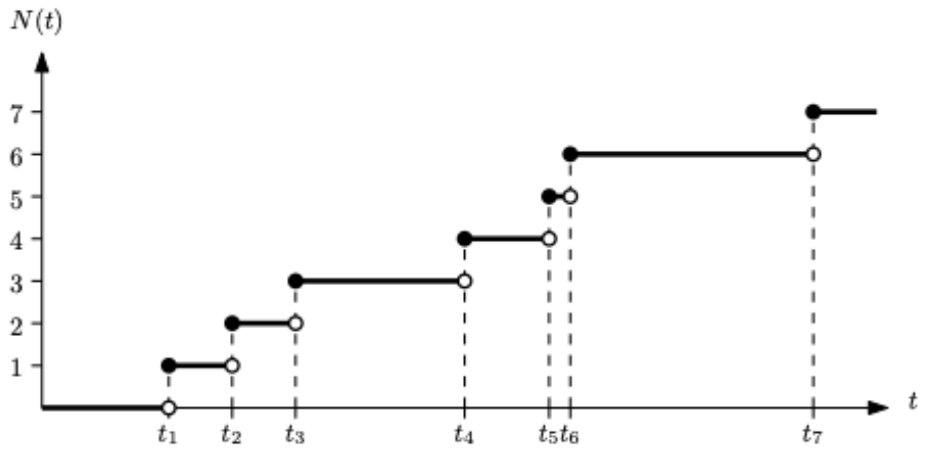


FIGURE 2.3: Hawkes Process [16]

Accordingly to the figure 2.3, we can see the time events from $t1$,$t2$,$t3$.....$tn$

and along the vertical axis, the counting process $N(t)$. We can calculated the intensity function using Hawkes process from equation 2.3.

$$\lambda(t) = \mu(t) + \sum_{i:\tau_i<t} \gamma(t - t_n)) \tag{2.3}$$

In addition several researchers have made generalized versions of the Hawkes process which have different exciting functions that can be applicable to various fields of study [16]. Due to the use cases of uni-variate processes, there came to be multivariate Hawkes process d-dimensional counting process $N(d)$ associated with intensity function which can be defined in the given equation 2.4.

$$\lambda_{i,d}dt = \mathbb{P}\left(N_i \text{ has } a \text{ jump in } [t, t + dt] \mid \mathcal{F}_t\right) \tag{2.4}$$

The probability of the future events increasing at time t depends on the history events which is called self-exciting. Therefore we can see a rise in the intensity of the upcoming events until it plateaus and then has a exponential decay which is called the intensity based Hawkes process.

**Limitations**

The major limitation of Hawkes process is that history events cannot suppress the occurrences of future events which is not likely in a real world scenario with complex events.Hawkes process is a parametric model [16] therefore it tries to fit into a parametric form of the equation making it less flexible to change and leading to errors in the prediction. That's where neural networks comes in, since they are good universal function approximators ( i.e they can mimic any equation)

## 2.2 Deep Learning

Deep learning or deep neural network is a branch of ML that mirrors the functioning of a human brain in processing of data as well in the decision making process. It is also referred to as artificial intelligence that consists of a web of networks connected to neurons capable of learning unstructured

knowledge. The world is filled with data in every form drawn from social media sites, e-commerce sites and many others. Companies understand the importance of these data that can potentially help the companies get more insights. The data is usually unstructured, therefore cleaning the data is an impossible task for a human without the help of AI systems.

In deep learning, generally RNN, LSTM, GRU, and a variant of the convolutional neural are popular in temporal point process analysis. The basic understanding of these architectures are seen in this section.

### 2.2.1 Review of popular deep neural learning architectures

There are a variety of simple tasks which can be solved by using vanilla neural networks. Classification and regression problems for example. When the inputs are assigned with a class or a label, the problem can be solved using a classification type model and when a real value is used where predictions are made, it can be solved using a regression type neural network.

### 2.2.2 Recurrent Neural Networks

Vanilla type neural network do not need to remember the things they learn during training. This causes an disadvantage on many levels when solving data which is sequential in nature. These sequential data are data points which are interdependent to each other. For example the stock prices or user behaviour. That's when Recurrent neural network comes into the picture, RNN can keep information of the sequential data that helps in given more context in identifying patterns and any co-relations between the data points.

Figure 2.4 shows the simplest architecture of RNN which is Incorporated with a feed forward neural network, it can store the information forwards and backwards in time [17]. It comprises of a single neuron able to process inputs sequentially providing an output and sending it back to itself. The RNN unit has a set of weights aligned to each input. After the output is computed, the weights are re-adjusted by using the loss function, this is called back-propagation through time(BPTT).

There are problems in RNN which are the vanishing gradient problem and the exploding gradient problem as well [17]. In addition to the hidden layers can be added but still after certain time, the data is lost or forgotten by the

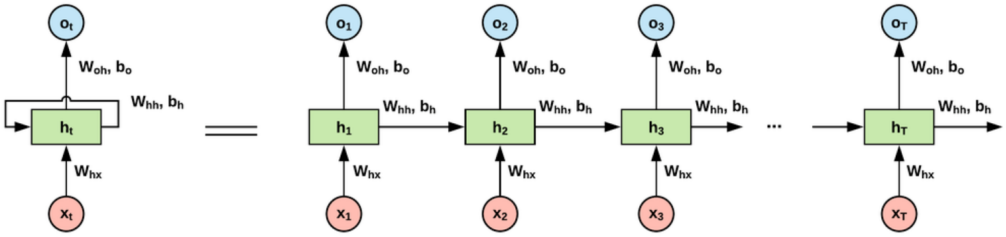


FIGURE 2.4: RNN architecture [17]

recurrent unit. Parameter optimization helps in tuning the model for better performance and better model accuracy.

### Long-Short Term Memory Networks

Long Short Term Memory networks is a modified version of RNN, capable of learning long-term dependencies. Since RNN has a vanishing gradient and exploding gradient problem, LSTM can solve this issue with the introduction of a memory unit which can remember information for a long period of time [18]. LSTM is designed in such a way that they can solve many of the sequential data which has a dependency problem which could not be solved by RNN. Especially used in solving time series forecast such as stock prices or earthquake data.

All types of RNN based model form a type of chain of modules of the neural network. The repeating module has a different structure of several neural network layers. Here the *tanh* layer is used as a connecting chain refer figure 2.5.

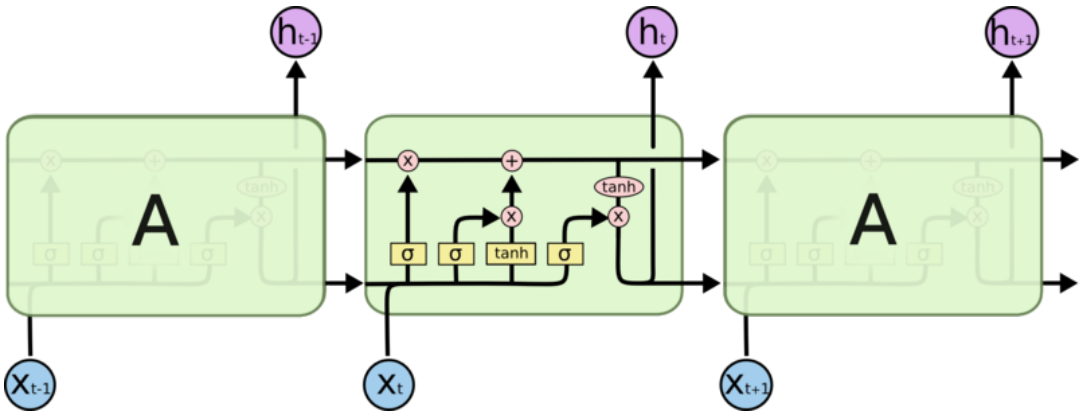


FIGURE 2.5: LSTM Architecture [18]

The main functionality of LSTM in the cell which works like a chain with minor linear interactions. And the information can flow through it unchanged. It also has the capacity to add or remove any information from the cell state. This type of regularization of information can be dealt with by using gates. Gates can be handled in such a way that it can handle if the information needs to be let through or not by a value. Gates are featured with a sigmoid neural network that helps such information to pass or not [18]. An LSTM has three of these gates, to protect and control the cell state. Using LSTM has helped dealing with sequential data to another level, in terms with long term and short term dependencies. There has been a lot of improvements in the field of Recurrent neural networks with the Gated units taking a advantage on the way models are treating sequential data.

**Convolutional Neural Networks**

MultiLayer Perceptrons (MLP), which are conventional neural networks (MLP). Neural networks functions like the human brain, in which neurons are stimulated by attached nodes and triggered only when a certain threshold value is reached. MLPs have a number of disadvantages, especially when it comes to image processing. For each input, MLPs use a single perceptron refer figure 2.6. For large pictures, the number of weights quickly becomes unmanageable. There are approximately 150,000 weights to prepare for a 224x224 pixel picture with three color channels. As a consequence, preparation can be challenging, and over-fitting can occur.

Clearly we can see that MLP cannot be used for such type of data including temporal data, there are many types of CNN model to incorporate temporal data which are univariate or multivariate where the model can learn from past observations and to predict the next value in the sequence. A dense completely connected layer follows the convolutional and pooling layers, which interprets the features derived by the convolutional portion of the model. A flatten layer is used between the layers and the dense layer to reduce the feature maps to a single one-dimensional vector, as seen in figure 2.7.

The model's design leads to its multi-scalability in the first convolutional layer, convolution is performed on three parallel independent branches. Each branch uses different time and frequency scales to extract different types of features from the data. This network's structure is made up of three stages: transformation, local convolution, and full convolution [19]. An advantage

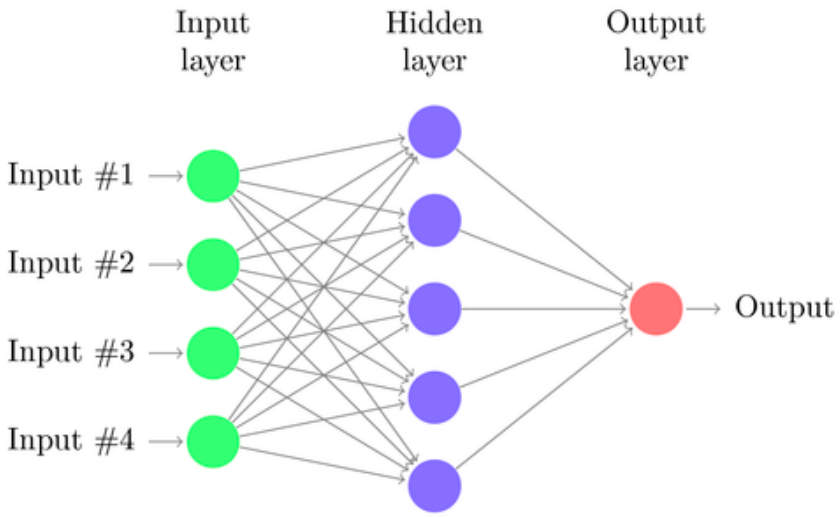


FIGURE 2.6: Standard Multilayer perceptrons

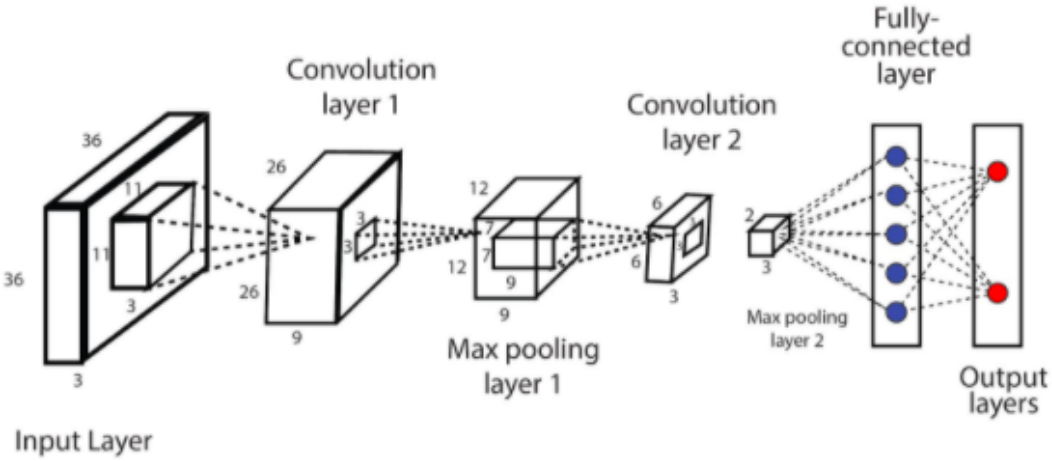


FIGURE 2.7: Model architecture of CNN [19]

of CNNs is the processing of large amounts of data can be done and helps in the feature representation of the structure of data sequences.

### Temporal Convolutional Networks

Temporal Convolutional Networks (TCNs) are a complex version of Convolutional Neural Networks (CNNs), combining some of the best implementations of modern convolutional architectures. Since TCN uses the basic building blocks of CNN, since they are dominating and are prefect neural models in forecasting. There are a variety of implementation on TCN models for sequence modelling tasks since it has longer effective memory in the field of deep learning. TCN results suggest that it outperforms the baseline of other recurrent neural models in the range of sequence modelling [20].

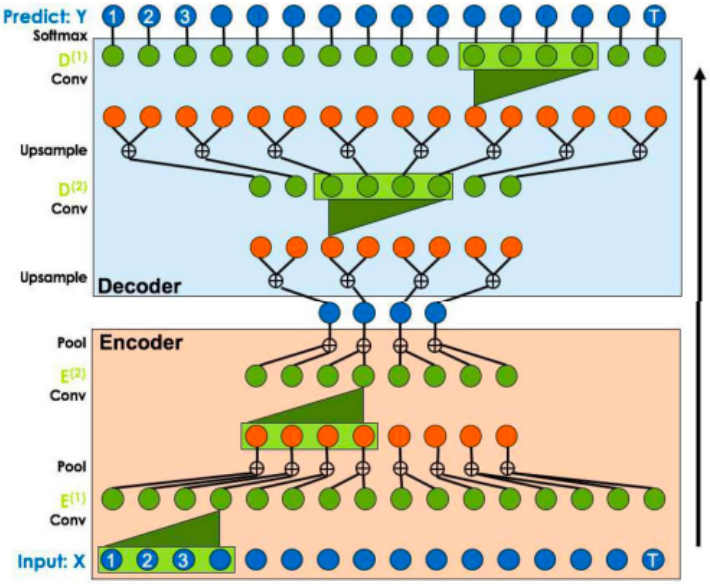


FIGURE 2.8: Encoder-Decoder of Temporal Convolutional Network [20]

The TCN architecture shows that they excel in model accuracy when compared to other neural nets like RNN, LSTM, GRU. And the model is much more simplier. TCN has been widely used temporal point process and time series forecasting and even audio and image processing. They can process data parallelly which outperforms RNN based models which brings vanishing gradient problems now leading to better performance. TCN can take a any number of length as input and outputs the same no of length as shown in the figure 2.8. Instead of using a cell state to preserve information from previous outputs as in LSTMs, TCNs use connection between previous hidden layers configured with two hyperparameters: dilation factor and filter size.

From the following figure 2.9, ($d$) is the factor of dilation that decides the number of layers between the output and the hidden layers. $d = 1$ says there are no intervals between layers and $d = 2$ says that a connection should be made or creates a connection between layers. ($k$) is the total size which is a deciding factor of the number of layers in between. Input layer is used by $d = 1$, all the information of the network is stored in hidden layers generating a "receptive field". Here optimization of the parameters can take place which is best suited for tuning. It Helps in increasing performance speed and the increased accuracy. The size of the receptive increases in size as we tune the parameters during optimization creating more advantage to the model. Mainly used in the field of image classification where pixel-wise inputs are used [21].

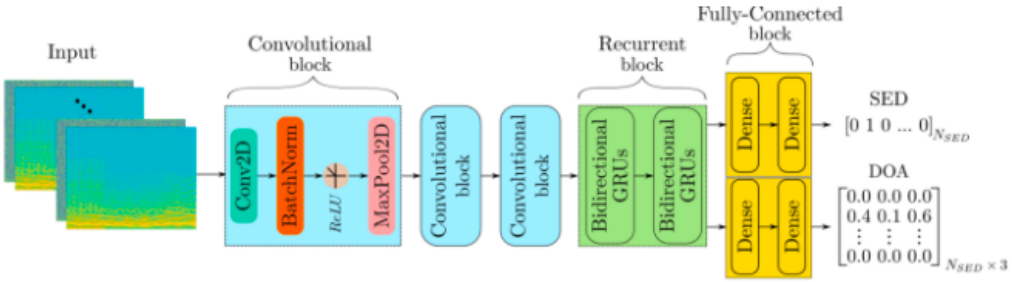


FIGURE 2.9: Temporal Convolutional Network Model [20]

For the most part, recurrent neural network architectures are still commonly associated with deep learning in sequence modeling. However, research has shown that a TCN can outperform these types of models in several tasks, both in terms of predictive success and productivity.

### 2.2.3 Recurrent Marked Temporal point process

Due to the model's inflexibility, the classic Hawkes process is insufficient to find the best intensity feature. More information can be extracted from the events, something called as a marker in needed. This may include financial transactions, social media user activity, and clinical incidents such as outbreak diagnosis. The intensity function can be applied to the classic temporal point method to collect more information, and each marker is treated as an independent dimension. Because of the increased complexity of the intensity feature caused by markers, the model's versatility is decreased. It is normal

to have a large number of markers, which creates a sparsity issue in each dimension, where only a few events will occur.

The paper [22] aims to introduce a more complex MTPPs by using a RNN model that can use the information it has on the events such as the event timings and the markers. RNN here is used to understand the dependencies it has on the event scale by using the gradient method which helps in avoiding the over-fitting of the model itself. The ability to capture more complex event sequences in a more stochastic way. RMTPP model can understand the dynamics of model it holds whether it it known or unknown. This can achieve a better model performance and an accurate prediction. The proposed RMTPP has implications beyond space and time settings.

**Architecture of RMTPP**

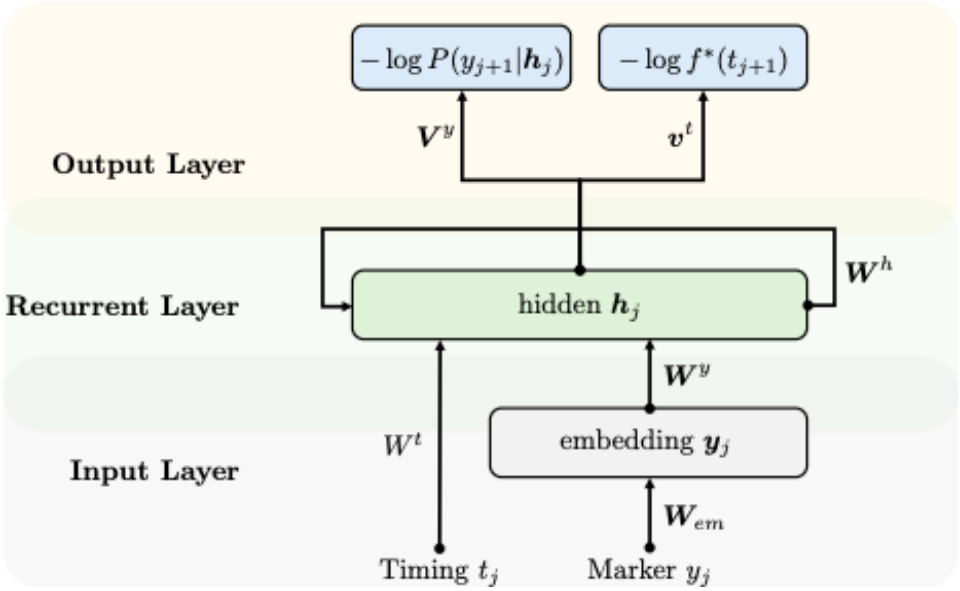


FIGURE 2.10: Model Architecture of RMTPP [22]

Architecture of RMTPP, refer Figure 2.10, shows the time of the event $t_j$ and the marker $y_j$. These input pair $(t_j, y_j)$ will be the inputs that goes into the recurrent neural network. The marker event input will first be embedded along with the weighted matrix and then send into the RNN. The hidden states $h_j$ learns about the non linear dependency from the event inputs i.e the history events. The hidden vector is updated after receiving the current input. Through back propagation phase, the loss is reduced, here it's the log likelihood of the model and update the parameters with the help of the loss

function. The output is the next event type and when exactly the event will take place.

#### Limitations

RNN-based versions, on the other hand, have two major disadvantages. Long-term dependencies are unlikely to be captured by recurrent neural networks, including those fitted with forget gates, such as Long Short-Term Memory and Gated Recurrent Units. The trainability of recurrent neural networks is the second downside.

### 2.2.4 Neural Hawkes Process

The Hawkes process works on determining the event function of the events from the hidden states of the recurrent neural network (RNN). Hawkes's process is a self-exciting process which uses the conditional intensity function by taking into consideration the history events which can influence the current events. It performs in a similar way as to a deterministic finite state automation which states that if a finite sequence of inputs, here being 0s and 1s are taken, the finite state machine will run through the state sequence and accept or reject certain values in order to search the simplest model to find the finite state machine, mainly used for pattern matching by researchers. The figure 2.11 shows an example of how the state transition happen, similar to as a Markov chain process for infinite number of states. There are previous works on using RNN based models on Hawkes process [19], But there are drawbacks on how the model interprets the temporal points. The Neural Hawkes Process [23] uses LSTM so that it can capture into the context the infinite sequence which is lost when using the classical Hawkes process method. The model is such a way that the states are updated on every event occurrence in time and between those events are also updated. LSTM holds good for long term dependencies for both the history events as well as the future events, being able to capture more complex sequence dependencies.

Hawkes's process doesn't know how to cope with the missing data issue, since the Hawkes process is only able to capture partial event sequence. Dealing with missing data in the temporal point process can be solved using the Bayesian Optimization (BO) method [25]. By finding the probability distribution of each missing value and imputing the missing value, by drawing a sample from its distribution. In many cases there is a probability that some

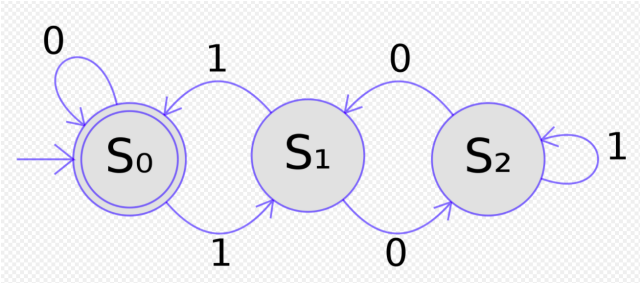


FIGURE 2.11: Example of how deterministic finite state automation works [24]

events are not recorded and can be very complex when working with statistical models to predict the intensity function using Hawkes's process since it affects the influence in the events that happen in the future. Therefore, using Bayesian Optimization, by using Gaussian process regression it interpolates from the existing data points. The paper [23] states to incorporate an LSTM model which is RNN based and is complex enough to learn the distribution of the true prediction from the hidden states as well as update the states table, and also observe the past events from of the Bayesian model from the hidden state while making updates when coming across new observations.

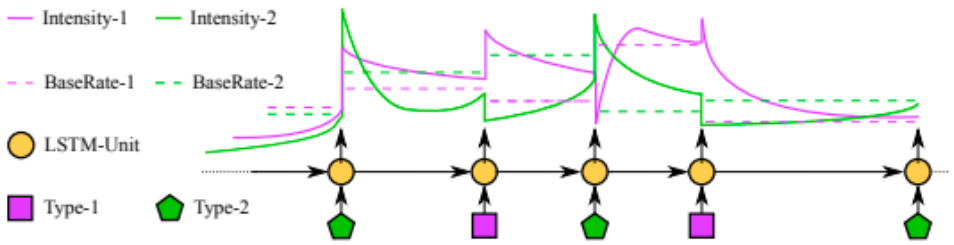


FIGURE 2.12: An event stream from a neural Hawkes process [23]

The figure 2.12 illustrates on how the event streams arrive at the hidden state from the neural network. LSTM reads the sequence all the sequences in the temporal time process which includes the past events. From the figure the future intensities can be determined from the history. There are 2 types of which, an event of one type can excite the same type of event and inhibit the second type of event based on the approach. The sudden jump in intensities can be seen as this happens shows a shift in intensities.

$$\ell = \sum_{\mathbf{i}:\mathbf{t_i}\leq\mathbf{T}} \log \breve{}_{\mathbf{k_i}}(\mathbf{t_i}) - \underbrace{\int_{t=0}^{T} \lambda(t)dt}_{\text{call this } \mathbf{\Lambda}} \tag{2.5}$$

The loss function generated from the LSTM model shown in the equation 2.5. Using the model's flexibility of multivariate Hawkes process on the CLSTM which is an interpolated version of the standard LSTM. This increases the interpretability of the model and the events seen which consists of both the effects of excitation and inhibition of the events.

**Limitations**

There is always a drawback in using the RNN based models because of the inability to capture more long term dependencies. Also to train the model is difficult because of the high computational cost.

### 2.2.5 Self-Exciting Hawkes Process

Self-attention by the name itself explains itself, providing more attention of the events in a temporal time process. These attentions are used to measure the events in the past which can follow a dependency to the current event. SAHP explains the need of computing event embedding along with positional embeddings before they can be inputted into the model. Positional embeddings of the based on the relative position of the events so that the event orders are taken into account. The self-attention here relies on the embeddings which takes into consideration of the order which is the positional embedding. Usually, positional embeddings are based sinusoidal functions such as the transformer neural network. [26] implies that there is a drawback in the actual calculations of these embeddings using sinusoidal functions, meaning that the distance of the position of each event is calculated by a constant shift of phase therefore not taking into account the order of the events. [26] proposes a shifted method for positional embedding called the time shifted positional embedding method [27]. The second part of the papers novelty states that the method is more interpretable than other RNN based neural network models.

### Model Architecture of SAHP

There are 3 tweet types ("small", "medium" and "large" retweeters) in respective to the time the post was made. $(v1, t1)$ and so on are the event type and event time can are embedded with event type embedding and positional embedding. The vectorized input in added sequentially into the self-attention model which is a multi-head attention model. $N(x)$ which states there can be as many attention layers in the model as possible until it's requires. After which it moves into a Add & Norm section where the normalization and the dropout of certain neurons which have less information and has no relevance to the prediction are dropped. Dropout is used to reduce complexity in the performance and the accuracy, and it also helps in avoiding model over-fitting. A feed forward layer is added for normal control of the flow of data. The output that is received is the historical hidden vector, with this we can compute the intensity of the type at a particular time. Attention is a mechanism combined in the RNN allowing it to focus on certain parts of the input sequence when predicting a certain part of the output sequence, enabling easier learning and of higher quality. The parameters of intensity function can be computed using the hidden vector $h_{u,i+1}$ using three non-linear transformation layer.

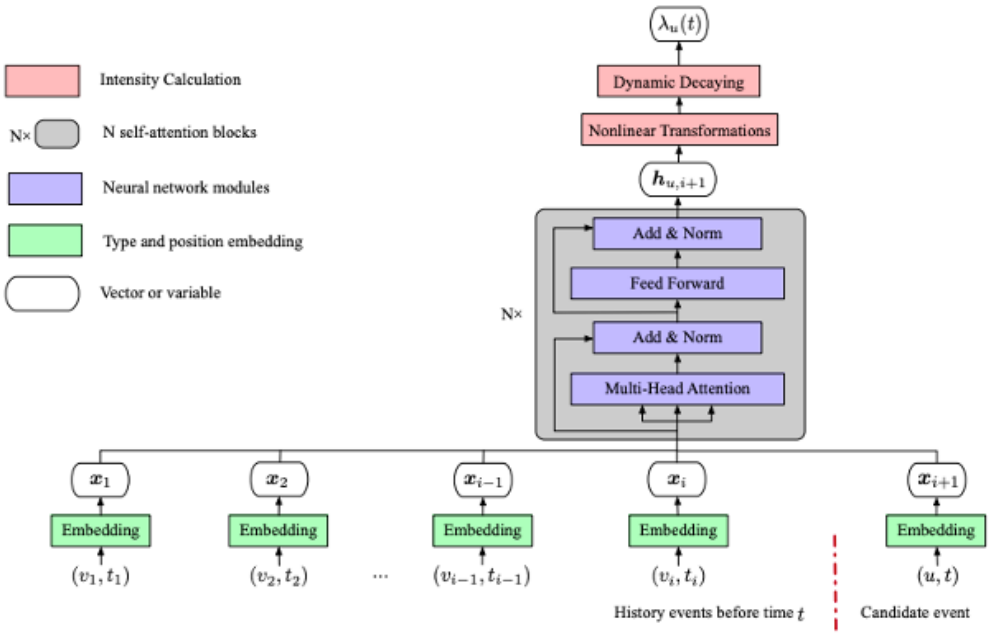


FIGURE 2.13: Model architecture of self-attentive Hawkes process [26]

Softplus is used for the decaying parameter defined in the dynamic decaying layer of the architecture. Since RNN takes into account the information about the sequence forward and backwards in time. Therefore, the information about the future is masked sing masking. [26] SAHP model is able to capture both the effects of exciting and inhibition and the figure 2.13 shows the architecture of SAHP shows how the events are encoding into vectors. .

#### Limitations

The drawbacks of the Self-attention Hawkes process in the use of RNN which is having vanishing gradient [28] problem therefore unable to capture any long term dependencies in the event data. It is highly cost in efficient since there is a large computational cost.

### 2.2.6 Transformer Hawkes process

#### Transformer neural network

Unlike the previous neural network models, transformer neural network is an encoder-decoder which transforms from one sequence to another without the use of an RNN model architecture. It uses the attention mechanism which can capture long-term dependencies in and input the values in parallel rather than sequentially unlike to RNN models like GRU and LSTM. There are many applications of in the field of NLP using transformer based neural network such as Bidirectional Encoder Representations from Transformers(BERT) [29] and Generative Pre-trained Transformer(GPT) [30].

The figure 2.14 shows that self-attention mechanism [31]. The input and the output are embedded into vectors since the model cannot directly input strings. Since the transformer doesn't have an RNN layer which can remember the sequence of data inputs, there are positional encoding that can give a relative position of the order of sequences.

A transformer has an attention mechanism which is described as a multi-head attention containing a Query and a key value pair which are all a set of vectors. They are used as mapping the vectors to an output. The output is a value of the weighted sum and these can be linked to each value by the query along with the key [31]. The figure 2.15 shows the representation of the multi-head attention mechanism

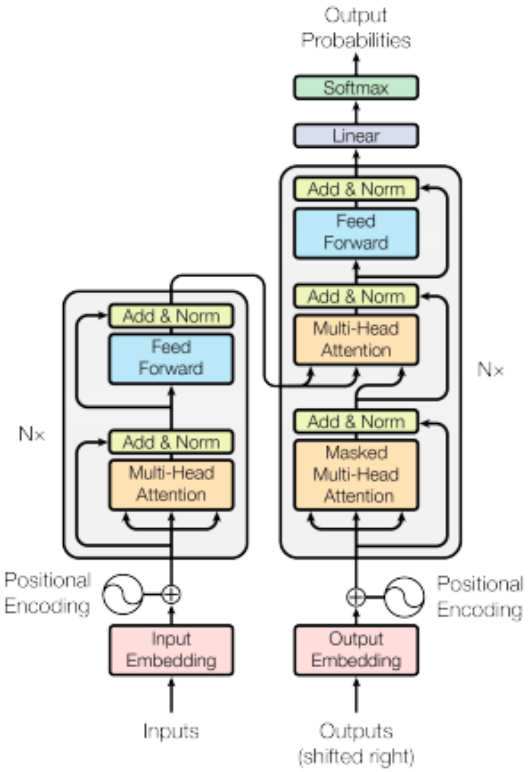


FIGURE 2.14: Transformer Architecture [31]

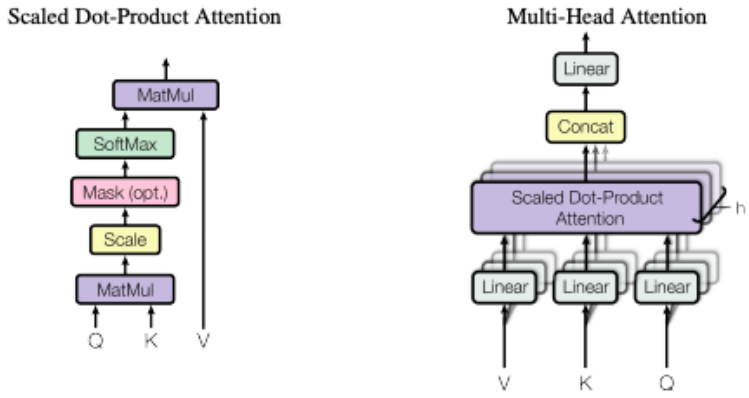


FIGURE 2.15: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel [31]

Transformer uses less computation cost compared to other neural network models and the results are better. Since transformer models were only used in Natural language processing (NLP) which are classification-based model. Since we are dealing with point process containing a value or prediction of a certain event, it is more of a classic auto regression model leading to change

in the transformer architecture.

**Model of THP**

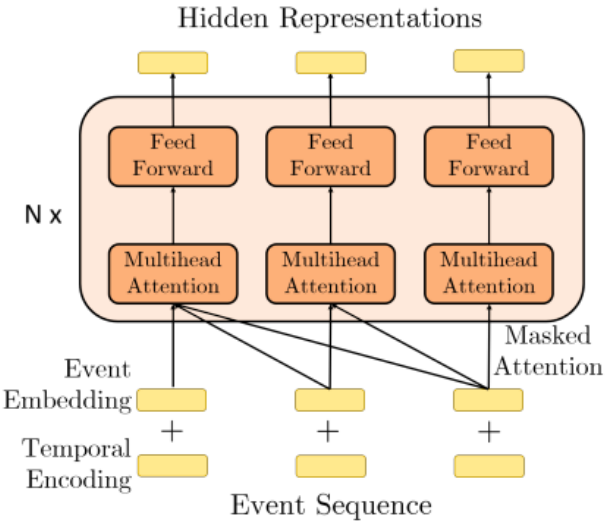


FIGURE 2.16: Model of Transformer Hawkes process [32]

There are various self-attention models such as SAHP [26] which was previously explained showing the multi head mechanism using an RNN model architecture [33]. The way it is done is a score will be calculating stating the level of dependence between two events and the larger the score the stronger the dependency and smaller the score, weaker the dependency. By this way the modules capture an event at any given temporal from the current time.

Figure 2.16 shows the representation of the model architecture where the event embeddings are done using multi-head attention and feed forward layers that run in parallel and the output are the hidden vectors which are further used to attain the intensity function.

The figure 2.17 shows how THP depends from an RNN and CNN model. It can capture long and short-term dependencies. And the input vector can process data paralleling rather than sequentially.

As we can see from the figure 2.18 where two events can be computed by stating each of the events to be independent to each other. There are ways to model such efficiency using graph method where a user is represented by a vertex and the connection between the user and associated user would be

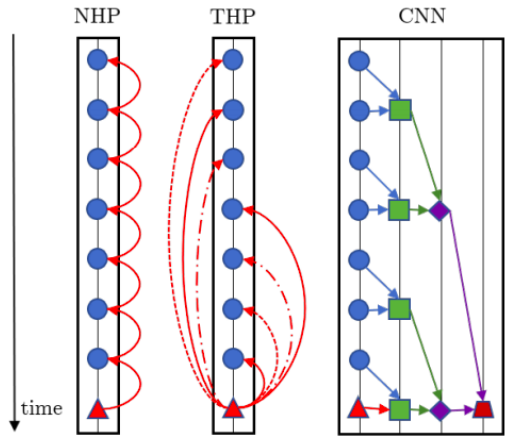


FIGURE 2.17: Comparison of the event sequences in various models [32]

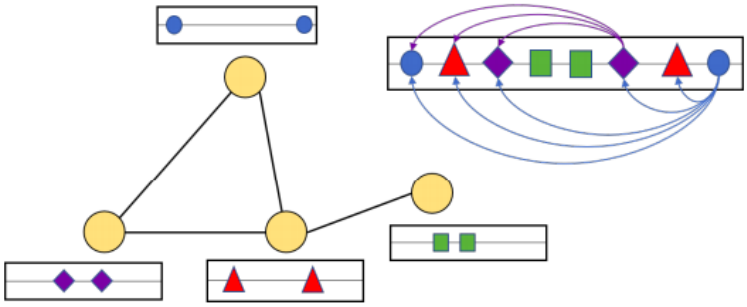

FIGURE 2.18: Illustration of event sequences on a graph [32]

represented by a edge of the graph described in Redqueen [6]. THP incorporated the relational graphs to the model via a similarity metric among users. Graph regularization can be used to calculate such a metric [34]. THP can achieve the state of the art both in performance, likelihood and event prediction accuracy.

## 2.3 Reinforcement Learning

### 2.3.1 Introduction to RL

Reinforcement Learning (RL) is a type of machine learning which deals with sequential decision making. Reinforcement learning solves the difficult problem of correlating the most immediate actions with a delayed return reward. Just like human beings, the RL agent has to wait to see the end results of

their actions after making the decision. There are a lot of applications that RL has been applied to especially in game settings like chess and other grid games [35]. The RL learner played against an actually chess champion and won which shows that it learns from itself and makes decision on the go. Other examples would be an adaptive controller or a mobile robot. The main idea behind RL (see figure 2.19) is that it has an agent that acts on the environment to obtain maximum reward. The objective of the RL agent is to

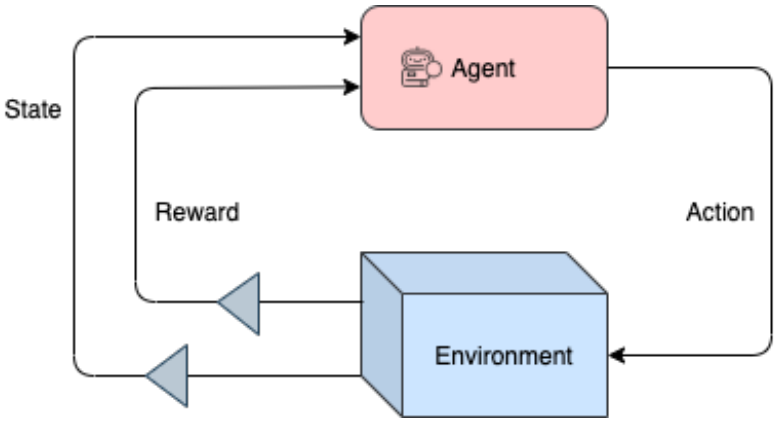


FIGURE 2.19: Reinforcement Learning cycle [36]

learn from trial and error using an optimal policy to receive that action state and reward. This reward is given to an agent to perform the best action for that state. An optimal policy is a Q table for classical RL which has the mapping values from states to actions defined by the Markov decision process. There are various components that exist in the RL problem such as Policy, Value function, and the Bellman Optimal condition will be discussed in this chapter.

**Markov Decision Process**

A Markov Decision Process (MDP) is defined by:

- **A set of states, s** - This is the optimal position of the agent at a specific time in the environment.
- **A set of actions, a** - This is a specific action performed by the agent under the policy.
- **A scalar reward, r** - This is the reward that will be given to the agent after the action is performed.

- **Transition probability, p** - The probability that the agent will make a move/transit from one state to the other state.

- **The reward function, R** - This reward function is a value that the agent receives after performing an action. The value can be positive or negative based on the actions.

**Policy,** $\pi$ - It is defined as a mapping or a Q-table which consists states to actions values.

**Return, G** -It is the sum of rewards that an agent receives from the policy so that it maximizes its rewards.

**Value Function,** $V_\pi(s)$ - It is the expected sum of rewards that the agent will receive from the policy of the *goodness""* of the state. It has state value function and action value function.

**Bellman Equations** - This is the formulating equation of how the RL calculates the reward function (refer equation 2.6).

$$v(s) = \mathbb{E}\left[R_{t+1} + \gamma v\left(S_{t+1}\right) \mid S_t = s\right] \tag{2.6}$$

If the transition probabilities and the reward functions are known, the Bellman equation can be solved using using this approach, also known as Dynamic programming. If the probabilities are unknown and the methods are known as model-free algorithms and if the probabilities are unknown then methods like model based algorithms can be used.

**Exploration and Exploitation:** The agents either explores the environment randomly or exploits the Q-table. The agent tries to make an action which has never done before during training is called the exploration part of the RL agent's behaviour and the then finds the optimal policy. But if the agent has a policy already defined then it uses this policy and the value functions to make the action in the environment which is called exploitation. If the agent doesn't explore and fails to achieve better actions for that state, this is called the the exploration-exploitation dilemma. There are two exploration approaches an RL can take are greedy policy and Boltzmann policy.

### 2.3.2 Types of RL

#### Model free methods

Model-free methods are the methods which are applied to an RL problem which does not require any model of the environment. There are different types of model free methods which specify either on the value function which tries obtain the maximum returns or the policy in order find the optimal policy necessary for the agent in an RL environment [36]. Before we get into the equation we need to understand that in the Markov decision process we have two types of policy, one is a deterministic policy which is mapping from state to action and the other is stochastic policy which is the distribution over the actions for each state.
Value of a policy $\pi$, expected discounted reward if starting in some state and following policy $\pi$ (expressed via Bellman equation)

$$V^{\pi}(s) = R(s) + \gamma \sum_{s \in \mathcal{S}} P\left(s' \mid s, \pi(s)\right) V^{\pi}\left(s'\right) \tag{2.7}$$

Optimal value function (value function of optimal policy $\pi^{\star}$, policy with highest value)

$$V^{\star}(s) = R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in S} P\left(s' \mid s, a\right) V^{\star}\left(s'\right) \tag{2.8}$$

- **Value function based Methods:** These are the methods that are based on learning the value function $\pi$ or $\pi^{\star}$, some of the algorithms are Temporal difference(TD), state-action-reward-state-action(SARSA) algorithm and Q learning (here Q refers to the Q-table that consists of the state-action pair) [35]. TD and SARSA are called the on-policy setting since they are trying to learn the value of the policy to generate actions. And the Q-learning is the off-policy setting which are trying to learn a different policy to generate actions.

- **Policy search methods:** These are the methods that directly learn or approximate the optimal policy $\pi^{\star}$. Its a way to avoid forming MDP and the value function but instead focus on the policy. A simple policy search procedure (fitting machine learning models) is used in order to approximate the policy value which is the return, that is the expected sum of the sum of rewards when an action is performed by the agent in

the environment. There are methods like policy gradient method and REINFORCE algorithm under the policy search method.

- **Actor critic methods:** These are the methods which directly store the policy known as the actor, actor is the policy since its the one by whom the action is performed. Critic is the value function because it the value in which the policy is based upon and it falls under the on-policy setting [37]. It helps in the computation of the actions and learns the policies better. This type of algorithm have the ability to store the actor as well as the critic in a tabular format. Function approximators is a space which can be used for mapping to the value functions, but with the Actor-critic algorithm, it decides the direction in which the gradient ascent instead of the value function reducing memory therefore reduces the variance.

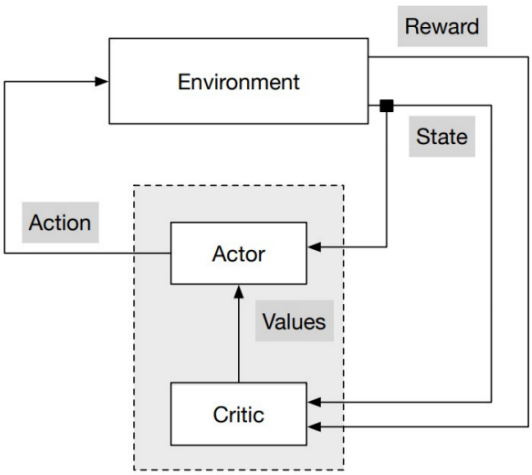


FIGURE 2.20: Actor-Critic Architecture [1]

### Model based methods

The model free methods do not specify the model of the environment or it model environment is unknown. But Model-based methods are simply the models that either use the information it obtains from the environment which improves the algorithm and efficiency i.e if we do not know the MDP then it simply estimates it from the data itself [35]. There are two approaches of handing this, first one is the build the model of the environment itself from the value function and policy and the second is the learn the model of the environment by performing trial and error experiments.

- **Dyna-Q:** This type of architecture tries to directly integrate model learning, planning and RL as the learning model and infers Q-table itself to learn the what is the maximum reward [38]. There is something called as a rollout and this constantly updates the Q- functions in the table of the function approximator.

- **Real Time Model Based Architecture(RTMBA):** RTMBA tries to parallelize the processes of model learning, planning and direct RL [39]. This is an improvement over Dyna-Q algorithm as it allows to make an action without waiting. It works by having three threads: model learning, planning and action thread. So the model learning waits for the action to perform, updated the table, plans a rollout and then the action thread makes an action based on the Q-table. RTMBA is able to perform better in terms in efficiency and learning.

### 2.3.3 Limitations

The are some major limitations in classical RL, which is the the model-based methods if the MDP model itself isn't accurate then the policy itself when applied becomes sub optimal or the policy fails to perform. The use of function approximators in RL increases the training time making it computationally expensive. The efficiency of the MDP model relies majorly on the amount of account it needs and being able to scale to higher dimension seems next to impossible in classical RL. Scalability to extent to a larger dimensional Q-table of continuous state-action pair is often accomplished with the help of deep reinforcement learning which will be elaborate in the research methodology section.

# Chapter 3

# Research Methodology

In this chapter, Research Methodology will be addressed in detail and illustrated section by section, starting with the problem formulation which states what precisely is the framework trying to solve, along with the Preliminaries which gives a broad description about the previous work on solving similar problems and the limitations of such methodologies.

The second section gives a brief overview of the various popular and state-of-the-art Deep Reinforcement Learning algorithms that are chosen as candidate agents for the Optimis framework which will be bench-marked in the evaluation chapter.

The third section explains the datasets involved in this approach with the description and how it is collected and pre-processed.

In the implementation section, the architecture of the Optimis framework is laid out. Then, the training and the prediction workflow is outlined along with the various components of the framework.

Finally, the experiment setup is described, which consists of an custom Open AI Gym environment along with a description of the two popular RL libraries Stable Baselines 3 and RLlib and the rationale for choosing the PPO implementation in RLlib is given.

## 3.1 Problem formulation

Introduction to "when to post" problem[1] is already understood in the earlier section which was first deduced by [40] which is the goal to help social

[1] https://sproutsocial.com/insights/best-times-to-post-on-social-media/tw-times

media users on attaining visibility by remaining on the feed the longest. So deciding when to exactly post to gain attention of the required community to finally reach the target audience. Finding out the optimal time to post for a particular person might not be able to reach the audience because in case the user is trying to target a high influence without them following cannot reach to their feed according to how the social media works. Therefore targeting the community and finding the right time to post then would form a right viral effect to reach the target users i.e by finding out the low traffic regions for the post to be made. So in this thesis, the Optimis framework is developed, which will be focusing on not just a particular user but the community it represents. This part of the influence maximization or also called as smart broadcasting problem to be solved by using Deep Reinforcement learning where the user acts as an agent and posting is the action the user generates, and network users using the hashtag would be the feeds of that forms the environment. This environment generates the feedback and the reward would be the post that stays in the visibility of the audience for the longest amount of time. In conclusion the problem boils down to finding out the DRL algorithm to gain the maximum rewards, giving us the output of the best Optimal time.

### 3.1.1 Preliminaries

#### Previous work

In similar context to smart broadcasting using various models, a recent line of work [22],[26],[32],[41],[6],[7] has stated the alternate use of MTPPs to develop adaptive models using optimal control. Even though previous states the use of MTPPs in this area, it still entirely solve the above problem. The limitations of such models have been explained in the literature review section. But to be brief. Most of the model out there are using neural networks which in turn prevents them to use state of the art neural network in deep learning since it make assumptions of the conditional intensity and the mark distribution of the MTPPs. Using classical RL also has its own limitations as we would need more memory in dimensionality of the space. In this research, Deep reinforcement learning overcomes the drawbacks by taking both the agents actions as well as the feedback it receives from the environment are an random discrete events in continuous time. It is found to be useful in various domains like stock trading strategies [42], social media information systems such as learning applications, spread of fake news. The

goal of Optimis would be,

- Finding the low traffic region of the temporal point graph which would indicate the best time to post.
- Maximize the arbitrary reward function for the agent as to optimal policy so that the state is choose would be the best at that particular time.
- The reward function also depends on the feedback it receives form the environment, So using a policy gradient method to solve the smart broadcasting problem makes it feasible.
- The deep neural network, here a transformer based neural network would be used as the function approximator which is embedded in the RL.

## 3.2 A Deep Reinforcement Learning Approach

Deep Reinforcement Learning (DRL) gained a lot of traction in the past few years especially in the field of Artificial Intelligence (AI) community. The difference between classical RL and DRL is the use of deep neural networks as function approximators for the policy that is in the RL framework, see figure 3.1. It has received a lot of praise because it can achieve high human like performance especially in the fields of robotics, Autonomous driving, Atari games and many others. The other difference that the scalability of dimensional space it can extend to. Getting into the framework of Deep reinforcement learning as they have successfully applied in the fields of events that happen in a continuous time frame, various algorithms are used based of the type of problem statements. A salient point to note is that hyper-parameter tuning of model-free or model-based methods can cause significant change in the performance of the way the model finally gives the results [43]. This is also true for the construction of reward function, which should be judiciously done. The following sections give a brief overview of the various popular and state-of-the-art Deep Reinforcement Learning algorithms that are chosen as candidate agents for the Optimis framework.

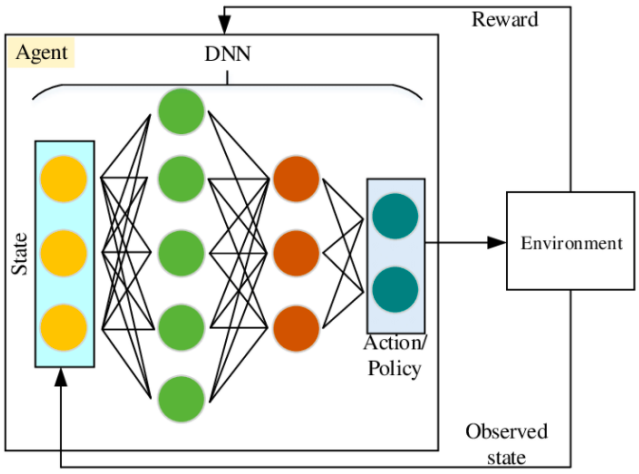


FIGURE 3.1: Schematic structure of DRL [44]

### 3.2.1 DRL Algorithms

**Continuous action space based Algorithms**

Most of the RL algorithms are used based on the action spaces - either continuous or discrete action spaces. Some of them are listed below. For evaluation purpose we will be using the discrete action space algorithms.

***1. Deep Deterministic Policy Gradient Algorithm***

Deep Deterministic Policy Gradient (DDPG) is an improvement [45] and also, combined the ideas of Deep Q-Network (DQN) and Deterministic policy Gradient (DPG). Primarily the network is trained to solve continuous problems using off-policy over a small sample and by using the DQN's replay buffer to store previous transitions.Utilizing the target networks of DQN to stabilizes the learning model, hence training the DQN during the temporal difference backups. DDPG is an off-policy Algorithm that normally works on continuous action space.

***2. Trust region policy optimization Algorithm***

TRPO is a model free- gradient policy method that works on an on-policy algorithm. Since is a large policy change for policy gradient methods which destroys training [46], TRPO can updates the policies by taking a single update after the trajectory, this helps in using both the discrete and continuous action space. TRPO is also sample efficient, refer figure 3.2 which shows the model can learn from small sample size and provide good model performance.

***3. Actor-Critic using Kronecker-Factored Trust Region Algorithm***

ACKTR is a type of policy gradient method with an optimization which has a trust region [47]. It is similar to other DRL algorithms but it uses the neural network on actor-critic algorithm to estimate the policy and anther network to estimate the value function. This trust region optimization works better then TPRO; The models learns both discrete and continuous state action spaces. There are 2 optimization- first order and second, first is using stochastic gradient descent optimization and the second uses the approximate curvature. It reduces computational complexity.


FIGURE 3.2: Sample efficiency for DRL

**Discrete action space based Algorithms**

From the above figure we can see the how the sample size can effect the algorithm's performance. Actor critic methods are in the ball park of a median size sample size which works better in terms when they are an off-policy based method.

***1. Deep Q-Network***

DQN stands for deep Q-network was the first RL algorithm used, where a neural network was used as a function approximator. In performed so well in terms of playing a game with human level accuracy and even out-performed the champions. The DQN takes in the frames as input and outputs the state values for each action. There are mainly 4 techniques [48]. Experience replay is able to store information needed to learn such as the state transitions , the rewards and the actions as well i.e data which is needed to update the Q-table(mapping state-action pair). The target network works with the parameters which are replaced during learning, such as weights are updated and hyperparametrized. Clipping rewards clips the score where the rewards are clips depending on how the rewards are handled. Finally the skipping frame where the frames as inputted after DQN calculated the Q values. The clipping rewards reduce stability during learning and skipping frames reduces computational cost and gathers more experiences.

***2. Advantage Actor Critic***

A2C is known as Advantage Actor Critic, It follows the policy gradient method and it is the asynchronous version of A3C, synchronous which means that the it waits for the actor to make the action ans only then updated the policy therefore helping the computational efficiency. A2C method learns the model every step directly and by using the value function together. The main difference between A2C and DQN is that DQN uses the replay buffer while the actor-critic method doesn't use ti and learns the model and its values every step [49]. Advantage here refers to the policy value which represents the state and its action probability, because of this value as the expected output that is determined by the critic to the actor.

***3. Proximal Policy Optimization Algorithm***

Almost all the gradient methods out there are shown successful breakthroughs in the field of deep learning but the results aren't that great. If the step-size choice is too small then the RL progress is too slow and if it is too large, it is overwhelmed by the noise and are overly sensitive to the sample size, which means gradient decent methods are have very poor sample efficiency. Researchers have found many other like TRPO and Actor Critic Algorithms to solve such drawbacks to optimize the policy update. But TRPO in fact isn't very compatible with the parameter sharing between the value function and the policy, comparison over PPO with other DRL algorithms are shown in figure 3.3. That's where PPO comes to the picture, OpenAI is a AI research and development company that released this new class of RL for the purpose of ease of use and better performance. It is implemented by using the cost function and then the gradient decent is run. Relatively there is very little fine tuning and it is very sample efficient. Convergence problem seen in gradient methods were solved using this algorithm.

**Learning Parameters for PPO**

PPO by the name itself says that the there is a policy optimization which means that this approach consists of constraints as a penalty being used as an optimizer to reach the objective as well as a PPO-clip which clips the objective. PPO is an on-policy algorithm, this algorithms works well since its easier to tune and used as a benchmark most of the problems. It has shown better results when comparing to other models and is cost efficient [50].

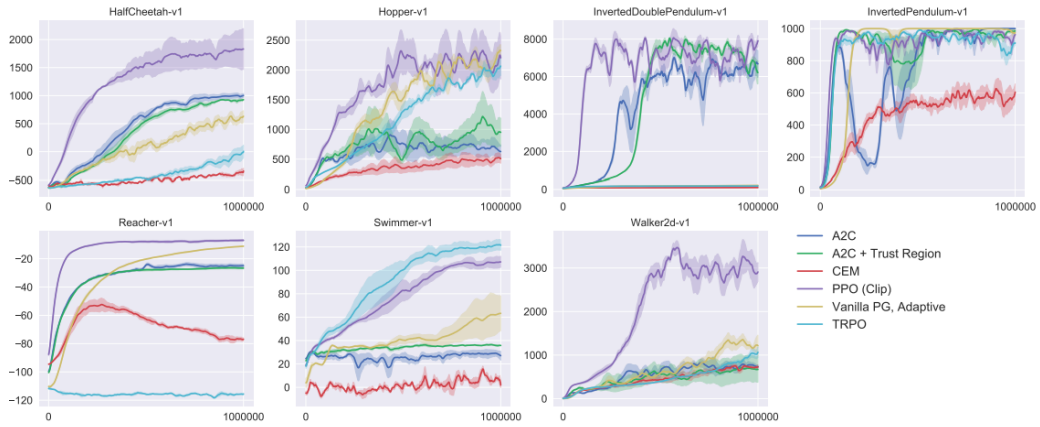


FIGURE 3.3: Comparison of several DRL algorithms [50]

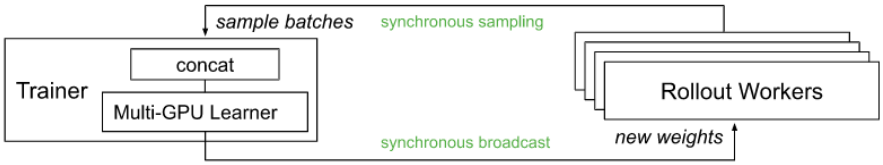


FIGURE 3.4: Architecture of PPO [50]

***Minorize-Maximization MM algorithm*** - Using the MM algorithm by iteratively maximizing a lower bound function. That means when the lower bound function is increased the expected reward is approximated. This is repeated until the best policy converges. First the lower bound value needs to be found so that such an optimization can take place. There are two ways such an optimization can happen, one is the line search and the other is the trust region.

**1. Line search:** Line search is like the gradient decent methods that most DRL uses since it is fast and simple in optimization of the objective function but the if the step-size is too small then the training takes forever and if if it too large then there is noise which is the learner wont learn and directly fail from the bad policy. these gradient methods use the line search and since it is a on-policy method, the next action is made from the current step using the same bad policy update therefore directly decreasing performance.

**2. Trust region:** PPO uses trust region where it determines the maximum step size it needs to explore and after locating the most optimal point of exploration within the region it trusts, it starts and resumes the search from there, So the region can be dynamically adjusted according to the policy from

the new and old if there is a large divergence. In PPO, the policy is changed with the help of KL divergence, see equation 3.1. KL means Kullback–Leibler divergence which measures the probability of one distribution over the other.

$$\mathrm{L}^{\mathrm{CLIP}}(\theta) = \hat{E}_t \left[ \min \left( r_{\mathrm{f}}(\theta) \hat{A}_t, clip \left( r_t(\theta), 1 - \varepsilon_s 1 + \varepsilon \right) \hat{A}_{\mathrm{t}} \right) \right] \tag{3.1}$$

$\theta$ - policy parameter
$\hat{E}_t$ - empirical expectation over time
$r_t$ - ratio of the probability of the policies
$\hat{A}_t$ - estimated advantage at time t
$\varepsilon$ - hyperparameter

The hyperparameter is usually either 0.1 or 0.2 and using this equation is a way the policies are controlled. By removing the KL penalty and implementing a stochastic gradient decent helps to conduct this trust region update to be more adaptive in nature.

**Algorithm of PPO**
PPO is an approach of the actor critic methods where it uses two neural networks, one for the actor and the other is the critic, The model that is the actor model which takes part of the learning task based on which actions to be performed by the agent under a observed environment. The action predicted by the actor will be given to the critic model, If the action done is positive then the environment will send the response in terms of a reward, if negative then the its given a negative reward.So the goal of the critic model is to learn and evaluate the rewards as to get the maximum returns at the end. PPO is one of various methods that enforce KL constraint and there are two variants for it

**1. Adaptive KL Penalty:** Algorithm 1 shows the PPO with Adaptive KL Penalty which states policy update solves unconstrained optimization problem. Initial KL penalty is not that important since it adapts quickly.

**2. Clipped Objective:** Algorithm 2 shows the PPO with Clipped Objective which states the new objective function is the addition of the soft constraint over the gradient decent optimizer which is the KL penalty and by using the $\varepsilon$ as the hyperparameter and the policy update $\theta_{k+1}$.

**Algorithm 1** PPO with Adaptive KL Penalty

**Input:** initial policy parameters $\theta_0$, initial KL penalty $\beta_0$, target KL-divergence $\delta$

**for** $k = 0, 1, 2, \ldots.$ **do**
Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$ Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta \| \theta_k)$$

by taking $K$ steps of minibatch SGD (via Adam)

**if** $\bar{D}_{KL}(\theta_{k+1} \| \theta_k) \geq 1.5\delta$ **then**
$\beta_{k+1} = 2\beta_k$
**else if** $\bar{D}_{KL}(\theta_{k+1} \| \theta_k) \leq \delta / 1.5$ **then**
$\beta_{k+1} = \beta_k / 2$
**end if**
**end for**

**Algorithm 2** PPO with Clipped Objective

**Input:** initial policy parameters $\theta_0$, clipping threshold $\epsilon$

**for** $k = 0, 1, 2, \ldots.$ **do**
Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$ Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CUP}(\theta)$$

by taking $K$ steps of minibatch SGD (via Adam), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \tau \sim \pi_k \mathrm{E}\left[\sum_{t=0}^{T}\left[\min\left(r_t(\theta)\hat{A}_t^{\pi_k}, clip\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k}\right)\right]\right]$$

**end for**

## 3.3 Dataset

### 3.3.1 Description of the Dataset

The datasets that are required for the DRL problem are stored in a repository for use. For the purpose of this problem, the requirements of synthetic data was used to find the best solution after which the model was worked on real-world twitter dataset. There is a lot of work for prepossessing the dataset received in streams since they have to be properly analysed and rechecked if the values are in the right order and the Optimis agent can predict the next best time to post.

#### Synthetic Dataset

Synthetic data is the data that is computer generated by the use of simulations. There are various ways to create simulated data or synthetic data. one by using hawkeslib[2] and the other is a ticks[3] library. Synthetic datasets can be very useful to meet specific requirements that the original data failed to. Its especially used to find solutions to theoretical problems where data isn't accessible and also to protect confidential data.

**1.** ***Library - hawkeslib***

```
from hawkeslib import UnivariateExpHawkesProcess
import numpy as np
uv = UnivariateExpHawkesProcess()
#mu, alpha, beta are the controlling parameters
timestamps = uv.sample(1000)
```

The above code demonstrates how the simulated data is captured using hawkeslib library and the figure 3.5 shows the plot of the data of the timestamps which was used in out model to initially get the results. Its an easy to use library for implementing vanilla Hawkes process to provide simulated temporal point processes that can be used to run models such as Hawkes, Poisson, Bayesian Poisson process and test them for statistical purposes and now they are used in various domain including modeling seismic activity.

[2]https://hawkeslib.readthedocs.io/
[3]https://x-datainitiative.github.io/tick/modules/hawkes.html

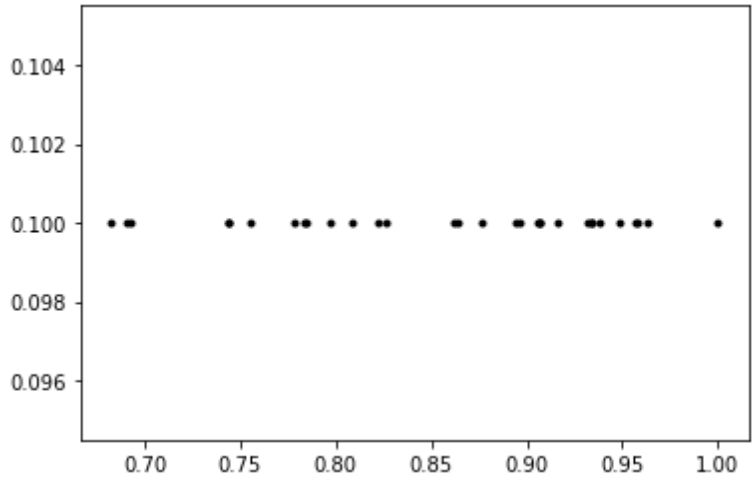


FIGURE 3.5: Simulated data using hawkeslib

**2. *Library -tick.hawkes***

```
from tick.hawkes import SimuPoissonProcess
# intensity represents the specified background intensity value
poi = SimuPoissonProcess(intensity, end_time=run_time, verbose=False)
poi.simulate()
poi.timestamps
```

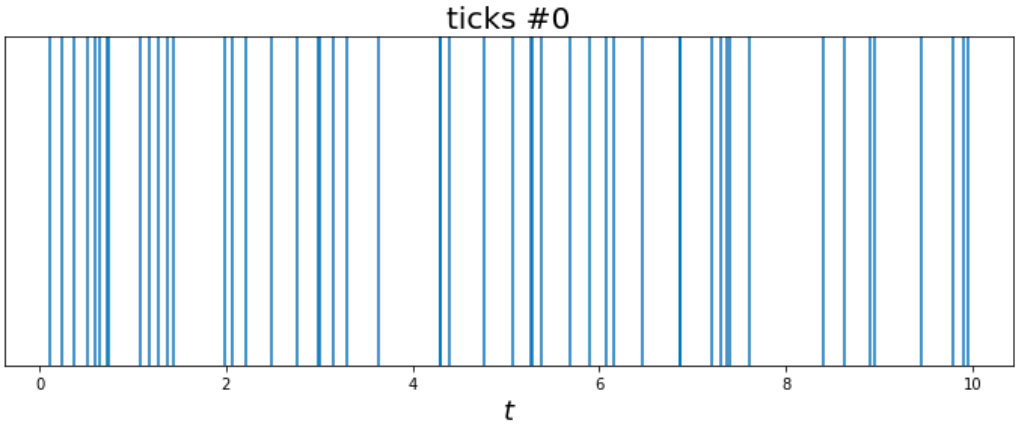


FIGURE 3.6: Simulated data using tick

The above code demonstrates how the simulated data is captured using tick library and the figure 3.6 shows the plot of the temporal data. tick.hawkes is a variation of the simulated data from the tick library itself specially used in Hawkes process

**Real-world Twitter Dataset**

Twitter dataset is a live real world dataset colected from twitter using a twitter scrapper called "tweepy". Tweepy is used to get live streams of data of a particular community using the required hashtag(#). The input streams are then collected i.e. almost one year of data of the community feed. The data then has to be cleaned and processed to get the required timestamps.

### 3.3.2 Data Acquisition

Data Acquisition is the collection of data from twitter and there are numerous ways which we can collect the data from this particular social media platform such as tweepy, twython and various others. Twitter data mining using python is used to fetch the twitter object that contains the tweet text, the user name, when was the tweet posted, at what time and even the location. There is so much information we can extract from the tweet object. Figure 3.7 shows a workflow of how the data is collected using tweepy and then the timestamps of each feed is collected and used in the model later.

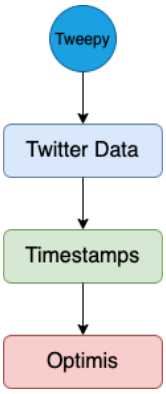


FIGURE 3.7: Data acquisition workflow

***Library - tweepy***

Tweepy is an open-source python package that is used fairly often by developers for an easy access of twitter data made available by twitter. Tweepy handles all this mess on behalf of the user making the application prone to errors and provides a way for developers to communicate with the Twitter API.

## 3.4 Implementation

The implementation section describes the Optimis framework, and its training and prediction workflow. Then it goes through the various key modules of the framework: namely the Data Preprocessor and the LSTM Forecaster.

In the experimental setup sub-section, the implementation of the custom Open AI Gym environment is outlined along with the the RL library chosen for the PPO implementation along with its rationale.

### 3.4.1 Optimis framework

The diagram below, figure 3.8 depicts the framework of Optimis and how it works to solve the "when to post" problem using the RL approach.

The Optimis Agent is a PPO algorithm that is working in a custom Open AI Gym environment that consists of a window of discrete time steps. The PPO agent can make either of two actions, 'post' or 'no post'. The agent can work under constrained budget (for instance, make only 3 posts per week) or given a threshold action penalty (make a post that will fetch an air-time of at least 30 minutes).

The PPO agent consists of an actor who talks to the critic which is the value function of the policy (value of the state and action), here being the custom LSTM custom policy network which acts as a function approximator since it can be scaled to higher dimension.

Since the social network environment has large amounts of sequential data, with strong dependencies on past events, recurrent neural network architectures, especially LSTMs are suited for modeling the state.

The policy which is the critic will evaluate the goodness of the action and the reward it holds back to the environment and the agent makes a decision based on it.

The Figure 3.8 shows the outline of the Optimis framework. Optimis agent (PPO) is the decision maker here. The discrete temporal points are fed to the Gym environment, and the state is generated as a sliding window of the

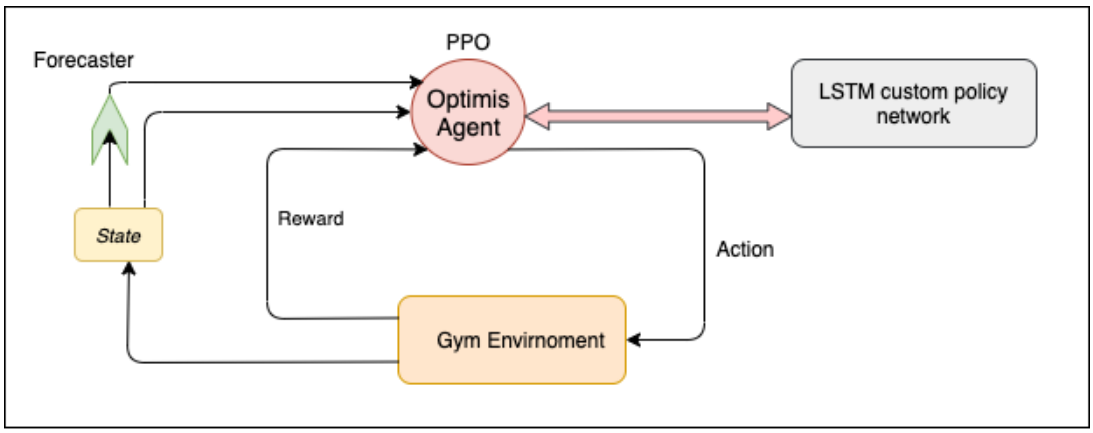


FIGURE 3.8: Framework of Optimis

points. The environment is equipped with preprocessing steps that are required in order to facilitate the learning of the agent. The state (window of preprocessed temporal points) then passes through a forecaster that enables the agent to make better predictions. The PPO agent is equipped with a custom LSTM policy network. Two actions are available at each time-step, post or no post, and the agent receives appropriate rewards for each action based on the user settings. The following describes the training and prediction workflow of Optimis.

### Training workflow

The training procedure is illustrated in Figure 3.9. The datasets once collected are saved and gone through the pre-processing steps to get the timestamp of the desired events, which is then sent into the the model for training. During training, the sliding window for the time steps, the reward criteria for posting behaviour and the user-defined budget is set first. The Optimis agent then trains on the settings, and finally, after convergence, the PPO's policy, which is essentially as LSTM model is extracted and saved to file.

### Prediction workflow

The prediction workflow is illustrated in the figure 3.10. The saved model which is extracted from the training workflow is then loaded into the Optimis agent during the prediction workflow. The real-time data, in this case twitter, first passes through the preprocessor, then it's loaded into the environment and the state with the sliding window which was used during the training

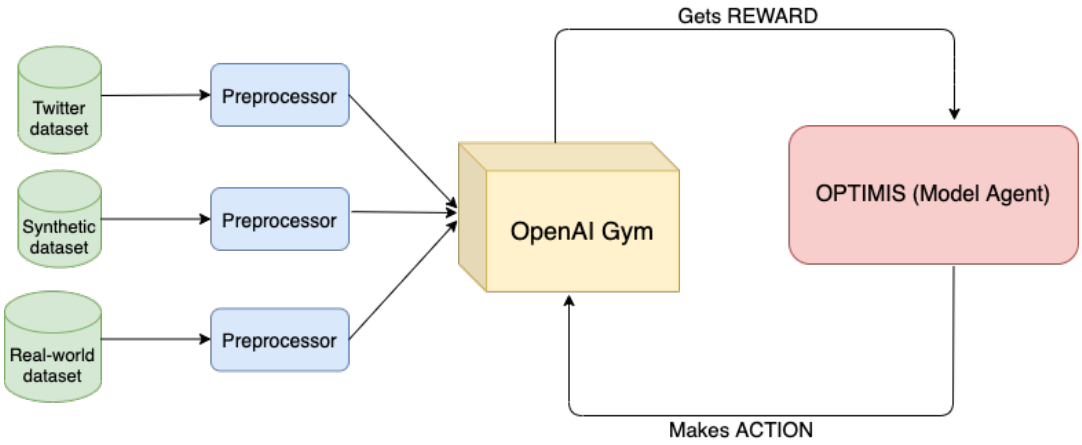


FIGURE 3.9: Training model architecture of Optimis

workflow is implemented. The Optimis agent, with the help of the forecaster module makes a decision on each state. The decision of the agent at each step is the final outcome of the prediction workflow.

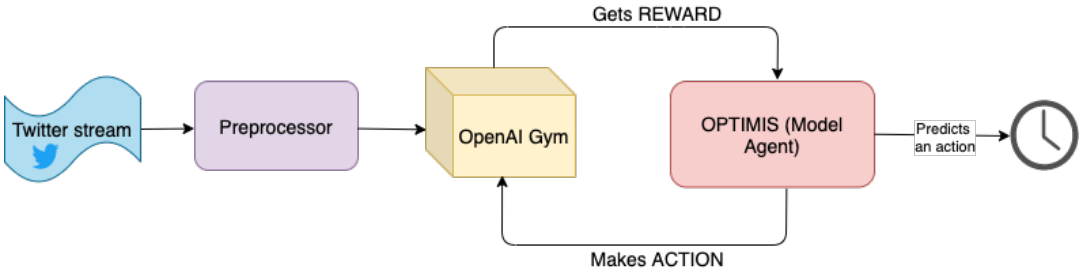


FIGURE 3.10: Prediction model architecture of Optimis

### 3.4.2 Data Preprocessor

For the scope of the thesis, text preprocessing is not employed as the focus is on temporal events therefore only the timestamps of the tweets are required and the tweet text is not considered.

**Converting from discrete to a continuous domain:**

In order to make the data suitable for training in a time-series based RL environment, the data should be first transformed from the discrete domain to a continuous domain. For achieving this, the first-order and second-order time difference of the event points are taken. The preprocessing workflow is a two-step recursive procedure where the first order time difference is

calculated from the original discrete event data, and then the second-order time difference is calculated from the first-order time difference data.

timestamps = [$x_1$,$x_2$,$x_3$,.....$x_n$]
*#1st-order time difference is taking the difference [$x_2$-$x_1$ , $x_3$-$x_2$.....]*
1st-order-td = [$d_1$,$d_1$,$d_1$,.....$d_n$]
*#2nd-order time difference is taking the difference [$d_2$-$d_1$,.....]*
2nd-order-td = [$d_1^1$, $d_2^1$,$d_3^1$.....$d_n^1$]

The first-order time difference of the discrete temporal points ´are essentially the post intervals that exist between two posts (discrete points). By using first-order difference, it becomes possible to issue a threshold value that facilitates the user's demand for producing agent actions that guarantee post survival times of greater than some value. For instance, if the user wants to make sure that the post survives for a period of at least 60 minutes, a threshold value can be set in order to ensure that the agent produces posts that are of minimum survival time of 60 minutes.

The second-order time difference is taken when a particular budget is not set. For instance, the demand is to simply maximize post survival time. The second-order time difference is a representation of the 'acceleration' or direction of the change of the post interval length. A positive value of second-order time difference indicates that the post intervals are expanding, and this would be a hint for a optimal time to place a post.

### 3.4.3 LSTM Forecaster

The forecaster module helps the Optimis agent make better decisions and converge faster during training. The forecaster introduces an additional layer of analysis to the state that the agent observes. The hidden patterns in the temporal state might require a significant number of iterations for the PPO agent to identify. The Forecaster module helps alleviate this problem by training beforehand on the training data and fitting a regression model on the state values. The LSTM forecaster, during the prediction workflow, adds to the state by forecasting the next state value, thereby enabling the PPO agent to utilize the forecast data to make better decisions.

### 3.4.4 Custom Gym Environment

OpenAI Gym [4] library is a toolkit for developing custom environments as well as comparing algorithms with standard environment. Since it is an open source library, it gives access for to many researchers in the field where they can perform comparison and playing of the standard environments [51]. It can be computed using TensorFlow, Theano and even Pytorch for numerical computations.

There are basic concepts to be known in the RL framework, one being the environment itself and the other is defining the agent which is the algorithm. So the core of the OpenAI gym library is *gym.Env*.

| **gym.spaces** | **example** | **Description** |
|---|---|---|
| gym.spaces.Discrete | Discrete(10) | Specifies a space of discrete events |
| gym.spaces.MultiDiscrete | MultiDiscrete([(1, 3), (0, 5)]) | Multiple dimensional where each dimension containing discrete points |
| gym.spaces.Box | Box(np.array((-1.0, -2.0)), np.array((1.0, 2.0))) | multidimensional continuous spaces with bounds |
| gym.spaces.Tuple | Tuple([Discrete(5), Box(np.array([0,0]) | Tuple of spaces such as discrete or continuous states with containing the number of actions |

TABLE 3.1: gym.spaces modules

Most environments have two attributes which are action space and the observation space, they contain the gym space classes which are action spaces and state spaces defined by the classes module. The list of spaces are outlined in the table 3.2.

[4] https://github.com/openai/gym

### Custom gym environment

Optimis uses a custom gym environment within which the state, action, reward is defined. The skeleton of the custom environment code is shown below.

```
class CustomEnv(gym.Env):
  def __init__(self):
          self.post_times = twitter_timestamp
          self.action_space = gym.spaces.Discrete(2)
          self.observation_space = spaces.Box(low=0, high=500,
                                        shape=(5,), dtype=np.float32)
          self.num_actions_made = 0
          self.max_allowed_actions = 3
          self.min_allowed_reward_threshold = 80
  def step(self, action):
          ...
          return state, reward, done, info
  def reset(self):
          ...
          return state
```

There are three base class methods within *gym.Env* which are the following:

1. **reset:** It resets the environment state.

2. **render:** Returns the frame of the environment. This function is typically implemented for robot simulations and Atari games, see figure 3.11.

3. **step:** It the step that the agent takes in the environment, which in turns returns the rewards after the action is performed in the environment.

### Defining the state, action and reward

According to the "when to post" problem, the RL environment should be built is such a way as to learn from the environment mimicking how a human would function when given the very same problem.

During the CustomEnv class initialisation, the twitter timestamps are saved

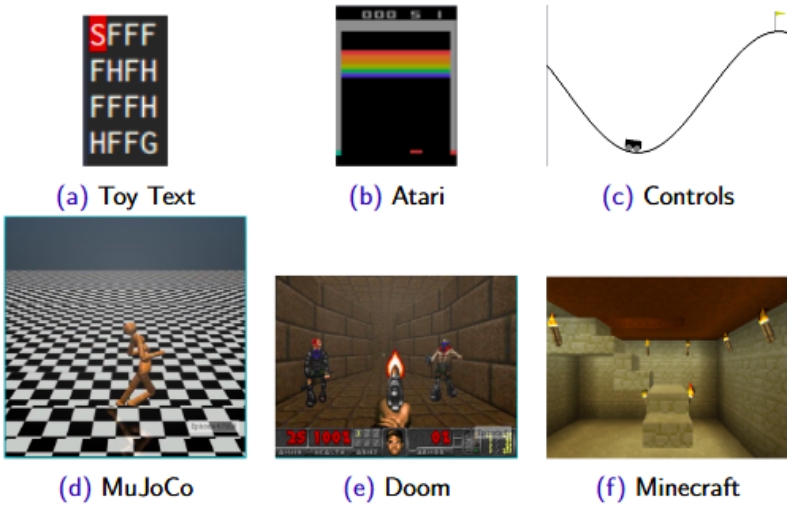


FIGURE 3.11: Domain Example OpenAI [51]

in the the class attribute *self.post_times*. The action space has to be defined. In this case, the agent makes two discrete actions: post or no post. Therefore *gym.spaces.Discrete(2)* is assigned to *self.action_space*.

Next, the observation space is defined as *spaces.Box(low=0, high=500,shape=(5,), dtype=np.float32)*. This is where the state is contained. The shape of the observation space is the sliding window length of the temporal points that exist in the state. *low* and *high* denote the minimum and maximum allowed values of the state vector.

An internal attribute *self.num_actions_made* keeps a count of the number of posts made till a particular time. The user-defined attribute *self.max_allowed_actions* sets the posting budget if the user wishes to. Another user-defined attribute is the *self.min_allowed_reward_threshold* which scales the reward down to only allow posts that are greater than the specified post survival time.

If the budget is defined (max allowed actions), then the second-order time difference is enabled to calculate the n best post times. Else, if the min allowed time is defined, then the first-order time-difference preprocessing is done and the reward discount of threshold is applied.

The first and second-order difference of the discrete temporal points are calculated in the initialisation part of the environment class. The code snippet

is as follows:

```
self.max_post_time = post_times[-1]
self.post_times = post_times/self.max_post_time
self.post_times_1st_diff = np.diff(self.post_times)
self.post_times_2nd_diff = np.diff(self.post_times_1st_diff)
```

The LSTM forecaster is trained in the initialisation of the environment as well:

```
# define input sequence
raw_seq = self.post_times_1st_diff
# choose a number of time steps
n_steps = 19
# split into samples
X, y = split_sequence(raw_seq, n_steps)

n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))

print('training the forecaster..')
self.lstm_forecaster = Sequential()
#hyperparamters
self.lstm_forecaster.add(LSTM(50, activation='relu',
                                input_shape=(n_steps, n_features)))
self.lstm_forecaster.add(Dense(1))
#compiling the model
self.lstm_forecaster.compile(optimizer='adam', loss='mse')
#fitting the lstm forecaster model
self.lstm_forecaster.fit(X, y, epochs=100, verbose=1)
```

Two methods of reward calculation is employed. In the first case, when a minimum allowed posting time is defined, then the reward is as follows:

```
# 1st method of how the reward function is calculated
reward = self.post_times_1st_diff[self.current_step-1]
        - self.min_allowed_reward_threshold
```

In the case when no constraints are provided, the reward function simply becomes the next value of the second-order time difference.

```
#2nd method of how the reward function is calculated
reward = self.post_times_2nd_diff[self.current_step-2]
```

**Illustration of the generalisability**

Note that the CustomEnvironment is designed in such a way that it can be extended to incorporate other applications as well. For instance, in the case of a trading application, the user can set *max allowed trades = 3* per week, and to control the risk and expected profit made, the *min allowed reward threshold* can be set to a high number to make sure that the agent only performs a 'buy' action when the expected reward is high.

### 3.4.5 Optimis Agent Implementation

***Stable-baselines3***

Stable-Baselines3 (SB3) is a set of algorithms that have been implemented in PyTorch for the RL community, there have many algorithms for the research community to replicate and work with. Before which existed the SB2, it had limited features compared to SB3. Identifying good baselines to build projects is very important so that new ideas can be created on top of existing ones. It is used more like a comparison tool to play round it and benchmark the algorithms. Beginners in the field of research can learn how the environment functions by tweaking the parameters and along with the change in rewards by how if effects the stability of the model and whether it is sample efficient.

| Features | SB2 | SB3 |
|---|---|---|
| State of the art RL methods | ✓ | ✓ |
| Custom environments | ✓ | ✓ |
| Custom policies | × | ✓ |
| Custom callback | × | ✓ |

TABLE 3.2: Features of SB3

The main idea of using SB3 is for research purpose so that it can be used as a toolkit without having the trouble to implement it from scratch. its more like a plug and play mechanism for comparison. RL Baselines3 is a collection of pre-trained Reinforcement Learning agents using SB3. It provides basic

scripts for training, evaluating agents, tuning hyperparameters, plotting results and recording videos. SB3 is able to maintain a stable core providing the most latest features like Truncated Quantile Critics or Quantile Regression DQN. It has built-in policy network such as MLPPolicy network and CNNPolicy network to work on.

```
import gym
from stable_baselines3 import PPO, DQN, DDPG, A2C, TRPO
#Running the PPO with recurrent policy
optimis = PPO(LstmPolicy, env, verbose=0)
optimis.learn(total_timesteps=25000)
```

***RLlib***

RLlib[5] is an open-source library [52]. Comparing SB3, RLlib is found to be training very fast and the parameters can be easily tuned and using the same gym environment, the results have proved to be far better. RLlib can also show the tensor-board for visualization purpose i.e. monitor resource use and the training performance. It has built-in policy network such as RNN Policy network and LSTM Policy network to work on. RLlib's uses that data to store in GPU memory using the GPU. It improves the performance.

| **Features** | **RLlib** |
|---|---|
| State of the art RL methods | ✓ |
| Custom environments | ✓ |
| In-built policies | ✓ |
| Custom policies | ✓ |
| Custom callback | × |

TABLE 3.3: Features of RLlib

```
from ray import tune
from ray.rllib.agents.ppo import PPOTrainer
#Running the PPO with recurrent policy
Optimis = tune.run(PPOTrainer, config={"env":'my_env',
                                       'num_workers': 1, 'num_gpus': 1,
                                       "model": {"use_lstm": True}})
```

[5]https://github.com/ray-project/ray

### 3.4.6 Running example

Lets take a few example high level strategies that can a social media user would like to achieve. Three examples strategies are listed here: the first one is just a condition that the post should survive as the first post in social media for at least half an hour. The second one is that the user wants to maximize the post visibility of her three posts, and no minimum time required. The third one is: the user wants to make 3 posts in the next hour and make sure that they get an average airtime of at least 30 seconds. Now this is a complex strategy, and we can pick this for the running demo.

```
#setting strategies
#unit of time is seconds
env.set_constraints(action_penalty=30,
                    max_allowed_actions=3,
                    time_limit=3600)
```

So, breaking it down, the user wants to post 3 times, that means the maximum allowed actions is 3. Then the time limit is specific as 1 hour, so the agent should make these 3 posts in the coming hour, therefore the time limit is set as 3600 seconds or 1 hour. Then, the post time should be at least 30 seconds, so, an action penalty of 30 is set to make sure the agent is reward only for posts made that last for more than 30 seconds as seen in the code.

From the figure 3.12 above, an illustration of a running example is shown. First the tweets are collected, and then a sliding window of 10 tweet timestamps are used for each timestep. The sliding window moves 1 step forward into time at each timestep. The tweets are collected, and then a sliding window of 10 tweet timestamps are used for each timestep. The sliding window moves 1 step forward into time at each timestep. The tweets are collected, and then a sliding window of 10 tweet timestamps are used for each timestep. The sliding window moves 1 step forward into time at each timestep. Then the first order time differences are sent to the forecaster that predicts the next value of the time difference. Note that the forecaster is trained during the environment initialisation with the training data. The forecaster here predicts a value of 22, the combination of state first order time differences and the forecasted value becomes the new state. The combination of state first order time differences and the forecasted value becomes the new state. The reward

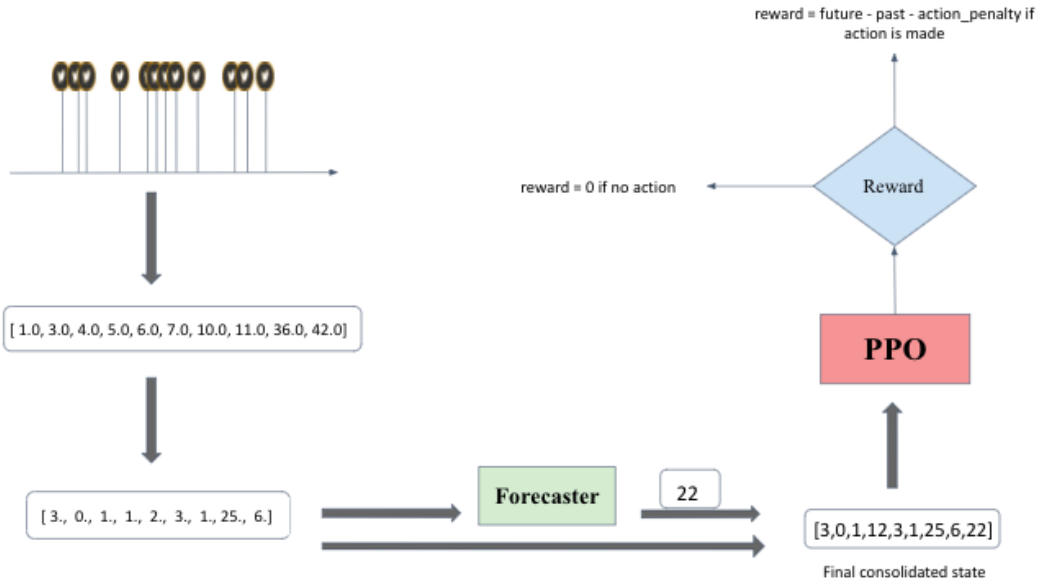


FIGURE 3.12: Illustration of a running example

calculation function calculates the reward value based on the action made by the agent. If the agent makes no action, then the reward obtained is 0. If the agent makes an action, the reward is the future timestamp value minus the current timestamp value minus the action penalty, which is 30 in this case. The reward calculation only happens in the training step and during prediction, the agent simply predicts an action. If the agent makes an action, the reward is the future timestamp value minus the current timestamp value minus the action penalty, which is 30 in this case. The reward calculation only happens in the training step and during prediction, the agent simply predicts an action.

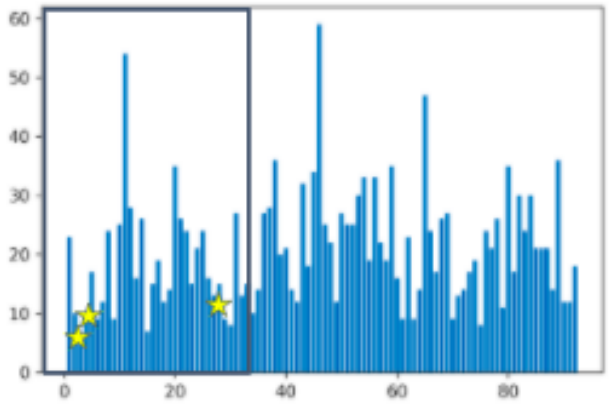


FIGURE 3.13: Graph for the running example

As you can see from the figure 3.13, it shows the timesteps the agent predicted at. One point to note that in the twitter dataset which is chosen, the maximum interval between points is around 54 seconds, so from the difference between current post time and next post time, it is observed that the agent DOES make posts that last for more than 30 seconds. Furthermore, the agent made only 3 posts, and it stopped posting after 1 hour, or 3600 seconds. The graph seen in the right hand side, is a visualisation of the 9000 tweets grouped by 100 timestamps. The stars denote the approximate location of the posts that the agent made. The bounding box denotes the time limit under which the agent is required to make those 3 posts.

**Summary**

In summary, a PPO-based reinforcement learning agent is implemented in order to learn "when to post" in social media in order to get the maximum content visibility and reach. The PPO agent is trained in an custom OpenAI-based Gym environment made to accommodate the discrete point process data. The environment internally preprocesses the point process data and transforms it into a format and domain which is suitable for training the RL agent. The environment connects to two data sources: twitter stream, and synthetic point process library, hawkeslib. Furthermore, the RL agent is equipped with an LSTM-based forecaster that forecasts the next point as a guiding point for making better predictions. RLlib's implementation of PPO is used as Optimis agent. Therefore by adding a layer of customization over the API like action penalty, max allowed action and time limit, many high-level posting strategies are possible and by implementing an environment with preprocessing techniques that make point processes suitable for simple in-built agents like PPO, DQN, and A2C.

# Chapter 4

# Evaluation

In this chapter, the optimis agent is benchmarked on the basis of a number of chosen controlling parameters which will be explained in the section below. Two datasets are primarily used for evaluation: A synthetic event data (created by the hawkeslib library) and a real-world data (twitter data of machine leaning hashtag for a week).

## 4.1 Control parameters

Controlling parameters are the variables used for evaluating the performance of the experiment settings. The following subsections explain each of the control parameters in detail. The chosen control parameters remain the same for both the datasets chosen for evaluation.

### 4.1.1 Algorithms considered

These are the considered algorithms for the purpose of evaluation. PPO, A2C and DQN are chosen as the candidate algorithms for the Optimis agent. RLlib implementations of these algorithms are used. For benchmarking against parametric models, the state-of-the-art broadcasting algorithm Redqueen is considered.

### 4.1.2 Action Penalty

Since the RL algorithms are sensitive to the reward functions employed, the reward values obtained after each action of the agent should be carefully constructed. For Optimis, one of the major challenges of the reward construction is that if the reward is too sparse, the agent will not try to post at all, and if there is no negative reward for making posts that does not survive for a sufficiently long period of time, the agent will engage in a spam-like behaviour

of posting all the time in order to simply maximize the reward. In order to avoid both these extreme behaviours, an action penalty parameter is introduced, which adds a negative value to each posting action taken so that the agent will be more judicious while trying to post. No negative reward is assigned for not posting at all.

### 4.1.3 Evaluation metric

The mean survival time is chosen as the metric which is used to judge the performance of the agent. Mean survival time is calculated as the average time for which a point or a tweet survives as the most recent post in the timeline. The longer the tweet stays as the top as the most recent, the better the value of that point in time. The objective of the Optimis agent is to decide "when to post" in order to maximize the mean survival time.

## 4.2 Evaluating on the Synthetic dataset

### 4.2.1 The synthetic data

For simulating data we use Hawkes library called hawkeslib, and the controlling parameters such as mu, alpha, beta have a value of .1,.3,.1 respectively to produce the synthetic data which will be used in the Optimis framework.

```
from hawkeslib import UnivariateExpHawkesProcess

mu, alpha, beta = .1, .3, .1
uv = UnivariateExpHawkesProcess()
uv.set_params(mu, alpha, beta)
post_times = uv.sample(1000)
```

#### The baseline metrics

The produced synthetic data consists of discrete points with intensities increasing and decreasing at various (as can be seen in the figure 4.1). There are a total of 152 discrete points upto 1000 units. The average survival time of a point is 0.0065, which will be considered as the baseline. The goal of the agent will be to maximize the survival time of a point by placing it at a point in the timeline where the intensity of point distribution is low.

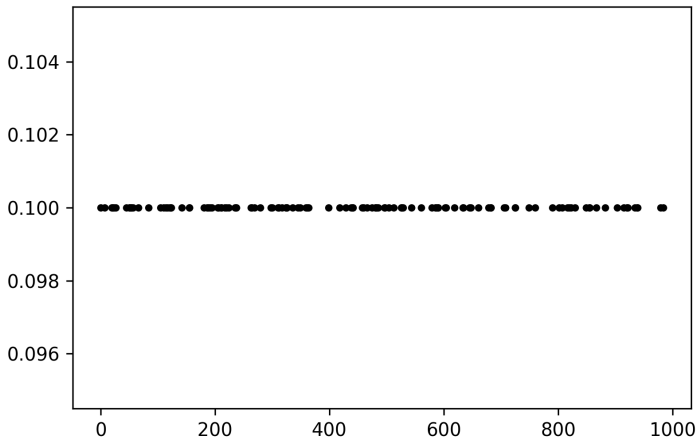


FIGURE 4.1: Simulated data using hawkeslib

### 4.2.2 The results

Table 4.1 shows the evaluation of the three algorithms PPO, A2C and DQN with various action penalties and their reported mean survival times over the simulated data.

PPO with action penalty of 0.003 reports the highest mean survival time of 0.0901 which is approximately 10x higher than the baseline. DQN shows competitive results as well. A point to be noted is that DQN trains much faster than the rest of the algorithms and is more robust to the action penalty value choice, whereas A2C shows the highest sensitivity to the action penalty parameter. The mean survival time for some configurations indicate that the agent did not make a single post during the testing episodes.

## 4.3 Evaluating on the Twitter dataset

For the purpose of evaluating on a real-world setting, twitter data was chosen to perform the analysis. Tweets containing the hashtag *"#machinelearning"* was extracted from the date 1st march 2021 to 5th march 2021, a total of 5 days.

The twitter data consists of 1913 tweets. The first 100 tweets are selected for illustration (as shown in figure 4.2). As can be observed, the tweet posting pattern in this particular community exhibits a similar pattern as to that

| Algorithm | Reward | action penalty | mean survival time |
|---|---|---|---|
| Baseline | | | 0.0065 |
| RedQueen [6] | | | 0.0236 |
| DQN | 0.248 | 0.002 | 0.0316 |
| | 0.0.248 | 0.002 | 0.0203 |
| | 0.0487 | 0.0025 | 0.0254 |
| | 00.145 | 0.00275 | 0.0284 |
| | 0.166 | 0.003 | 0.0416 |
| | 0.228 | 0.00325 | 0.0401 |
| | 0.0672 | 0.0035 | 0.0570 |
| | 0.0406 | 0.0035 | 0.0301 |
| | 0.116 | 0.004 | 0.0315 |
| A2C | 0.247 | 0 | 0.0211 |
| | 0.277 | 0.0005 | 0.0242 |
| | 0 | 0.00075 | 0 |
| | 0 | 0.001 | N/A |
| | 0 | 0.002 | N/A |
| **PPO** | 0.196 | 0.002 | 0.0316 |
| | 0.188 | 0.0025 | 0.0385 |
| | 0.0536 | 0.00275 | 0.0665 |
| | **0.0644** | **0.003** | **0.0901** |
| | 0.204 | 0.00325 | 0.0407 |
| | -0.0015 | 0.0035 | N/A |

TABLE 4.1: Evaluation results on synthetic data

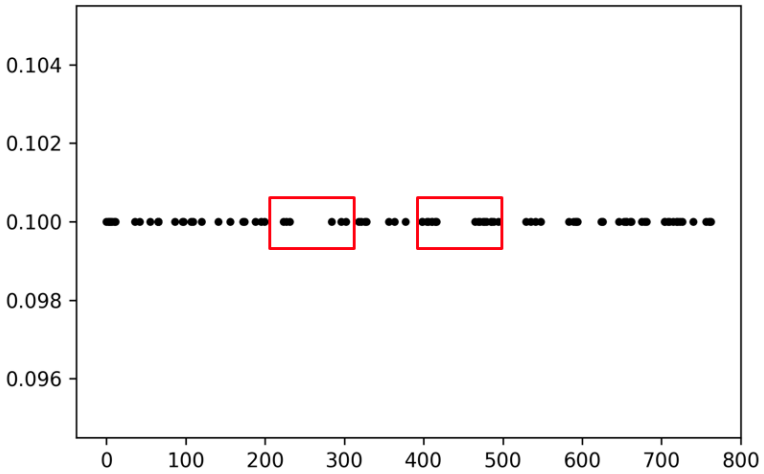


FIGURE 4.2: Twitter highlight data

which is observed in the simulated Hawkes data, i.e, regions of increasing and decreasing intensities in posting activity. The motivation of the agent should be to place posts in the low-traffic regions, which are highlighted in

red bounding boxes.

Since 1913 tweets are posted over a period of 5 days, that corresponds to a mean post survival time of 3.76 minutes. The normalised mean post survival time is calculated by taking 5 days as 1 unit, thus making it equal to 0.00052. This value is considered as the baseline survival time for the experiment.

## 4.4 Evaluating on running example

The evaluation on the running example -

- Fetched tweets related to #machinelearning
- Total number of tweets: 1913
- Duration: 2.5 hours
- Average interval between two points: 4.79 secs
- Top 5 intervals found in the dataset: 54.0, 53.0, 49.0, 45.0, 45.0 secs

The values in the bracket are the mean reward obtained after training. Note that if the value is negative, the agent won't post a single action during prediction and no posts are denoted NA. From the below table 4.3, it is clear that the preprocessing steps along with the PPO algorithm shows the best results.

| Algorithm | No preprocessing | First-order time difference | Forecaster | Forecaster+FO |
|---|---|---|---|---|
| DQN | (-32.3) N/A | (7.1) N/A | (8.5) 32.9 | (11.2) 34.1 |
| A2C | (-12.8) N/A | (-13.4) N/A | (-4.8) N/A | (-4.2) N/A |
| PPO | (8.6) 35.7 | (42.6) 45.1 | (56.9) 44.9 | (63.2) 45.8 |

FIGURE 4.3: Evaluation for running example

### 4.4.1 The results

Table 4.2 shows the evaluation of the chosen state-of-the-art parametric model Redqueen [6] against the three algorithms RL-based algorithms PPO, A2C and DQN with various action penalties and their reported mean survival times over the real-world twitter data.

| Algorithm | Reward | action penalty | mean survival time |
|---|---|---|---|
| Baseline | | | 0.00052 |
| RedQueen [6] | | | 0.0019 |
| DQN | 0.198 | 0.0000325 | 0.0012 |
| | 0.210 | 0.000035 | 0.0014 |
| | 0.007 | 0.0000375 | 0.0011 |
| | 0.006 | 0.00002 | 0.0009 |
| | 0.018 | 0.00004 | 0.0013 |
| A2C | 0 | 0.00001 | N/A |
| | 0 | 0.00002 | N/A |
| | 0 | 0.00003 | N/A |
| **PPO** | 0.003 | 0.00001 | 0.0011 |
| | 0.0064 | 0.00002 | 0.0028 |
| | **0.00778** | **0.00003** | **0.0033** |
| | 0.232 | 0.0000325 | 0.00149 |
| | 0.241 | 0.000035 | 0.0017 |
| | 0.003 | 0.0000375 | 0.00171 |
| | 0.00014 | 0.00004 | N/A |
| | 0 | 0.00002 | N/A |

TABLE 4.2: Evaluation results on Twitter data

PPO with action penalty of 0.00003 reports the highest mean survival time of 0.0033 which is approximately 10x higher than the baseline. The mean survival time for some configurations indicate that the agent did not make a single post during the testing episodes. Using A2C algorithm, the agent did not make a single post while testing over various action penalties. DQN's mean survival time remained in the same range 0.0006-0.0015 and was unable to go beyond the threshold. The model was trained between 100,000 to 500,000 time-steps as the collected twitter data is large compared to the synthetic hawkes data, thus requiring more training.

# Chapter 5

# Conclusion

In this thesis, a reinforcement learning based-framework is designed with the purpose of:

1. Enabling social media users to find the optimal time to post in an online social environment like Twitter or Facebook so that their messages are broadcast widely and effectively.

2. Assisting researchers in building reinforcement-learning algorithms on discrete temporal settings by providing a framework which is scalable to other related domains.

The following key components which helped achieve the aforementioned objectives:

1. **A recurrent-policy-based PPO agent** that is capable of interpreting 2nd-order time difference of discrete events in order to maximize post survivability.

2. **LSTM-based forecaster** module that assists the PPO agent in its interpretation of the temporal environment.

3. **An Open AI Gym Environment** that supports temporal point process based settings with inbuilt support for basic preprocessing techniques such as 1st order and 2nd order time difference transformations. The environment is generic enough to accomodate a number of applications based on discrete events.

The Optimis framework is evaluated on 2 settings: 1) A real-world twitter dataset containing a particular hashtag #machinelearning and 2) A synthetic environment created by a hawkes process.

On both the twitter and synthetic hawkes process generated data, the agent performs significantly better (approximately 10 times better) than the baselines average post survival time, thereby indicating the effectiveness of Optimis framework. The successful learning on the simulated environment indicates that Optimis framework can be applied to a broader spectrum of applications involving temporal point processes.

## 5.1 Future scope

The Optimis framework is designed keeping in mind the scalabilty and the possibility of extensions based on it. As mentioned in the motivation, a slightly tangential but significant use case would be in tackling fake news where a different module can detect the spread of fake news in a social network and the Optimis agent can be used as an countermeasure to effectively broadcast the fact-checked results to the affected community.

Other significant applications that can be built on top of Optimis can range from trading applications, personalised memory retention software that can model a user's memorization capability and quantitative trading.

# Bibliography


[1] *Automated Fact Checking*. en. URL: https://fullfact.org/about/automated/ (visited on 03/15/2021).

[2] Wei Chen, Yajun Wang, and Siyu Yang. "Efficient influence maximization in social networks". In: vol. 199-208. Jan. 2009, pp. 199–208. DOI: 10.1145/1557019.1557047.

[3] Siqi Liu and Milos Hauskrecht. "Nonparametric Regressive Point Processes Based on Conditional Gaussian Processes". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/9cc138f8dc04cbf16240daa92d8d50e2-Paper.pdf.

[4] *Learning with Temporal Point Processes*. URL: http://learning.mpi-sws.org/tpp-icml18/ (visited on 03/15/2021).

[5] L Jatiningsih, Respatiwulan, Y Susanti, S S Handayani, and Hartatik. "The parameter estimation of conditional intensity function temporal point process as renewal process using Bayesian method and its application on the data of earthquake in East Nusa Tenggara". In: *Journal of Physics: Conference Series* 1217 (May 2019), p. 012063. ISSN: 1742-6588, 1742-6596. DOI: 10.1088/1742-6596/1217/1/012063. URL: https://iopscience.iop.org/article/10.1088/1742-6596/1217/1/012063 (visited on 03/24/2021).

[6] Ali Zarezade, Utkarsh Upadhyay, Hamid Rabiee, and Manuel Gomez Rodriguez. *RedQueen: An Online Algorithm for Smart Broadcasting in Social Networks*. 2016. arXiv: 1610.05773 [stat.ML].

[7] Utkarsh Upadhyay, Abir De, and Manuel Gomez-Rodriguez. *Deep Reinforcement Learning of Marked Temporal Point Processes*. 2018. arXiv: 1805.09360 [cs.LG].

[8] Shruthi Gurudath. *Parametric and Non-Parametric Models In Machine Learning*. en. Sept. 2020. URL: https://medium.com/analytics-vidhya/parametric-and-nonparametric-models-in-machine-learning-a9f63999e233 (visited on 03/15/2021).

[9] Rafael Lima and Jaesik Choi. *Hawkes Process Kernel Structure Parametric Search with Renormalization Factors*. May 2018.

[10] Jeffrey Heaton. "An Empirical Analysis of Feature Engineering for Predictive Modeling". In: (Jan. 2017).

[11] *Non-Parametric Model*. May 2019. URL: https://deepai.org/machine-learning-glossary-and-terms/non-parametric-model (visited on 03/15/2021).

[12] Martijn Otterlo and Marco Wiering. "Reinforcement Learning and Markov Decision Processes". In: *Reinforcement Learning: State of the Art* (Jan. 2012), pp. 3–42. DOI: 10.1007/978-3-642-27645-3_1.

[13] Kowshik chilamkurthy. *Understanding Point Processes*. en. May 2020. URL: https://towardsdatascience.com/understanding-point-processes-6e3d2f6c5480 (visited on 03/01/2021).

[14] Achim Klenke. "The Poisson Point Process". In: *Probability Theory*. London: Springer London, 2014, pp. 543–561. ISBN: 9781447153603 9781447153610. DOI: 10.1007/978-1-4471-5361-0_24. URL: http://link.springer.com/10.1007/978-1-4471-5361-0_24 (visited on 03/15/2021).

[15] Eric W. Weisstein. *Hawkes Process*. en. Text. URL: https://mathworld.wolfram.com/HawkesProcess.html (visited on 03/01/2021).

[16] Patrick Laub, Thomas Taimre, and Philip Pollett. "Hawkes Processes". In: (July 2015).

[17] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. *On the difficulty of training Recurrent Neural Networks*. 2013. arXiv: 1211.5063 [cs.LG].

[18] Ralf C. Staudemeyer and Eric Rothstein Morris. *Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks*. 2019. arXiv: 1909.09586 [cs.NE].

[19] Shuai Xiao, Junchi Yan, Mehrdad Farajtabar, Le Song, Xiaokang Yang, and Hongyuan Zha. *Joint Modeling of Event Sequence and Time Series with Attentional Twin Recurrent Neural Networks*. 2017. arXiv: 1703.08524 [cs.LG].

[20] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. 2018. arXiv: 1803.01271 [cs.LG].

[21] Fisher Yu and Vladlen Koltun. *Multi-Scale Context Aggregation by Dilated Convolutions*. 2016. arXiv: 1511.07122 [cs.CV].

[22] Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. "Recurrent Marked Temporal Point

Processes: Embedding Event History to Vector". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1555–1564. ISBN: 9781450342322. DOI: 10.1145/2939672.2939875. URL: https://doi.org/10.1145/2939672.2939875.

[23] Hongyuan Mei and Jason Eisner. *The Neural Hawkes Process: A Neurally Self-Modulating Multivariate Point Process*. 2017. arXiv: 1612.09328 [cs.LG].

[24] Michael Rabin and Dana Scott. "Finite Automata and Their Decision Problems". In: *IBM Journal of Research and Development* 3 (Apr. 1959), pp. 114–125. DOI: 10.1147/rd.32.0114.

[25] J. Derek Tucker, Lyndsay Shand, and John R. Lewis. "Handling missing data in self-exciting point process models". en. In: *Spatial Statistics* 29 (Mar. 2019), pp. 160–176. ISSN: 22116753. DOI: 10.1016/j.spasta.2018.12.004. URL: https://linkinghub.elsevier.com/retrieve/pii/S2211675318301726 (visited on 03/01/2021).

[26] Qiang Zhang, Aldo Lipani, Omer Kirnap, and Emine Yilmaz. *Self-Attentive Hawkes Processes*. 2020. arXiv: 1907.07561 [cs.LG].

[27] Kehai Chen, Rui Wang, Masao Utiyama, and Eiichiro Sumita. "Recurrent Positional Embedding for Neural Machine Translation". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 1361–1367. DOI: 10.18653/v1/D19-1139. URL: https://www.aclweb.org/anthology/D19-1139.

[28] Sepp Hochreiter. "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6 (Apr. 1998), pp. 107–116. DOI: 10.1142/S0218488598000094.

[29] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[30] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[32] Simiao Zuo, Haoming Jiang, Zichong Li, Tuo Zhao, and Hongyuan Zha. *Transformer Hawkes Process*. 2021. arXiv: 2002.09291 [cs.LG].

[33] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].

[34] Han Yang, Kaili Ma, and James Cheng. *Rethinking Graph Regularization for Graph Neural Networks*. 2020. arXiv: 2009.02027 [cs.LG].

[35] blackburn. *Introduction to Reinforcement Learning : Markov-Decision Process*. en. Aug. 2020. URL: https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da (visited on 03/03/2021).

[36] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Second edition. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018. ISBN: 9780262039246.

[37] Prashanth L.A., Prasad H.L., Shalabh Bhatnagar, and Prakash Chandra. "A constrained optimization perspective on actor–critic algorithms and application to network routing". en. In: *Systems & Control Letters* 92 (June 2016), pp. 46–51. ISSN: 01676911. DOI: 10.1016/j.sysconle.2016.02.020. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167691116000566 (visited on 03/04/2021).

[38] R.S. Sutton and A.G. Barto. "Reinforcement Learning: An Introduction". en. In: *IEEE Transactions on Neural Networks* 9.5 (Sept. 1998), pp. 1054–1054. ISSN: 1045-9227. DOI: 10.1109/TNN.1998.712192. URL: http://ieeexplore.ieee.org/document/712192/ (visited on 03/04/2021).

[39] Gen Endo, Jun Morimoto, Takamitsu Matsubara, Jun Nakanishi, and Gordon Cheng. "Learning CPG-based Biped Locomotion with a Policy Gradient Method: Application to a Humanoid Robot". en. In: *The International Journal of Robotics Research* 27.2 (Feb. 2008), pp. 213–228. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364907084980. URL: http://journals.sagepub.com/doi/10.1177/0278364907084980 (visited on 03/04/2021).

[40] Nemanja Spasojevic, Zhisheng Li, Adithya Rao, and Prantik Bhattacharyya. "When-To-Post on Social Networks". en. In: *Proceedings of*

*the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Sydney NSW Australia: ACM, Aug. 2015, pp. 2127–2136. ISBN: 9781450336642. DOI: 10.1145/2783258.2788584. URL: https://dl.acm.org/doi/10.1145/2783258.2788584 (visited on 03/05/2021).

[41] Ali Zarezade, Abir De, Utkarsh Upadhyay, Hamid R. Rabiee, and Manuel Gomez-Rodriguez. *Steering Social Activity: A Stochastic Optimal Control Point Of View*. 2018. arXiv: 1802.07244 [cs.SI].

[42] Zhuoran Xiong, Xiao-Yang Liu, Shan Zhong, Hongyang Yang, and Anwar Walid. *Practical Deep Reinforcement Learning Approach for Stock Trading*. 2018. arXiv: 1811.07522 [cs.LG].

[43] Xuefei Huang, Seung Hong, Mengmeng Yu, Yuemin Ding, and Junhui Jiang. "Demand Response Management for Industrial Facilities: A Deep Reinforcement Learning Approach". In: *IEEE Access* PP (June 2019), pp. 1–1. DOI: 10.1109/ACCESS.2019.2924030.

[44] Quoc-Viet Pham, Nhan Nguyen, Thien Huynh-The, Long Le, Kyungchun Lee, and won-Joo Hwang. *Intelligent Radio Signal Processing: A Contemporary Survey*. Aug. 2020.

[45] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. *Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971 [cs.LG].

[46] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. *Trust Region Policy Optimization*. 2017. arXiv: 1502.05477 [cs.LG].

[47] Yuhuai Wu, Elman Mansimov, Shun Liao, Roger Grosse, and Jimmy Ba. *Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation*. 2017. arXiv: 1708.05144 [cs.LG].

[48] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[49] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. arXiv: 1602.01783 [cs.LG].

[50] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[51] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. *OpenAI Gym*. 2016. arXiv: 1606.01540 [cs.LG].

[52] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. *RLlib: Abstractions for Distributed Reinforcement Learning*. 2018. arXiv: 1712.09381 [cs.AI].