Learning Meters of Arabic Poems with Deep Learning

A Thesis Presented to the Faculty
of Information Technology and Computer Science
Nile University

In Partial Fulfillment
of the Requirements for the Degree
of Master of Science

By
Moustafa Alaa Mohamed
March 2019

CERTIFICATION OF APPROVAL

Learning Meters of Arabic Poems with
Deep Learning

By
Moustafa Alaa Mohamed

Prof.Ahmed Hassan (Chair) ITCS	Date
Assoc.Prof. Nashwa AbdelBaki (Supervisor) ITCS	Date
Assoc.Prof. Waleed A. Yousef (Co-Supervisor) Computer Science, Helwan University	Date
Prof. Samhaa El Beltagy (Co-Supervisor) Computer Science, Cairo University	Date
Prof. Mohsen Rashwan (Member) Electronics and communication Engineering, Cairo University	Date

ABBREVIATIONS

ML ..... Machine Learning.

DL ..... Deep Learning.

NN ..... Neural Networks.

DNN ..... Deep Neural Network.

RNN ..... Recurrent Neural Networks.

LSTM ..... Long-Short Term Memory.

BILSTM ..... BI Directional Long-Short Term Memory.

GRU ..... Gated Recurrent Unit.

BinE ..... Binary Encoding.

OneE ..... One-Hot Encoding.

TwoE ..... Two-Hot Encoding.

APCD ..... Arabic Poem Comprehensive Dataset.

F1 Score ..... Harmonic Precision-Recall Mean (F1 Score).

ReLU ..... Rectified Linear Unit (ReLU).

CNN ..... Convolutional Neural Network.

ACKNOWLEDGMENT

First and foremost, I would like to thank God for giving me the strength, knowledge, ability and opportunity to undertake this research study and to persevere and complete it satisfactorily. Without His blessings, this achievement would not have been possible.

I would first like to thank my family,especially Mom, whose dream was to see me achieve this degree -God bless her- and Dad, for the continuous support they have given me throughout my studies; I could not have done it without them. Second, I would like to thank my wife Kholoud for all the love, comfort, and encouragement which motivated me to complete this thesis and my master's studies at Nile University.

I would like to thank my supervisor, Associate Professor Waleed A. Yousef, for introducing me to the field of machine learning and guiding me patiently throughout my undergraduate studies and MSc. studies. I am grateful for being given the chance to be part of the Informatics group and to collaborate with such brilliant researchers at Nile University.

I thank the members of my committee, Professors Ahmed Hassan, Samhaa El-Beltagy, Mohsen Rashwan, and Associate Professor Nashwa Abdelbaki, for their valuable time and ideas to enhance this research work.

I thank all my co-authors, without whom this work would not have been possible, especially Taha M. Madbouly and Omar M. Ibrahime for all their hard work and their dedication to complete this work.

I thank Mohamed Al-Aggan for his valuable insights in my research. Thanks also to Mohamed Abdel Aziz who helped me during our study in NU. I have met many great people during my studies. To name a couple of my professors who greatly affected my career, I would like to thank Dr.Ayman Ezzat and, Dr.Sameh Al-Ansari. I also need to thank two of my friends who helped me a lot and mentor me: Deyaaeldeen Al-Mahallawi and Omar Marzouk.

TABLE OF CONTENTS

Dedication i

ABBREVIATIONS i

Acknowledgment ii

Table of Contents iii

List of Figures v

List of Tables vi

Abstract 1

1 INTRODUCTION 2

1.1 Thesis Outline . . . 2
1.2 Arabic Poetry . . . 2
1.3 Deep Learning . . . 2
1.4 Thesis Objectives . . . 3

2 BACKGROUND 4

2.1 Arabic Arud . . . 5
2.1.1 Al-Farahidi and Pattern Recognition . . . 5
2.1.2 Feet Representation . . . 6
2.1.3 Arabic Poetry feet . . . 7
2.1.4 Arabic Poetry Meters . . . 7
2.1.5 Meters Relation . . . 11
2.2 Deep Learning Background . . . 14
2.2.1 Logistic Regression . . . 15
2.2.2 The Neuron . . . 19
2.2.3 The Neural Network Representation . . . 20
2.2.4 Neural Network Computation . . . 20
2.2.5 Recurrent Neural Networks (RNNs) . . . 26
2.2.6 Long Short Term Memory networks (LSTMs) . . . 28
2.2.7 Machine Learning Model Assessment . . . 31

3 Literature Review **34**
3.1 Deterministic (Algorithmic) Approach . . . 34
3.2 English Literature . . . 35

4 DESIGN DATASET **36**
4.1 Dataset Design . . . 37
4.1.1 Data Scraping . . . 37
4.1.2 Data Preparation and Cleansing . . . 38
4.1.3 Data Encoding . . . 38

5 Model Training **41**
5.1 Parameters of Data Representation . . . 42
5.1.1 Diacritics . . . 42
5.1.2 Trimming Small Classes . . . 42
5.1.3 Encoding Techniques . . . 42
5.1.4 Data Representation Matrix . . . 43
5.2 Parameters of Network Configuration . . . 43
5.2.1 Working on Unbalanced data using Weighted Loss . . . 44
5.3 Experiments . . . 45
5.4 Hardware . . . 45
5.5 Software . . . 45
5.6 Implementation Outline . . . 45

6 Results **48**
6.1 Overall Accuracy . . . 48
6.2 Data Representation Effects . . . 48
6.3 Network Configurations Effects . . . 50
6.4 Per-Class (Meter) Accuracy . . . 50
6.5 Encoding Effect . . . 52
6.6 Comparison with Literature . . . 52
6.7 Classifying Arabic Non-Poem Text . . . 52
6.8 Discussion . . . 55
6.8.1 Dataset Unbalanced . . . 55
6.8.2 Encoding Method . . . 55
6.8.3 Weighting Loss Function . . . 55
6.8.4 Neural Network Configurations . . . 55
6.8.5 Model Assessment . . . 56

7 Conclusion and Future Work **57**
7.1 Conclusion . . . 57
7.2 Future Work . . . 58

LIST OF FIGURES

1.1 Thesis Working Steps. . . . 3

2.1 Al-Moshtabeh Meter Group. . . . 12
2.2 Illustrations on how Deep Learning can work based on char level figure presented [1]. . . . 15
2.3 Logistic Regression Function (S-Shape) 15
2.4 Convex and Concave functions examples. . . . 17
2.5 Gradient Descent 17
2.6 Description of neuron's structure this figure from [2] 20
2.7 Neural Networks Example Input layer with N input samples, One hidden Layer, and an Output Layer 21
2.8 Commonly used activation functions including the logistic sigmoid $\sigma(z)$, the hyperbolic tangent $\tanh(z)$, the rectified hyperbolic tangent RelU $Relu(x)$, and linear function 22
2.9 Neural Network Example with Backpropagation Step. . . . 23
2.10 Comparison Between different fitting types 24
2.11 A diagram demonstrating the effects of applying dropout with p = 0.5 to a deep neural networks [3] 25
2.12 Recurrent Neural Networks Loops adapted from [4] 27
2.13 The repeating module in a standard RNN containing a single layer adapted from [4] 27
2.14 The repeating module in an LSTM containing four interacting layers adapted from [4] 28
2.15 LSTM Gates and Configurations adapted from [4] 30

4.1 Arabic dataset Meter per class percentage ordered descendingly on x axis vs. corresponding meter name on y axis all class in the left of the red line (less than 1% assume to be trimmed in some experiments). . . . 38
4.2 Different encoding mechanisms 40

6.1 Overall accuracy of the 192 experiments plotted as 2 vertical rug plots (for the 12 different data representations: $\{Trimming, NoTrimming\} \times \{Diacritics, NoDiacritics\} \times \{OneE, BinE, TwoE\}$), each represents 16 exp. (for the 16 different network configurations: $\{7L, 4L\} \times \{82U, 50U\} \times \{0W, 1W\} \times \{LSTM, BiLSTM\}$). For each rug plot the best model of each of the four cell types —(Bi)LSTM labeled differently. Consistently, BiLSTM was the winner, and its network configuration parameters are listed at the top of each rug plot. . . . 49
6.2 The per-class accuracy score for the best four models with combination of ($\{Trimming\} \times \{Diacritics\}$); the x-axis is sort by the class size as in Figure 4.1. There is a descending trend in the class size, with an exception at *Rigz* 51
6.3 Encoding effect on Learning rate with the best model (1T, 0D, 4L, 82U, 0W, BinE) and when using the two other encodings instead of BinE. . . . 52

54 . . . . distribution. ranges score data Testing 6.4

LIST OF TABLES

2.1 Diacritics on the letter د . . . 4
2.2 Tanween diacritics on the letter د . . . 4
2.3 Arabic Letters used in our research . . . 5
2.4 The ten feet of the Arabic meters. . . . 7
2.5 Meters Group. . . . 12
2.6 Spam vs Normal Email Classifier example . . . 32

3.1 Overall accuracy of this article compared to literature. . . . 34

4.1 Aldiwan scraping output example . . . 39
4.2 Al-Mosoaa Elshearyaa scraping output example . . . 39

5.1 Data Representation Combination Matrix . . . 43

6.1 Top ten experiments results . . . 49
6.2 Sample Article Classification . . . 53

ABSTRACT

People can easily determine whether a piece of writing is a poem or prose, but only specialists can determine the class of poem.

In this thesis, we built a model that can classify poetry according to its meters; a forward step towards machine understanding of the Arabic language.

A number of different deep learning models are proposed for poem meter classification. As poetry is sequence data, then recurrent neural networks are suitable for the task. We have trained three variants of them; LSTM, GRU with different architectures and hyper-parameters. Because meters are a sequence of characters, we have encoded the input text at the character-level, so that we preserve the information provided by the letters succession directly fed to the models. Moreover, we introduce a comparative study on the difference between *binary* and *one-hot* encoding regarding their effect on the learning curve. We also introduce a new encoding technique called *two-hot*, which merges the advantages of both *binary* and *one-hot* techniques.

To the best of the author's knowledge, this research is the first to address classifying poem meters with featureless technique in a machine learning approach, in general, and in RNN featureless based approach, in particular. In addition, the dataset is the first publicly available dataset prepared for the purpose of future computational research.

Chapter 1

INTRODUCTION

In this chapter, we give an introduction and basic background knowledge on Arabic language. Some details on Arabic Poetry history will be provided, then the aim of this thesis will be stated.

1.1 Thesis Outline

This thesis is arranged as follows:

1. Chapter 1 presents some basic introduction and background knowledge on Arabic Poem and its definitions. Additionally, it contains details about the Arabic language and some features used during our work.
2. Chapter 2 presents a background related to Al-Arud, Deep Learning fundamentals,
3. Chapter 3 presents a literature review for the previous work on this topic.
4. Chapter 4 details the dataset acquisition. It introduces the dataset acquisition and encoding, including the essential pre-processing steps, and the justification for their need. Pre-processing steps are data extraction, data cleansing, data format, and different techniques for data encoding. The chapter contains comparisons among the three encoding techniques.
5. Chapter 5 presents model training and experiment design. The chapter also presents the details of the model selection, the architecture and hyper-parameters.
6. Chapter 6 presents the results and discussion. It contains the details of the 192 experiments done in this project. Additionally, the chapter presents the details of results' analysis and discussion.
7. Chapter 7 presents the conclusion and proposed future work.

1.2 Arabic Poetry

Arabic is the fifth most widely spoken language [5]. It is written from right to left. Its alphabet consists of 28 primary letters, 8 letters derived from the basic ones, so the total count of Arabic characters is 36 characters. The writing system is cursive; hence, most letters join to the letter that comes after them, although a few letters remain unconnected.

Arabic poetry(الشعر العربى) is the earliest form of Arabic literature. It dates back to the sixth century. Poets have written poems without knowing exactly what rules render a collection of words a poem. People recognize poetry by nature, but only talented ones can write poems. This was the case until *Al-Farahidi* (718 – 786 CE) analyzed the Arabic poetry; he noted that the succession of consonants and vowels produce patterns or *meters*, which make the music of poetry. He counted fifteen meters; thereafter, a student of him added one more meter to make sixteen. Arabs call meters بحور which means "*seas*". The study of Arabic Poetry Meter Classification is named *Al-Arud* (العَروض). It takes much time to become an expert in this field.

1.3 Deep Learning

Deep Learning, also named Deep Neural Network, is part of Machine Learning algorithms. Deep Learning attempts to simulate the human brain into neural dependency. By using Deep Learning, we can achieve better results when learning from the data. Deep

Neural Network needs a huge amount of data to achieve the expected learning curve and results. It also needs a massive amount of computation to train. We used the Recurrent Neural Network (RNN) to work on the Arabic text and show its ability to achieve outstanding performance on problems of text data. We also used LSTM to solve the long dependency issue in RNN Deep Learning Background.

1.4 Thesis Objectives

This thesis aims to explore the Arabic language feature using established Al-Arud rules. This exploration helps us understand how to use a machine learning model to understand Arabic language features without hand-crafted features or static rule-based models.

Artificial Intelligence currently works to do the human tasks such as our problem here. Our target in this thesis is to achieve the human accuracy which will make it easy for anyone to recognise the meter for any poem without referring to the language experts or to study the whole field.

In this thesis, we explain how to use deep learning to classify the Arabic poem. We also explain in detail the features of Arabic poem and how to deal with these. We explain how anyone can work with Arabic text encoding in a dynamic way to encode the text at the character level and deal with Arabic text features such as the Tashkeel. We were able to achieve classification accuracy score of 0.9638 for classifications of the sixteen Arabic meter classes.

This is a basic step to get deeper into the language and, later not only can we have more robust machine learning models which understand and classify the Poem/poetry, but we can also generate a similar text with the model without breaking the rules. All the previous are first steps toward a deeper understanding of the Arabic language features and rules using deep learning techniques. This research can open ideas about how machine understanding of Arabic Grammar rules leads to new ideas to generate better Arabic text models.

This research is the first to address classifying poem meters via a machine learning approach. It aims to have a deep understanding of Arabic language grammar and features in the Poem meters classification and is a base for further research related to the poem analysis problem.

In this study, we work on Poetry Meter Classification and use the latest technologies to check the class of poem. We also work to achieve near human expert results which make our work a breakthrough in the field concerning the results compared to previous results. Figure 1.1 shows the steps. First, we derived the data from the available sources with labeling. Second, the clean data was transformed and encoded. Finally, the cleaned data was trained, validated, and enhanced into the RNN model.

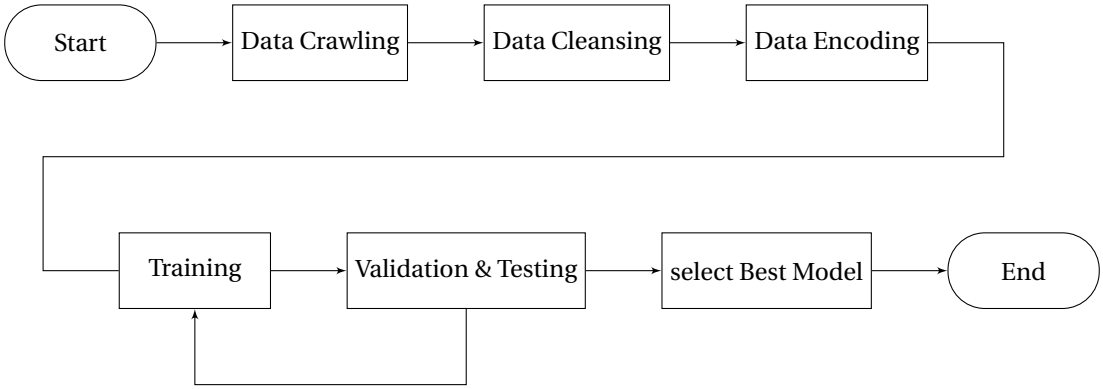


Figure 1.1: Thesis Working Steps.

Chapter 2

BACKGROUND

This chapter gives background knowledge as regards the Arabic Poetry and Al-Arud study rules and details. It also contains basic knowledge of Deep Learning concepts with a Literature Review of the previous work in this problem area.

Each Arabic letter represents a consonant, which means that short vowels are not represented by the 36 characters; for this reason, the need for diacritics arises. Diacritics are symbols that come after a letter to state the short vowel accompanied by that letter. There are four diacritics ◌َ ◌ُ ◌ِ ◌ْ which represent the following short vowels /*a*/, /*u*/, /*i*/ and *no-vowel* respectively; their names are *fat-ha, dam-ma, kas-ra and sukun* respectively. The first three symbols are called *harakat.* Table 2.1 shows the 4 diacritics on a letter. There are two more sub-diacritics made up of the basic four to represent two cases: *Shadaa* and *Tanween.*

Definition 1 ***Shadaa***
To indicate the letter is doubled. Any letter with shaddah (ّ) should be duplicated: the first letter with a constant (sukun) and second letter with a vowel (haraka) [29]; Table 2.1 shows the dal with shadda and the original letters.

Definition 2 ***Tanween***
This doubles the short vowel, and can convert Tanween fathah, Tanween dhammah or Tanween kasrah by replacing it with the appropriate vowel (◌ُ – dhammah, ◌َ – fathah or ◌ِ –kasrah) then adding the Noon letter with consonant to the end of the word [29]. Table 2.2 shows the difference between the original letter and the letter with Tanween

Diacritics	*without*	*fat-ha*	*kas-ra*	*dam-ma*	*sukun*	*letter with Shadda*	*letters without shadaa*
Shape	د	دَ	دِ	دُ	دْ	دَّ	دْدَ

Table 2.1: Diacritics on the letter د

Diacritics	*letter with tanween*	*letters without tanween*
Tanween Fat-ha	دً	دَ+نْ
Tanween Dam-ma	دٌ	دُ+نْ
Tanween Kas-ra	دٍ	دِ+نْ

Table 2.2: Tanween diacritics on the letter د

Arabs pronounce the sound /*n*/ accompanied *Sukun* at the end the indefinite words; that sound corresponds to this letter نْ, and is called *noon-Sakinah.* However, it is just a phone, not a part of the indefinite word; if a word comes as a definite word, no additional sound is added. Since it is not an essential sound, it is not written as a letter, but is written as *tanween* ◌ٍ ◌ً ◌ٌ. *Tanween* states the sound *noon-Sakinah*, but as you have noticed, there are 3 *tanween* symbols. This is because *tanween* is added as a diacritic over the last letter of the indefinite word; one of the 3 *harakat* accompanies the last letter, the last letter's *haraka* needs to be stated in addition to the sound *noon-Sakinah*, so *tanween* doubles the last letter's *haraka.*In this way, the last letter's *haraka* is preserved, in addition to stating the sound *noon-Sakinah*; for example, رَجُلْ + نْ is written رَجُلٌ and رَجُلِ + نْ is written رَجُلٍ

Those two definitions. Definition 1 and Definition 2 help us to reduce the dimension of the letter's feature vector, as we will see in the *preparing data* section. Diacritics make short vowels clearer, but are not necessary. Moreover, a phrase without full diacritics or with just some on some letters is right linguistically, so it is permissible to drop them from the text. In Unicode, Arabic diacritics are standalone symbols, each of which has its own unicode. This is in contrast to the Latin diacritics; e.g., in the set *{ê, é, è, ë, ē, ĕ, ě}*, each combination of the letter *e* and a diacritic is represented by one unicode.

Indices	Letters							
1:8	ا	ب	ت	ث	ج	ح	خ	د
9:16	ذ	ر	ز	س	ش	ص	ض	ط
17:24	ظ	ع	غ	ف	ق	ك	ل	م
25:32	ن	ه	و	ي	ء	آ	أ	ؤ
33:36	إ	ئ	ى	ة				

Table 2.3: Arabic Letters used in our research

2.1 Arabic Arud

Definition 3 ***Arud***
In Arabic, Arud has many meanings (the way, the direction, the light clouds and Mecca and Madinah[1] *[6]. Arud is the study of Arabic Poem meters and the rules which confirm if the Poem has sound meters & broken meters.*

The author of Arabic Arud is *Al-Farahidi* (718 – 786 CE) who analyzed Arabic poetry, then established the succession of consonants and vowels produce patterns or *meters*, which make the music of poetry. He was one of the famous people who know the melodies and the musical parts of speech. He identified fifteen meters. After that, a student of *Al-Farahidi* has added one more meter, making sixteen. Arabs call meters بحور which means "*seas*". Poets have written poems without knowing exactly what rules make a collection of words a poem.

Al-Farahidi established these rules in order to:

- Protect Arabic Poetry from broken meters.
- Distinguish original Arabic Poetry from non-original Arabic Poetry or from prose.
- Make the rules clear and easy for anyone who needs to write a poem.

It is claimed that one-day Al-Farahidi was walking into the metal-market and he noted that the knock of the metals matched the musical sound of the poetry he was saying; then he got an idea to explore the Arud of the poetry.

There are many reasons for the name Arud:

- It is said he put this rules in Arud place العَروض *with fat-ha, not with dam-ma such as the rules name* العُروض between Mecca and Al-Ta'if[6].
- Arud in Arabic is a noun deriving from the verb يعرض meaning "to be assessed". It is said that Any poem should be assessed by Al-Arud rules so, hence Al-Arud [7].

2.1.1 Al-Farahidi and Pattern Recognition

This subsection is our opinion of Al-Farahidi and the method he followed while working on Arabic Poetry Meter Classification.

1. Al-Farahidi thought there is a pattern for every collection of poetry; however, he rigorously worked on this problem. He started analyzing the poetry, adding every group with the same *tafa'il* to the same class.
2. He analyzed various cases for each meter and grouped them together to make the Arud general rules.
3. He revised the meters using these cases and generalized them to fit all poetry.
4. His student subsequently found some poetry which did not fit into any model, leading to a model for a new class.

The essential point which made us admire Al-Farahidi is his research method and his passion for creating a successful model. Also, his model is general and followed all the steps currently any data scientist follows to explore new patterns. Some people state that he died when he was thinking about the problem and hit a wall which caused trouble for him. His death story shows that he was thinking profoundly about this problem. One of the most interest things I found during this research is how Al-Farahid found this pattern, and his way to find new things.

[1] *Mecca and Madinah are two cities in Saudi Arabia.*

2.1.2 Feet Representation

A meter is an ordered sequence of feet. Feet are the basic units of meters; there are ten of these.

Definition 4 ***Feet***
A foot consists of a sequence of ***Sukun*** *(Consonants) represented as (0) and* ***Harakah*** *(Vowels) (/). Traditionally, feet are represented by mnemonic words called tafa'il* تفاعيل.

Feet consist of three parts:

- **Asbab (أسباب)** (the word means reasons in English, but in Arabic means connection between two things). It has two types:
  1. *Sabb Khafeef (*سبب خفيف*)* when the first letter is *harakah* and the second is *sukun* (/0) example (هَبْ, لَمْ).
  2. *Sabb Thakeel (*سبب ثقيل*)* when we have two *harakah* letters (//) example (لَكَ, بِكَ).
- **Awtad (أوتاد)** (English: wedges). It has two types:
  1. *Watd Magmo'a (*وتد مجموع*)* when two *harakah* letters are followed by *sukun* letter (//0) example (مَشَى, عَلَى).
  2. *Watd Mafrouq (*وتد مفروق*)* when two *harakah* separated by a *sukun* (/0/) example (مُنْذُ, مِصْرُ).
- **Fawasek (فواصل)** (means Breaks in English). It has two types:
  1. *Faselah Soghra (*فاصلة صغرى*)* when three harakah letters are followed by a sukun (///0) example (ذَهَبُوا, سُفُنَاً).
  2. *Faselah Kobra (*فاصلة كبرى*)* when four harakah letters are followed by a sukun letter (////0) example (جَعَلَهُمْ).[2]

2.1.2.1 Rules for Arabic Letter Representation

Arabic Arud has one general rule in the poetry representation: only spoken (not written) letters are represented, which means the letters represent only phonetics. The rules resulting from the general rule are outlined below.

- Any letter with *harakah* represented as (/).
- Any letter with *sukun* represented as (0).
- Any letter with *shaddah* represented by two letters; the first will be *sukun* and the second letter will be *harakah*, represented as (0/) example (مُحَمَّد) will be (//0//0).
- Any letter with tanween represented by two letters; the first is *haraka* (/) and the second is *sukun*.
- *Alef* without *hamze* (همزة الوصل) and *wow algmaa* are not represented: example (واعلَموا) will be (/0//0)
- If a letter is not written but spoken, for example (هذا) which includes alef but is not written (هاذا), the representation will be (/0/0).
- *Meem aljamaa* with *harakah* is represented by *mad*, for example (هُمُ) will be (//0).
- *Alef Mad* (آ) will be two letters: *Alef* with *harakah* and *Alef* with *sukun* example e.g. (آدَمُ) will be (/0//).
- If the verse ends with *harkah* we will add *sukun* to it.

Example: (note: in the representation below, the first line is similar to the second but with *Arud* language style).

أرَاكَ عَصِيَّ الدّمعِ شِيمَتُكَ الصّبرُ، أما للهوى نهيٌ عليكَ ولا أمرُ ؟
أراكَ عصيَ دمعَ شِيمَتُكَ صبرو، أما للهوى نهيَ عليكَ ولا أمرو ؟

#	Feet	Scansion	Construction
1	فَعُولُنْ	0/0//	*Watd Magmo'a* (فعو) and *Sabb Khafeef* (لن)
2	مَفاعِيلُنْ	0/0/0//	*Watd Magmo'a* (مفا) and two *Sabb Khafeef*s (عي) (لن)
3	مُفَاعَلَتُنْ	0///0//	*Watd Magmo'a* (مفا), *Sabb Thakeel* (عل) and *Sabb Khafeef* (تن)
4	فَاعِلاَتُنْ	0/0//0/	*Sabb Khafeef* (فا), *Watd Magmo'a* (علا) and *Sabb Khafeef* (تن)
5	فَاعِ لاتُنْ	0/0//0/	*Watd Mafrouq* (فاع) and two *Sabb Khafeef* (لا)(تن) [3]
6	فَاعِلُنْ	0//0/	*Sabb Khafeef* (فا) and *Watd Magmo'a* (علن)
7	مُتَفَاعِلُنْ	0//0///	*Sabb Thakeel* (مت), *Sabb Khafeef* (فا) and *Watd Magmo'a* (علن)
8	مَفْعُولاَتُ	0//0///	two *Sabb Khafeef* (عو)(مف) and *Watd Mafrouq* (لات)
9	مُسْتَفْعِلُنْ	0//0/0/	two *Sabb Khafeef* (مس)(تف) and *Watd Magmo'a* (علن)
10	مُسْتَفْعِ لُنْ	0//0/0/	*Sabb Khafeef* (مس), *Watd Mafrouq* (تفع) and *Sabb Khafeef* (لن)[4]

Table 2.4: The ten feet of the Arabic meters.

2.1.3 Arabic Poetry feet

Arabic poetry feet has ten *tafa'il* تفاعيل (scansion); any peom is constructed from these feet. There are eight from writing (syntax) perspective, but ten in the rules.

Definition 5 ***Meter***
Poetic meters define the basic rhythm of the poem. Each meter is described by a set of ordered feet which can be represented as ordered sets of consonants and vowels [8]. A meter in Arabic is named Bahr (بحر).

ولد الهدى فالكائنات ضياء وفم الزمان تبسم وثناء انشاء
الروح والملأ الملائك حوله للدين والدنيا به بشراء

Definition 6 ***Arabic Verse***
refers to "poetry" as contrasted to prose. While the common unit of a verse is based on meter or rhyme, the common unit of prose is purely grammatical, such as a sentence or paragraph [9]. A verse in Arabic is named Bayt (بيت).

Definition 7 ***Shatr***
A verse consists of two halves, both called shatr, and carries the full meter. We will use the term shatr to refer to a verse's half; whether the right or the left half.

Definition 8 ***Poem***
is a set of verses having the same meter and rhyme.

2.1.4 Arabic Poetry Meters

2.1.4.1 Al-Taweel الطويل

tafa'il

فعولن مفاعيلن فعولن مفاعلن فعولن مفاعيلن فعولن مفاعيلن

Example:

أَمَاوِيَّ إِنَّ المَالَ غَادٍ وَرَائِحٌ وَيَبْقَى مِنَ المَالِ الأَحَادِيْثُ وَالذِّكْرُ
أَماوِيْ يَ إِنْ نَمَا لَ غَادٌ وَرائِحْ وَيبْقَى مِنَ لْمَالِ لَ أَحَادِيـ ـثُ وَذِكْرُو
//0/0/0 //0/0/0 //0/0/0 //0/0/0
فَعُوْلُنْ مَفَاعِيْلُنْ فَعُوْلُنْ مَفَاعِلُنْ

[2] *Some Arab linguistic scientists assume the Faselah Soghra is a combination of Sabb Thakeel and Sabb Khafeef. Similarly, the Faselah Kobra is considered a combination of Sabb Thakeel and Watd Magmo'a. Therefore, they believed there are only two pure feet and combinations of these. In this thesis we assume there are three feet .*

[3] *We separated the letters (ع) and (لا) in (فاع لاتن) to show that this part is Watd Mafrouq and distinguish between this feet and (فاع لاتن) which contains Watd Magmo'a .*

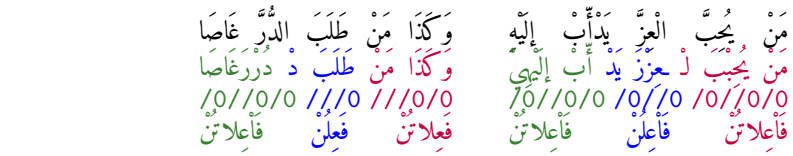
[4] *We separated the letters (ع) and (ل) in (مستفع لن) to show that it ends with a Watd Mafrouq and distinguish between this feet and (مستفعلن) which contains Watd Magmo'a*

2.1.4.2 Al-Madeed المديد

tafa'il

فاعلاتن فاعلن فاعلاتن فاعلاتن فاعلن فاعلاتن

Example:



2.1.4.3 Al-Baseet البسيط

tafa'il

مستفعلن فعلن مستفعلن فاعلن مستفعلن فاعلن مستفعلن فاعلن

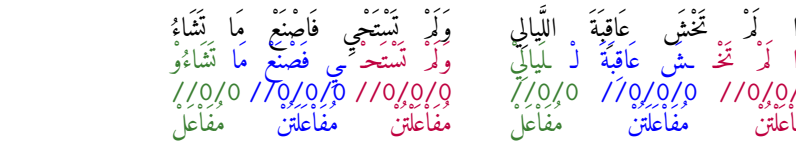
Example:

2.1.4.4 Al-Wafer الوافر

tafa'il

مفاعلتن مفاعلتن مفاعلتن مفاعلتن مفاعلتن مفاعلتن

Example:

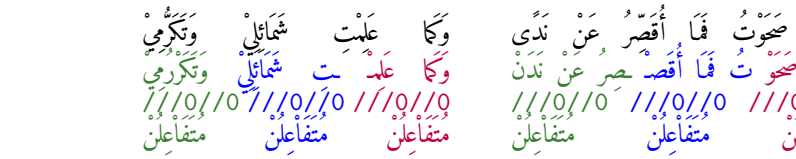


2.1.4.5 Al-Kamel الكامل

tafa'il

متفاعلن متفاعلن متفاعلن متفاعلن متفاعلن متفاعلن

Example:



2.1.4.6 Al-Hazaj الهزج

tafa'il

مفاعيلن مفاعيلن مفاعيلن مفاعيلن

Example:

فَهُبُّوْا يَا بَنِيْ أُمِّيْ إِلَى الْعَلْيَاءِ بِالْعِلْمِ
فَهُبُّوْ يَا بَنِيْ أُمِّيْ إِلَى الْعَلْيَاءِ بِالْعِلْمِ
مَفَاعِيْلُنْ مَفَاعِيْلُنْ مَفَاعِيْلُنْ مَفَاعِيْلُنْ

2.1.4.7 Al-Rejz الرجز

tafa'il

مستفعلن مستفعلن مستفعلن مستفعلن مستفعلن مستفعل

Example:

لاخَيْرَ فِيْمَنْ كَفَّ عَنَّا شَرَّهُ إِنْ كَانَ لِايُرْجَى لِيَوْمٍ خَيْرُهُ
لاخير فِي من كَفف عَنْ نا شَررهو إِنْ كَانَ لا يرجى ليو مِنْ خيرهو
/0//0//0 /0//0//0 /0//0//0 /0//0//0 /0//0//0 /0//0//0
مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ

2.1.4.8 Al-Raml الرمل

tafa'il

فاعلاتن فاعلاتن فاعلاتن فاعلاتن فاعلاتن فاعلاتن

Example:

قَادَنِيْ طَرْفِيْ وَقَلْبِيْ لِلْهَوَى كَيْفَ مِنْ قَلْبِيْ وَمِنْ طَرْفِيْ حَذَارِيْ
قَادَنِي طَر فِي وَقَلْبِي لَلْهوى كَيف مِن قَلْ بِي ومِن طَر فِي حذَارِي
/0//0//0 /0//0//0 /0//0 /0//0//0 /0//0//0 /0//0//0
فَاعِلاتُنْ فَاعِلاتُنْ فَاعِلا فَاعِلاتُنْ فَاعِلاتُنْ فَاعِلاتُنْ

2.1.4.9 Al-Sarea السريع

tafa'il

مستفعلن مستفعلن مفعولات مستفعلن مستفعلن مفعولات

Example:

وَمَنْ دَعَا النَّاسَ إِلَى ذَمِّهِ ذَمُّوْهُ بِالْحَقِّ وَبِالْبَاطِلِ
ومَن دعَا نَ ناسَ إِلَى ذَمِّهِى ذَمُّوه بِلْ حَقِّ وبِلْ بَاطِلِ
/0//0//0 /0////0 /0//0 /0//0//0 /0////0 /0//0
مُتَفْعِلُنْ مُسْتَعِلُنْ مَفْعَلا مُسْتَفْعِلُنْ مُسْتَعِلُنْ مَفْعَلا

2.1.4.10 Al-Monsareh المنسرح

tafa'il

مستفعلن مفعولات مستفعلن مستفعلن مفعولات مستفعلن

Example:

إِنَّ ابْنَ زَيْدٍ لاَزَالَ مُسْتَعْمِلا لِلْخَيْرِ يُفْشِيْ فِيْ مِصْرِهِ الْعُرُفَا
إِنَّ بْنَ زَيْ دَنْ لاَزَالَ مُستَعمِلَن لِلْخَيْ رِيُفْ شِي فِي مِصْ رِهِ لْعُرفَا
/0//0//0 /0//0//0 /0//0 /0//0//0 /0//0//0 /0//0
مُسْتَفْعِلُنْ مَفْعُوْلاتُ مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ مَفْعُوْلاتُ مُسْتَفْعِلُنْ

2.1.4.11 Al-Khafeef الخفيف

tafa'il

فاعلاتن مستفع لن فاعلاتن فاعلاتن مستفع لن فاعلاتن

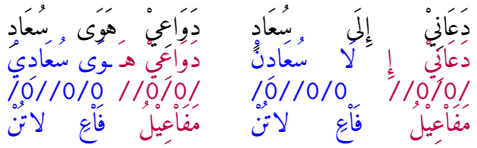
Example:

2.1.4.12 Al-Modarea المضارع

tafa'il

مفاعلين فاع لاتن مفاعلين فاع لاتن

Example:

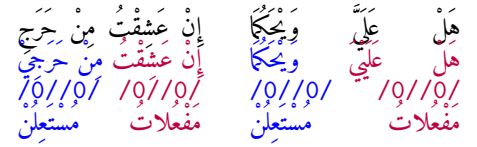


2.1.4.13 Al-Moktadeb المقتضب

tafa'il

مفعولات مستفعلن مفعولات مستفعلن

Example:



2.1.4.14 Al-Mojtaz المجتث

tafa'il

مستفع لن فاعلاتن مستفع لن فاعلاتن

Example:

2.1.4.15 Al-Motaqareb المتقارب

tafa'il

فعولن فعولن فعولن فعولن فعولن فعولن فعولن فعولن

Example:

2.1.4.16 Al-Motadarek المتدارك

tafa'il

فَاْعِلُنْ فَاْعِلُنْ فَاْعِلُنْ فَاْعِلُنْ فَاْعِلُنْ فَاْعِلُنْ فَاْعِلُنْ فَاْعِلُنْ

Example:

2.1.5 Meters Relation

Al-Arud classes have a relation between each other. The author of this relations is Al-Farahidi. He designed this relation to show the similarity and the difference between the meters. In this section, we will explain in brief this relation between them, and we will demonstrate its effect in Section 6. There are five groups of relationships between the meters. The details about how this relations arises will be omitted as it need more explanation in Arud details. But we will show brief about each one.

- **Al-Mokhtalef** المُخْتَلِف: This group also know as Al-Taweel group. As all the meters inside it is sub-child from Al-Taweel from its tafa'il. It contains three meters Al-Taweel, Al-Madeed and Al-Baseet. Table 2.5 show this group relations.
- **Al-Mo'talef** المُؤْتَلِف: This group contains two classes Al-Kamel and Al-Wafer. Table 2.5 shows this relation between them.
- **Al-Mojtaleb** المُجْتَلَب: This group contains three classes Al-Raml, Al-Rejz and Al-Hazaj. Table 2.5 shows this relation between them.
- **Al-Moshtabeh** المُشْتَبِه: This group contains six classes Al-Sarea, Al-Monsareh, Al-Khafeef, Al-Modarea, Al-Moktadeb and Al-Mojtaz. Table 2.5 shows this relation between them.
- **Al-Motafeq** المُتَّفِق: This group contains two classes Al-Motaqareb and Al-Motadarek. Table 2.5 shows this relation between them.

Group Name	Meter	Feet
Al-Mokhtalef	Al-Taweel	فَعُولُنْ مَفَاعِيلُنْ فَعُولُنْ مَفَاعِيلُنْ
	Al-Madeed	فَاْعِلاتُنْ فَاْعِلُنْ فَاْعِلاتُنْ فَاْعِلُنْ
	Al-Baseet	مُسْتَفْعِلُنْ فَاْعِلُنْ مُسْتَفْعِلُنْ فَاْعِلُنْ
Al-Mo'talef	Al-Wafer	مُفَاعَلَتُنْ مُفَاعَلَتُنْ مُفَاعَلَتُنْ
	Al-Kamel	مُتَفَاعِلُنْ مُتَفَاعِلُنْ مُتَفَاعِلُنْ
Al-Mojtaleb	Al-Raml	فَاعِلاتُنْ فَاعِلاتُنْ فَاعِلاتُنْ
	Al-Rejz	مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ
	Al-Hazaj	مَفَاعِيلُنْ مَفَاعِيلُنْ مَفَاعِيلُنْ
Al-Moshtabeh	Al-Sarea	مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ مَفْعُولاتُ
	Al-Monsareh	مُسْتَفْعِلُنْ مَفْعُولاتُ مُسْتَفْعِلُنْ
	Al-Khafeef	فَاعِلاتُنْ مُسْتَفْعِلُنْ فَاعِلاتُنْ
	Al-Modarea	مَفَاعِيلُنْ فَاعِلاتُنْ مَفَاعِيلُنْ
	Al-Moktadeb	مَفْعُولاتُ مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ
	Al-Mojtaz	مُسْتَفْعِ لُنْ فَاعِلاتُنْ فَاعِلاتُنْ
Al-Motafeq	Al-Motaqareb	فَعُولُنْ فَعُولُنْ فَعُولُنْ فَعُولُنْ
	Al-Motadarek	فَاعِلُنْ فَاعِلُنْ فَاعِلُنْ فَاعِلُنْ

Table 2.5: Meters Group.

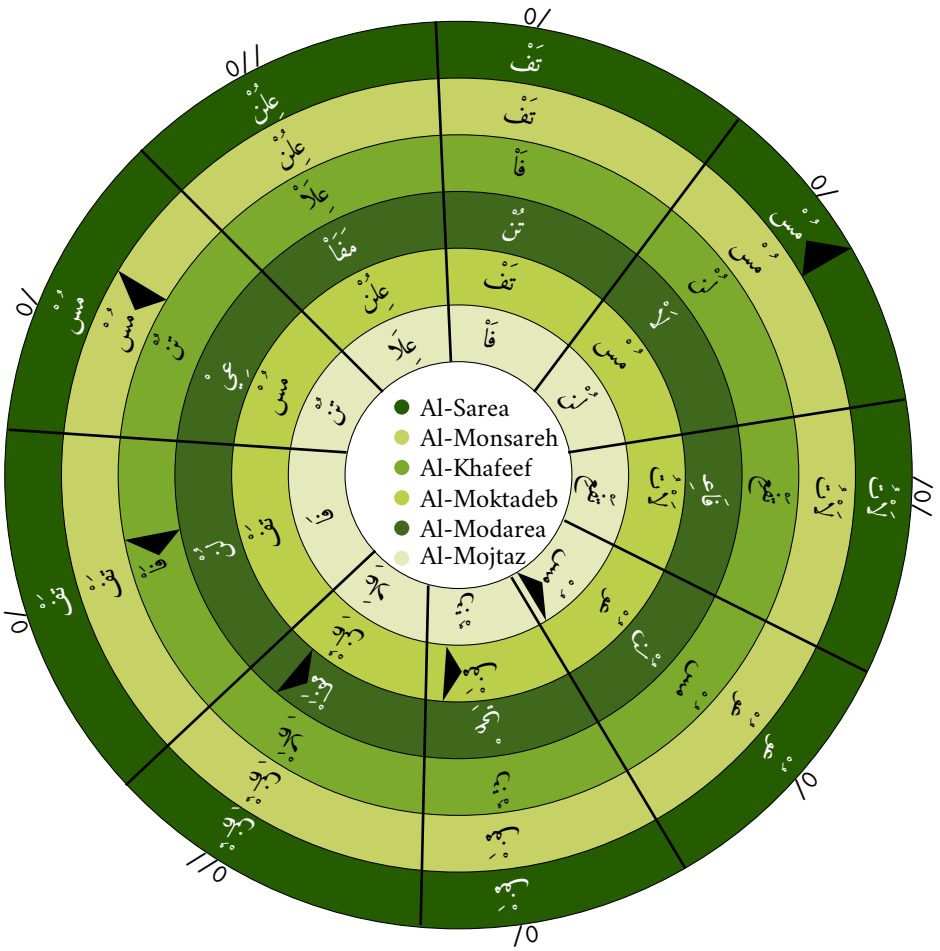
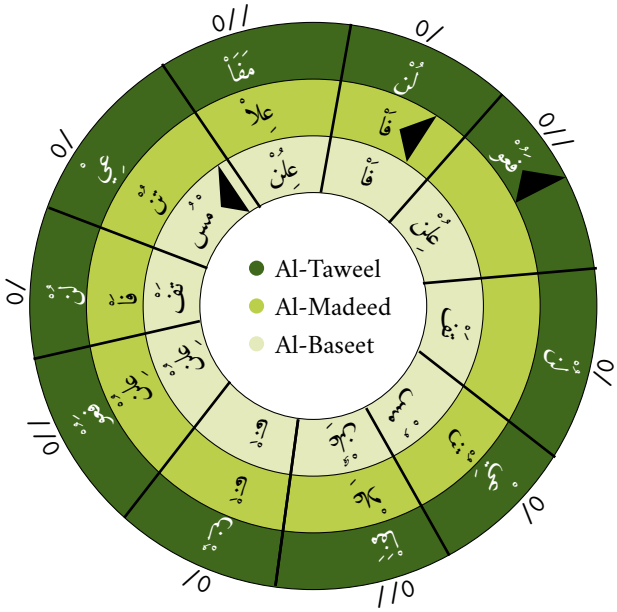
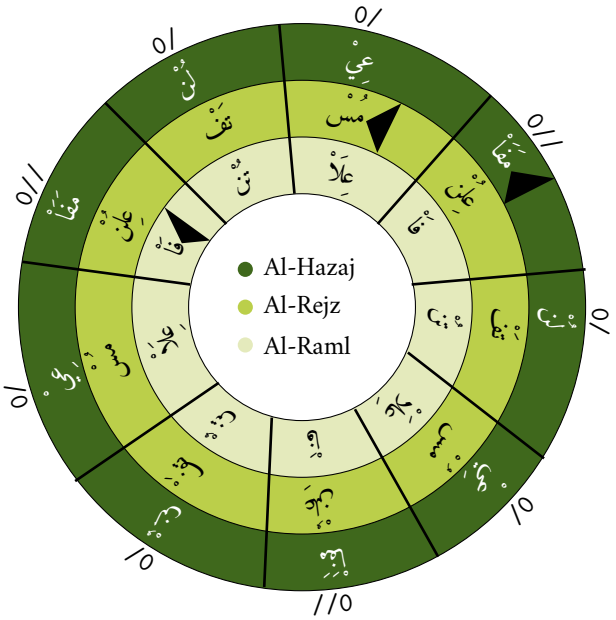


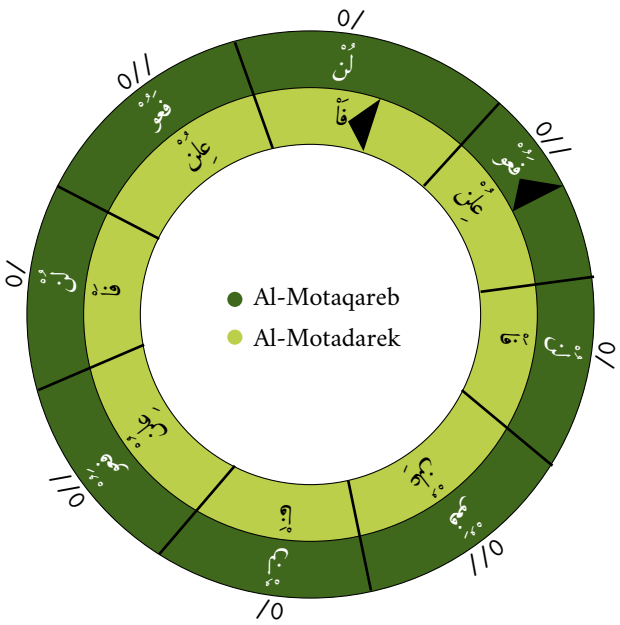
Figure 2.1: Al-Moshtabeh Meter Group.



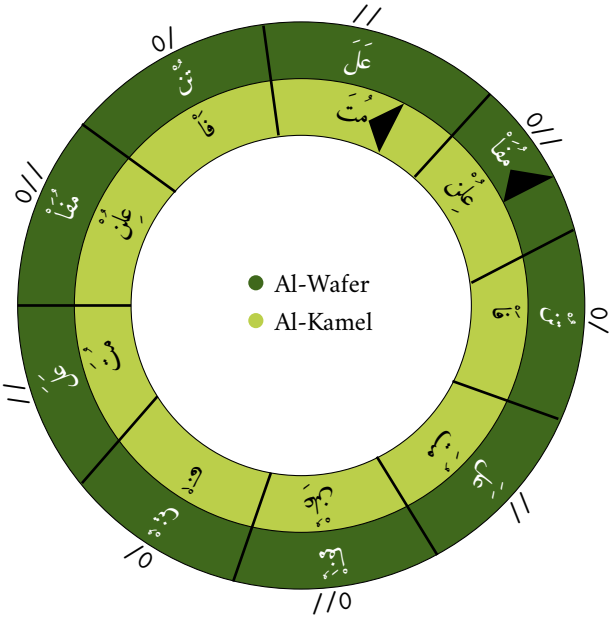
(a) Al-Mokhtalef Meters Group.



(b) Al-Mojtaleb Meters Group.



(c) Al-Motafeq Meters Group.



(d) Al-Mo'talef Meters Group.

2.2 Deep Learning Background

What is Deep Learning? *Deep Learning is a new approach to Machine Learning research which focuses on learning and understanding from the data without the need for a human operator to formally specify all the knowledge that the computer needs. This method is built using a hierarchy of concepts which enables the computer to learn complex concepts by building them layer by layer from simpler ones. A graph which shows how this concept is built will permit us to figure out a very deep graph with many layers; for this reason, we call this approach to AI deep learning [10].*

There were many early trials to utilize the AI in real life problems. For example, IBM's Deep Blue chess-playing system which defeated world champion Garry Kasprov in 1997 (Hsu, 2002).

Another approach to AI used hard-code knowledge about the world in informal language. A computer can understand statements from the formal language automatically using logical inference rules. This is known as the knowledge base approach to artificial intelligence rules. None of these projects has achieved significant success. For Example, Cyc has tried to gather a comprehensive ontology and knowledge base about the basic concepts of how the world works (Lenat and Guha, 1989).Cyc is an inference engine and a database of statements in a language called Cycl. A staff of human supervisors enters these statements. People struggle to devise formal rules with sufficient complexity to describe the world accurately [10].

The difficulty faced in the previous system is because the hard-coded knowledge has shown up the AI need to acquire its knowledge from the data itself. This capability is known as machine learning. This approach has introduced some algorithms which solve and tackle the problems from which we can, for example, check if the email is spam or not. Also, it is used for other problems, such as price predictions for housing. An example of these algorithms is *Naive Bayes* and *Logistic regression*.

This simple machine learning approach is working in the data but not with its original format; it requires some different representation to be input to the model. This different representation is named feature engineering. In the of Feature Engineering deciding whether email is spam or not, it can be word frequency, char frequency, class attributes, capital letters frequency, or some other data processing such as removing stop words from the input lemmatization. So, all the previous features provided by a human expert (who knows the problem in detail and analyzes which of its features affect the data) are then added as a feature to the input model.

However, for many tasks, it is difficult to identify the features which should be extracted. For example, we need to detect cars in photographs. We know every car has wheels. So, to detect cars, we can check if there is a wheel to be a feature for car detection. However, to detect or to describe wheels in terms of pixel values is a difficult task. The image may not be clear or may be complicated by shadows, the sun glaring off the metal parts of the wheel, the blurring in images may sometimes make it unclear, and so on [10]. One solution to this problem is to use machine learning itself to discover not only the output of the model but also the features which are the input for the model. This approach is known as representation learning. Learned representation can achieve better results than hard-designed representation. This approach also allows AI systems to rapidly adapt to new tasks or be automatically identified from any new data. Representation learning can automatically discover many features quickly or can take more time in case of complex tasks, but will at least provide an excellent set of features which can be adapted for any complex problem without the need for manual features. In this research, we used the AI to identify the features for our model, enabling this model to achieve breakthrough results, compared with the old method of manual feature machine learning.

Deep Learning is trying to solve many problems especially problems related to text and NLP by introducing representation. This learning builds complex representations in terms of another simpler layer of representations.

One of the examples of the text problems in DL is classifying text based on character-level. Figure 2.2 shows how deep learning can work to represent a sequence of a sentence by combining simpler representation, e.g. understand the character-sequence pattern which led to understanding complex representations.

The benefit of allowing the computer to understand the data and building the representation is the ability to build and understand very complex representations and also to utilize and combine features from simpler to deep representations in many ways, such as recurrence or sequences.

Deep Learning is currently involved in many problems, including text generation. Many research works use DL to generate text after learning the patterns from the text for example, training the model on Shakespeare's poems, after which the model attempts to generate a similar text. We can also use DL to classify the text of similar ideas to our research work. The main idea is how to use these techniques to get the best results without hand-crafted feature engineering.

Modern deep learning provides a compelling framework for learning data problems. This model becomes more complex by the adding more layers and more units within a layer. The Deep Learning model works perfectly on the big dataset which allows the model to learn the data features in a good way.

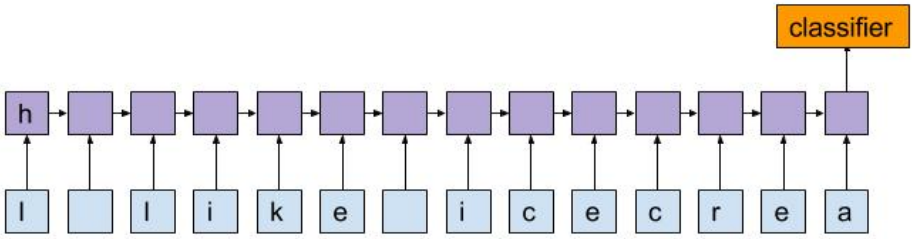


Figure 2.2: Illustrations on how Deep Learning can work based on char level figure presented [1].

In the remaining parts in this section we start introducing the main concepts and components used in deep learning, and the basic units of Recurrent Neural networks and LSTM.

2.2.1 Logistic Regression

Logistic Regression is a machine learning algorithm which we can assume has the basic idea behind the deep learning (explained later). Logistic Regression is also one of the most used machine learning techniques for binary classification.

A simple example of logistic regression would be an algorithm for fraud detection. It takes some raw data input and detects if it is a fraud case or not. Assume fraud case is one and non-fraud case is zero. David Cox developed logistic regression in 1958 [11]. The "logistic" name came from its core logistic function, also named *Sigmoid function* function in Equation (2.1). The Logistic function is shaped as an S-shape.

One of these function features can take any input real number and convert it into a value between 1 and 0.

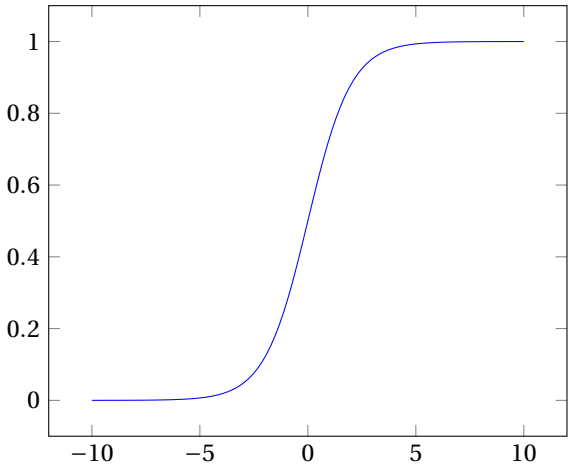


Figure 2.3: Logistic Regression Function (S-Shape)

Example: given x, we want to get the predictions of $\hat{y}$ which is the estimate of y when $\hat{y}$ is presented in Equation (2.2). So, to calculate the output function for Logistic Regression using Equation (2.3). Note: if we remove the Sigmoid function σ from the equation it becomes the Linear Regression model and $\hat{y}$ can be greater than 1 or negative. Figure 2.3 shows the Sigmoid function output.

$$x = \frac{1}{1-e^{-x}} \quad \text{where} \quad x \in \mathbb{R}^{n_x} \tag{2.1}$$

$$\hat{y} = P(y=1|x) \quad \text{where} \quad 0 \leq \hat{y} \leq 1 \tag{2.2}$$

$$\hat{y} = \sigma(w^t x + b) \quad \text{where:} \quad \sigma(z) = \frac{1}{1-e^{-z}}, w \in \mathbb{R}^{n_x}, b \in \mathbb{R} \tag{2.3}$$

2.2.1.1 Loss Error Function

Loss Error Function is the function which describes how well our algorithm can understand $\widehat{y}$ y b when the true label is y. It also can be defined as the difference between the true value of y and the estimated value of $\widehat{y}$. [5]. Equation (2.4) describe the loss function for Logistic Regression. There are other functions which can represent the loss functions but we take the example below. As we know y is the label which should be 1 or 0, this function makes sense to describe the loss function is outlined below:

- in case (y = 1) Equation (2.4) we need $\widehat{y}$ to be as big as possible to be equal or near *y true*, which is 1. So, $-(\log\widehat{y})$ will provide the value. Note, as explained before, Sigmoid function can't be greater than 1 or less than 0.
- in case (y = 0) Equation (2.4) we need $\widehat{y}$ to be as small as possible to be equal or near *y true*, which is 0. So, $-\log(1-\widehat{y})$ will provide the value.

$$\ell(y,\widehat{y}) = -(y\log\widehat{y} + (1-y)\log(1-\widehat{y})) = \begin{cases} -(1\times\log\widehat{y} + (1-1)\log(1-\widehat{y})) = -(\log\widehat{y}) & \text{if } y=1 \\ -(0\times\log\widehat{y} + (1-0)\log(1-\widehat{y})) = -\log(1-\widehat{y}) & \text{if } y=0 \end{cases} \tag{2.4}$$

2.2.1.2 Cost Function

To predict y from $\widehat{y}$, we learn from the input parameters in this case it will be ***(w,b)*** from Equation (2.3) as ***(w,b)*** are the parameters which define the relation between input dataset X and the output Y. So, Cost Function will measure how well you are doing in an entire training set and the ability to understand the relation between X,Y.

Cost function J in Equation (2.5) is the average of loss function applied to every training example which equals the sum of the loss for each training example divided by the total number of training examples

$$\begin{aligned} J(w,b) &= \frac{\sum_{i=1}^{m}\ell(y^i,\widehat{y^i})}{m} \quad \text{where m is the total number of training example} \\ &= \frac{-\sum_{i=1}^{m}[(y^i\log\widehat{y^i} + (1-y^i)\log(1-\widehat{y^i}))]}{m} \end{aligned} \tag{2.5}$$

2.2.1.3 Convex Function vs Non-Convex Function

In this subsection we give an overview of the convex and non-convex functions and their relation with deep learning. We do not explain the proofs or write them. We explain in general the definition and its related features to our topic.

Convex Function : In mathematics, a real-valued function defined on an n-dimensional interval is called *convex* if the line segment between any two points on the graph of the function lies above or on the graph, in a Euclidean space (or more generally a vector space) of at least two dimensions [12]. More generally, a function $f(x)$ Figure 2.4(a) is *convex* on an interval $[a,b]$ if for any two points x_1 and x_2 in $[a,b]$ and any λ where $0<\lambda<1$ [13],

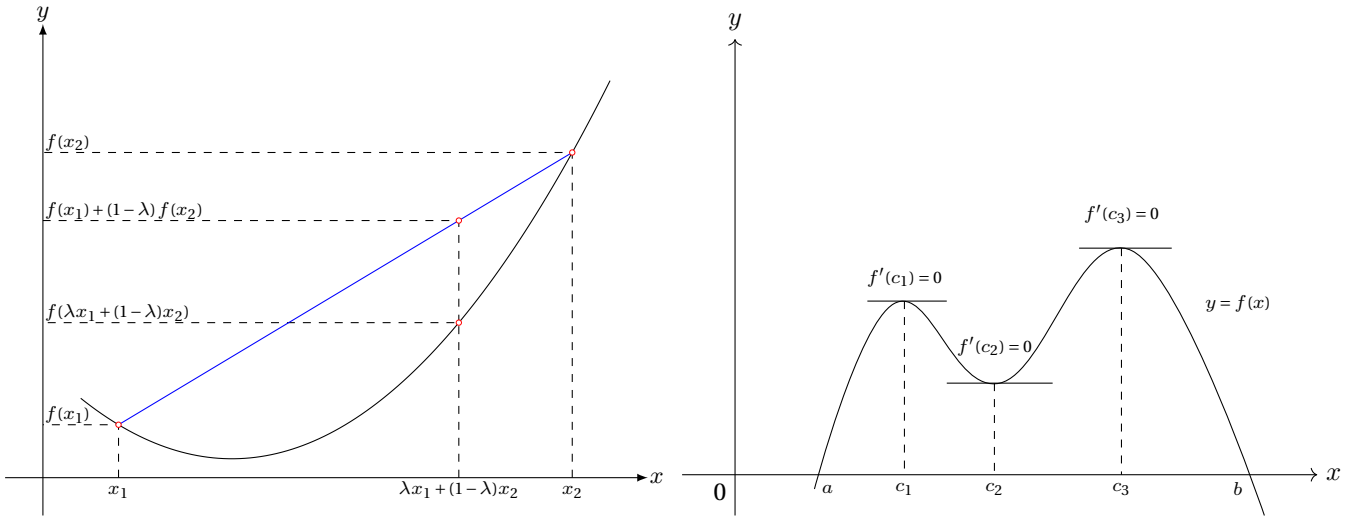
$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2) \tag{2.6}$$

For a twice differentiable function of a single variable, if the second derivative is always greater than or equal to zero for its entire domain then the function is *convex*. Well-known examples of convex functions include the quadratic function X^2 Figure 2.6(a). So, If $f(x)$ has a second derivative in $[a,b]$, then a necessary and sufficient condition for it to be *convex* on that interval is that the second derivative $f''(x) \geq 0$ for all x in $[a,b]$ [14].

If the inequality above is strict for all x_1 and x_2, then $f(x)$ is called **strictly convex**. *Convex function* on an open set has no more than one minimum. So, in strictly convex, the local minimum = global minimum. This feature is very important for any optimization problem. As we can see, most of deep learning problems are related to how to optimize the function to find the minimum point in this function. It is an easy problem to face a convex function but in the real world most cases are non-convex functions.

Non-Convex Function : In mathematics, a Non-Convex (also named concave) function is the negative of a *convex function*. A function $f(x)$ is said to be concave on an interval $[a,b]$ if, for any points x_1 and x_2 in $[a,b]$, the function $-f(x)$ is convex on

[5] *Parts of this subsections are explained into Andrew NG Coursera courses in deep learning and written using our understanding of this topic but the equations and the idea is taken from the course*



(a) Convex Function Example

(b) Concave Function Example

Figure 2.4: Convex and Concave functions examples.

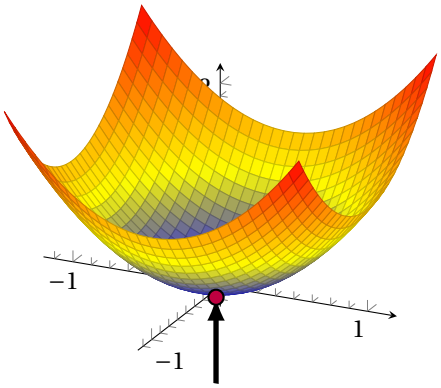


Figure 2.5: Gradient Descent

that interval.

$$f(\lambda x_1 + (1-\lambda)x_2) \geq \lambda f(x_1) + (1-\lambda) f(x_2) \tag{2.7}$$

The function is also called strictly concave if,

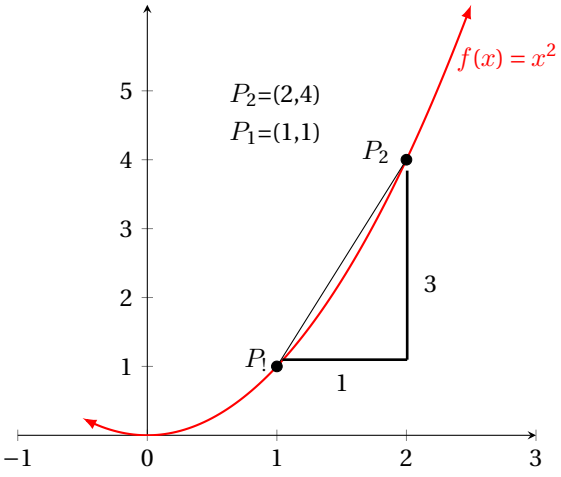
$$f(\lambda x_1 + (1-\lambda)x_2) > \lambda f(x_1) + (1-\lambda) f(x_2) \tag{2.8}$$

If f is twice-differentiable, then f is concave if and only if f'' is non-positive (or, informally, if the "acceleration" is non-positive). If its second derivative is negative then it is strictly concave, but the opposite is not true, as shown by $f(x) = x^4$ [15]. Also, a differentiable function f is (strictly) concave on an interval if and only if its derivative function f' is (strictly) monotonically decreasing on that interval; that is, a concave function has a non-increasing (decreasing) slope. The problem for non-convex optimization is to find the local minimum. Sometimes many local minimums can be found and it may be hard to optimize this function Figure 2.4(b).

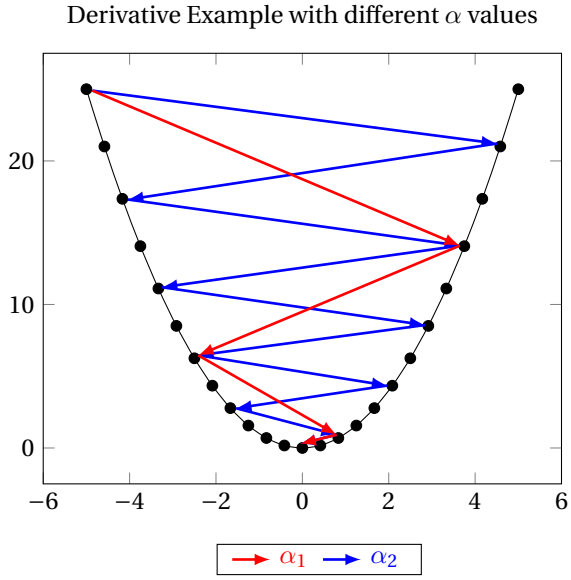
2.2.1.4 Gradient Descent

As we explained previously, we need to find the relation between X,Y from the input parameters ***(w,b)*** which will make the cost function in Equation (2.5) to the minimum. In other words we need to find the best value of ***J(w,b)*** which will represent the relation and reduce the error between y and $\widehat{y}$ So, we need to minimize ***J(w,b)***.To illustrate the relation between ***J(w,b)*** we assume for simplicity the relation is a function of one variable ***J(w)***. As shown in Figure 2.5 we have a curve which represents the function ***J(w)*** we need to find the minimum point in this curve which is the local minimum (red point in the previous figure) assuming it is a ***convex function***. we use Equation (2.9) to find the local minimum.

To explain how this equation works, let's take a simple function $f(x) = x^2$, then select a random point P_1 from Figure 2.6(a), then



(a) Derivative Example of function $f(x) = x^2$



(b) Derivative Example of function $f(x) = x^2$ where $\alpha_1 > \alpha_2$

pick another point P_2. Let's take the derivative *(which by definition is the slope of the function at the point which also the change between these two points)* The slope of this function is the height (3) divided by the width (1); this is the tangent of $J(w) = \frac{3}{1}$ at this point. If the derivative is positive, w will be update minus the derivative multiplied by learning rate alpha α as in Equation (2.9). We repeat the previous step until the value of w reaches the minimum. At this point, the derivative is negative, so w will start to increase again; at this step the algorithm will stop. Also, we can demonstrate the effect of different α values and their impact on the function; we can show this effect in Figure 2.6(b) but the main point is that it is not always a happy scenario. Sometimes the high $alpha$ is not a good idea; it depends on the problem and the dataset.

Now, let's generalize the above equation. Assume we have two parameters ***(w,b)*** and we need to calculate the cost function for ***J(w,b)*** we work on as two steps; first function in Equation (2.10a) wrt ***(w)***, and second function in Equation (2.10b) wrt ***(b)***

$$\begin{aligned} w &:= w - \alpha dw \quad \textit{alpha is learning rate} \\ &:= w - \alpha \frac{dJ(w)}{dw} \quad \textit{d represent the derivative wrt w} \end{aligned} \tag{2.9}$$

$$w := w - \alpha \frac{dJ(w,b)}{dw} \tag{2.10a}$$

$$b := b - \alpha \frac{dJ(w,b)}{db} \tag{2.10b}$$

2.2.1.5 Logistic Regression derivatives

As described, we need to calculate the gradient descent to get the best $\hat{y}$ which minimizes the total cost in equation (2.11). So, doing back propagation to get the value of dz, we need to calculate da in Equation (2.12), then calculate dz based on the output of da from Equation (2.13). After that, we start to take the derivative for z function parameters w_1, w_2, b. Once we have the values of dw_1, dw_2, db we can use them to calculate the estimated values of w_1, w_2, b in Equations (2.14), (2.15), (2.16).

$$\boxed{\hat{y} = \sigma(z) = a} \longrightarrow \boxed{z = w^t x + b = w_1 x_1 + w_2 + x_2 + b} \longrightarrow \boxed{\ell(a,y)} \tag{2.11}$$

$$\boxed{da = \frac{d\ell}{da} = \frac{d\ell(a,y)}{da} = -\frac{y}{a} + \frac{1-y}{1-a}} \tag{2.12}$$

$$\boxed{dz = \frac{d\ell}{dz} = \frac{d\ell(a,y)}{dz} = \frac{d\ell}{da}.\frac{da}{dz}} = \boxed{(-\frac{y}{a} + \frac{1-y}{1-a}).a(a-1)} = \boxed{a-y} \tag{2.13}$$

$$\boxed{dw_1 = \frac{\partial \ell}{dw_1} = x_1 dz} \longrightarrow \boxed{w_1 := w_1 - \alpha dw_1} \tag{2.14}$$

$$\boxed{dw_2 = \frac{\partial \ell}{dw_2} = x_2 dz} \longrightarrow \boxed{w_2 := w_2 - \alpha dw_2} \tag{2.15}$$

$$\boxed{db = \frac{\partial \ell}{db} = dz} \longrightarrow \boxed{b := b - \alpha db} \tag{2.16}$$

2.2.1.6 Implementing Logistic Regression on m example

To implement a simple 1 iteration, in the example below the sample code simulates the program structure. First, assume $J = 0, dw_1 = 0, dw_2 = 0, db = 0$. Then calculate the feedforward step. Then backpropagation calculate. Finally, update the parameters. We can transfer the above equation into the below python sample code.

```
import numpy as np
J = 0, dw_1 = 0, dw_2 = 0,db = 0, alpha = .02
# FEED FORWARD PROPAGATION
A = 1 / (1 + np.exp(-(np.dot(w.T,X) + b))#  Z = np.dot(w.T,X) + b
cost = (- 1 / m) * np.sum(Y * np.log(A) + (1 - Y) * (np.log(1 - A)))
# BACKWARD PROPAGATION (TO FIND GRADIENT)
dw = (1 / m) * np.dot(X, (A - Y).T) #  dz = A - Y
db = (1 / m) * np.sum(A - Y)
# UPDATE THE PARAMETERS
w = w - alpha * dw
b = b - alpha * db

```

2.2.2 The Neuron

Most computer research is trying to simulate the human brain as it is the most advanced creation. If we are trying to check how the model understands the new information, we can give a baby two bananas then ask him about them. The baby can remember it with all it new shapes. Similarly, if you inform any human about some information and trying to get a new inference, it will automatically detect this information. So, the new research tries to simulate the human brain model into an Artificial Intelligence model to achieve this performance. In this subsection, we try to give an overview of the relation between the new research era and the human brain.

The neuron is the foundation unit of the brain. The size of the brain is about the size of a grain of rice. The brain contains over 10000 neurons with average 6000 connections with other neurons [16]. These massive networks allow our brain to build its knowledge about the world around us. The neuron works by receiving the information from other neurons and processes it uniquely then passes the output to other neurons; this process is shown in Figure 2.6.

How do we learn a new concept? *The neuron receives its input from dendrites. The incoming neuron connection is dynamically strengthened or weakened based on how often it is used, and the strength of each connection determines the contribution of the input to the neuron's output. Based on the connection strength, it has weight then the input is summed in the cell body. This sum is transformed into a new signal which is propagated along the cell's axon and sent to other neurons [2].*

The above biological model can be translated into an Artificial Neural Network as described in figure 2.7. We have an input $x_1, x_2, x_3, ..., x_n$; every input has its own strength (weight) $w_1, w_2, w_3, ..., w_n$. We sum the multiplication of X and W to get the logit of the neuron, $z = \sum_{i=0}^{n} x_i w_i$. The logit is passed through a function f to produce the output $y = f(z)$. The output will be the input to other neurons. Note: in many cases, the logit can also include a bias constant. So, in this case the function will be

$$y = f(\sum_{i=0}^{n} x_i w_i + b)$$

.

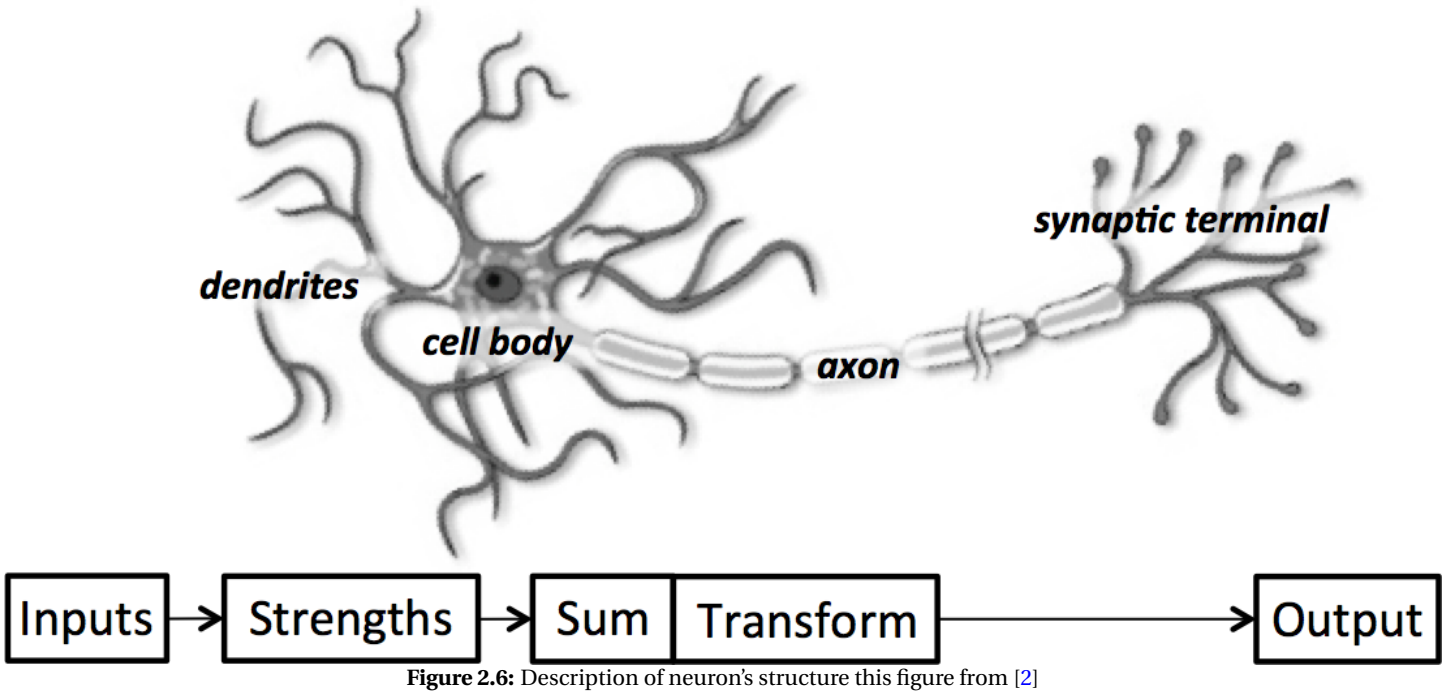


Figure 2.6: Description of neuron's structure this figure from [2]

2.2.3 The Neural Network Representation

As explained previously, we have been trying to simulate the human brain model into our research work in Deep Neural Network. So, we have multi-layers to allow the model to get in-depth knowledge and more computational performance to simulate the human brain.

Now, we represent the functions per layer as per the equations below where lt refers to layer number, i refers to the node number in the layer (2.17)

$$\boxed{z^l = W^l x + b^l} \longrightarrow \boxed{a_i^l = \sigma(z^l)} \longrightarrow \boxed{\ell(a^l, y)} \tag{2.17}$$

What is the Neural Networks component? Figure 2.7 represents an Example of Neural Networks representations. It consists of

- **Input Layer:** Input layer is the input data raw for the network, denoted as a^0.
- **Hidden Layers:** The layers between the input layers and the output layer can be any number of layers.These have a set of weighted input and produce an output through an activation function. Every layer in the hidden layer transmits the output to the other hidden layer as an input feature.
- **Output Layer:** This is one output layer with the final results from the hidden layers.

2.2.4 Neural Network Computation

In this subsection, we show as example on how we can compute the Neural Networks for every layer. In figure 2.7 we have an example of Input layer of N input, one hidden layer, and one output layer. We continue explain on this example in Equation (2.18).

$$Z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, a_1^{[1]} = \sigma(Z_1^{[1]}) \tag{2.18a}$$
$$Z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, a_2^{[1]} = \sigma(Z_2^{[1]}) \tag{2.18b}$$
$$Z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, a_3^{[1]} = \sigma(Z_3^{[1]}) \tag{2.18c}$$
$$Z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, a_4^{[1]} = \sigma(Z_4^{[1]}) \tag{2.18d}$$

If we need to compute the above equations, it is simply be represented as vectorized. The matrix below shows how we can

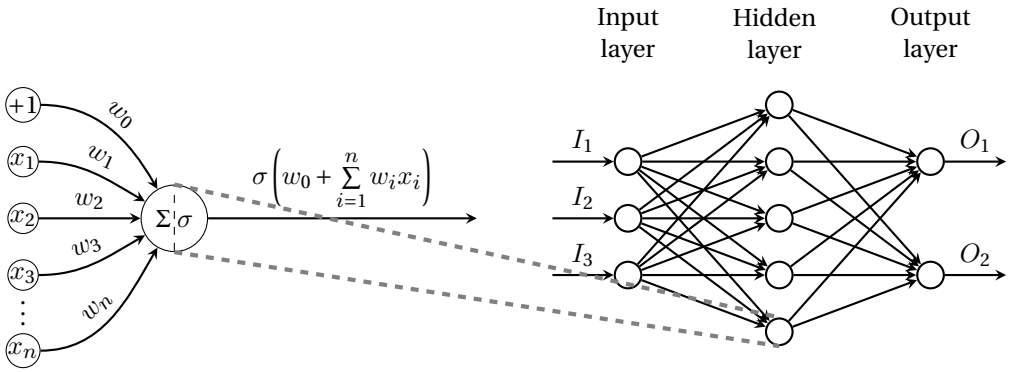


Figure 2.7: Neural Networks Example Input layer with N input samples, One hidden Layer, and an Output Layer

implement it.

$$z^{[1]} = \begin{bmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_3^{[1]} \end{bmatrix}$$

2.2.4.1 Linear Neurons and Their Limitations

We explained the equations for the feedforward Neural Network. We have only one point we need to discuss, which is the Activation function. Let's assume we continue using linear function in Figure 2.8(c), which represents linear equation $y = wx + b$. So, if we have mutli-layer networks, for example Equation (2.19), it will end as linear function because composition of two linear function will be linear function. So, we do not compute deep computation and we get limited information from the networks. So, to be able to detect the deep information we use different functions for the hidden layers, for example: tanh Figure 2.8(b) and its Equation (2.20). Another option is Sigmoid Figure 2.8(d) and Equation (2.1). Finally, Relu Figure 2.8(a) Equation (2.21). Most binary classification problems use Sigmoid function for output layer. We can use the same functions for the output but we can also use the linear for activation function in some cases.

$$Z^{[1]} = w_1^{[1]T} x + b_1^{[1]}, a_1^{[1]} = \sigma(Z_1^{[1]}) \tag{2.19a}$$
$$Z^{[2]} = w^{[2]T} a^1 + b^{[2]} = w^{[2]T}(w^{[1]T} x + b^{[1]}) + b^{[2]} \tag{2.19b}$$
$$= (w^{[1]T} W^{[2]T}) x + (w^{[2]} b^{[1]} + b^{[2]}) \tag{2.19c}$$
$$= W' x + b' \tag{2.19d}$$

$$a = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.20}$$

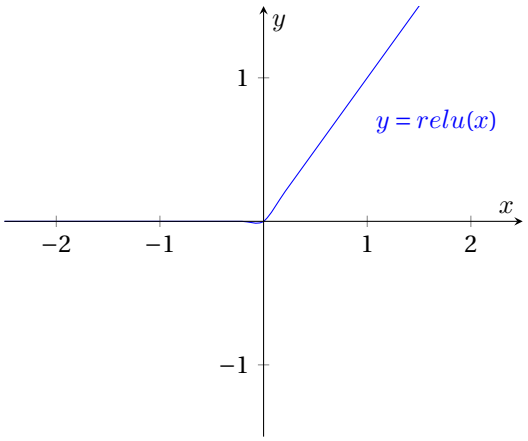
$$a = \max(0, z) \tag{2.21}$$

2.2.4.2 Softmax Output Layers

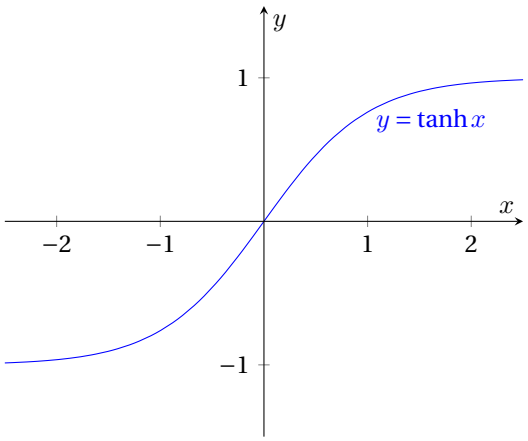
Sometimes our problem has multi-output results, not only 1 or 0. For example, we have a problem recognizing the characters from 0 to 9 in MNIST dataset, But we are unable to recognize digits with 100% confidence. So, we use the probability distribution to give us a better idea of how confident we are in our predictions. The result will be an output vector of the form of the $\sum_{i=0}^{9} P_i = 1$.

This is achieved by using a special output layer named softmax layer. This layer differs from the other as the output of a neuron in a softmax layer depends on the output of all the other neurons in its layer. This because its sum of all output equal 1. If we assume z_i be the logit of i^{th} softmax neuron, we can normalize by setting its output to represented from Equation (2.22).

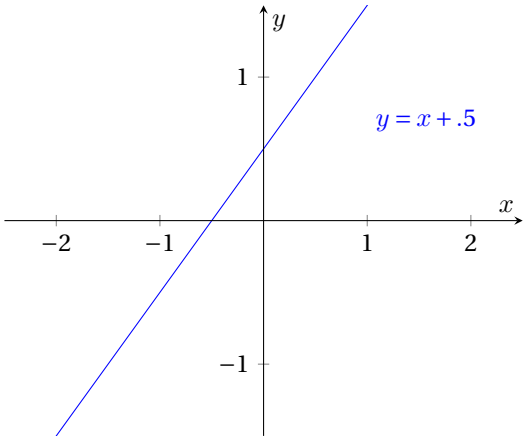
$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{2.22}$$



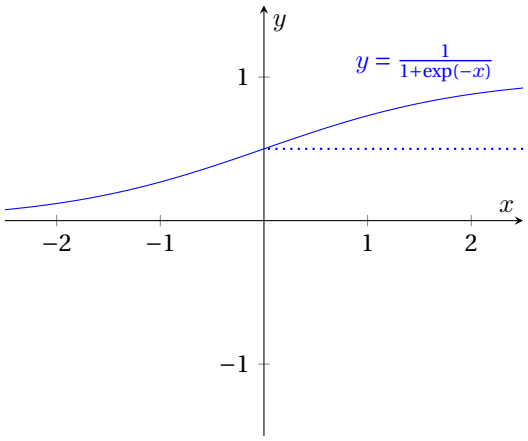
(a) RELU activation function.



(b) Tanh activation function.



(c) Linear activation function.



(d) Logistic sigmoid activation function.

Figure 2.8: Commonly used activation functions including the logistic sigmoid $\sigma(z)$, the hyperbolic tangent $\tanh(z)$, the rectified hyperbolic tangent RelU $Relu(x)$, and linear function

The strong prediction will have a value entry in the vector close to 1, while the other entries will be close to 0. The weak prediction will have multiple possible labels, having almost equal values [2].

2.2.4.3 Forward-Propagation in a Neural Networks

As an example, in Figure 2.9 we need to calculate the Forward propagation; we follow the equation (2.23). Note: we assume $X = a^{[0]}$ as initial function notation. Also, $\widehat{Y} = g(Z^{[4]} = A^{[4]}$ as the final output layer.

$$Z^{[l]} = w^{[l]}a^{l-1} + b^{[l]}, A^{[1]} = g^{l}(Z^{[l]}) \tag{2.23}$$

2.2.4.4 Back-Propagation in a Neural Networks

To compute the gradient descent in Neural Networks, we use an algorithm named *backpropagation*. The backpropagation algorithm was initially invented in the 1970s, but was relatively unknown until one of the most important papers in this field was published in 1986, describing several neural networks where backpropagation has a performance significantly better than the earlier approaches and making it possible to use neural networks to solve problems which were previously insoluble. Backpropagation is now the backbone for the learning in neural networks. We explained previously, how neural networks could learn their weights using gradient descent algorithm. In this part, we explain how to compute the gradient of the cost function.

Backpropagation not only an algorithm which gives us the expression for partial derivative of the cost function C with respect to weights w and bias b, but also suggests possibilities about the change of the cost function while changing its variables w&b and its effect on the overall network 2.9.

As explained in the Logistic Regression section (2.2.1.5), we can calculate the derivatives for logistic regression with one layer using this equations (2.11), (2.12), (2.13),

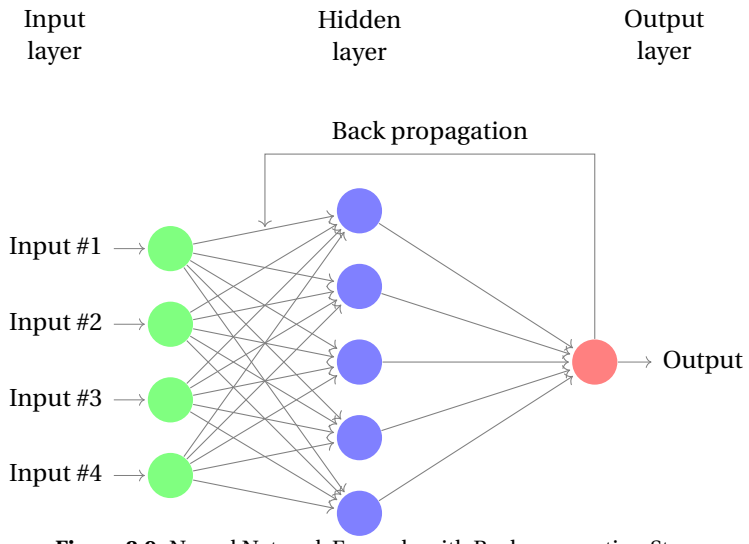


Figure 2.9: Neural Network Example with Backpropagation Step.

(2.14), (2.15), (2.16).

we generalize the derivatives equations to be for l layers from the equations below (2.24).

$$dz^{[l]} = da^{[l]} \times g^{[l]'}(z^l) \tag{2.24a}$$
$$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]} \tag{2.24b}$$
$$db^{[l]} = dz^{[l]} \tag{2.24c}$$
$$da^{[l-1]} = W^{[l]T} \cdot dz^{[l]} \tag{2.24d}$$

We can vectorize the above equation for Neural Network implementation as per the equations below (2.25).

$$dz^{[l]} = dA^{[l]} \times g^{[l]'}(z^l) \tag{2.25a}$$
$$dw^{[l]} = \frac{1}{m} dz^{[l]} \cdot A^{[l-1]T} \tag{2.25b}$$
$$db^{[l]} = \frac{1}{m} \text{np.sum}(dz^{[l]}, \text{axis=1}, \text{keepdims} = \text{true}) \tag{2.25c}$$
$$dA^{[l-1]} = W^{[l]T} \cdot dz^{[l]} \tag{2.25d}$$

If we check the input variable in the backpropagation we find it is da^l and this is the derivative of (2.4) which we can find, as explained previously from (2.12); this is the formula for the final layer in the feedforward step. If we need to calculate the vectorized version of this equation, we can use equation (2.26)

$$da = \frac{d\ell}{da} = \frac{d\ell(a,y)}{da} = (-\frac{y^{[1]}}{a^{[1]}} + \frac{1-y^{[1]}}{1-a^{[1]}} \ldots - \frac{y^{[m]}}{a^{[m]}} + \frac{1-y^{[m]}}{1-a^{[m]}}) \tag{2.26}$$

2.2.4.5 How we Initialize the Weights

As explained previously in Logistic Regression, we initialized the weights to Zero. However, in Deep Neural Networks this will not work. Note: It is acceptable to initialize the Bias to Zero but for the weights it will not work. Let's see what happens if we initialize the weights and Bias to Zero.

Assume we have two input vectors x_1, X_2, if we initialize $W^{[1]}$ to Zero from equation (2.27) and $b^{[1]}$ to Zeros. So, $a_1^{[1]} = a_2^{[1]}$ because both of the hidden units compute the same functions. Also, $W^{[2]} = [00]$ Then, when we compute the backpropagation, we find that $dz_1^{[1]} = dz_2^{[2]}$. After every iteration, we find that the two hidden units calculate the same function and we do not obtain more information from this Deep Neural Network. We need to highlight that the main idea from Neural Networks, as explained before, is that every hidden unit should work to get a new piece of information. The more hidden units, the more hidden information we

obtain, but if we initialize it to Zero, it will be the same function which is calculated, and we do not derive any new information.

$$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{2.27a}$$

$$W^{[2]} = \begin{bmatrix} 0 & 0 \end{bmatrix} \tag{2.27b}$$

$$a_1^{[1]} = a_2^{[1]} \quad dz_1^{[1]} = dz_2^{[1]} \tag{2.27c}$$

$$dw = \begin{bmatrix} u & u \\ v & v \end{bmatrix} \quad W^{[1]} = W^{[1]} - \alpha dw \tag{2.27d}$$

To initialize weights and to get the maximum value of the neural network computation, we should initialize the weight by small random numbers to avoid the big weights which will tend to cause a small slope from the Z where $Z^{[1]} = W^{[1]}X + b^{[1]}$ For example, if we use tanh we produce big tail values $a^{[1]} = g^{[1]}(Z^{[1]})$. So, big weights are more likely to produce slow learning rate.

2.2.4.6 Regularization

One of the most common problems anyone working with data science in general or deep learning faces is the overfitting problem. In statistics, overfitting is the production of an analysis that corresponds too closely or exactly to a particular set of data, and may, therefore, not fit additional data or predict future observations reliably [17]. An overfitted model is a statistical model that contains more parameters that can be justified by the data. The essence of overfitting is to have unknowingly extracted residual variation (i.e. the noise) as if that variation represented underlying model structure. As a practical example,a model performs adequately on training data but cannot predict test data. This means that the models are often free of bias in the parameter estimators, but have estimated (and actual) sampling variances that are needlessly large in the training and unable to be generalized to predict testing data. Figure 2.10 shows different ways of fitting, We need to avoid both underfitting and overfitting. We attempted to enhance the results in our training sets, where the model is more complex, so that the training error reduces but the testing error does not. While building the neural networks, the more complex models lead to more overfitting problems.We can enhance the results on

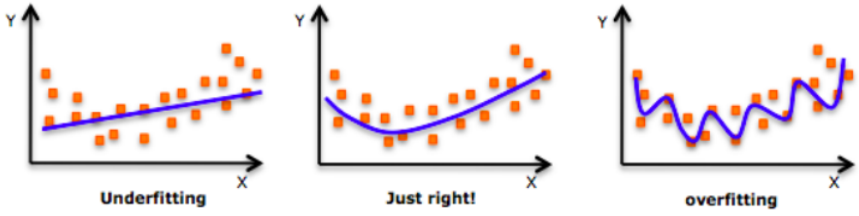


Figure 2.10: Comparison Between different fitting types

testing data by getting more sample data. Therefore, the model can learn the pattern from the data in a better way, but sometimes it is difficult to acquire more data, or it is more expensive than the enhancement required. Hence, we work to apply regularization techniques to prevent overfitting.

Regularization is a technique which applies slight changes or modifications to the algorithm to be more generalized. Sometimes we try to hide information or add noise on the training data to avoid overfitting in the training phase. This considerably improves the model performance in the testing data or unseen.

Different Regularization Techniques in Deep Learning

In this part, we explain how to apply different regularization techniques in deep learning.

- **L2 & L1 Regularization** are the most common types of regularization. They update the general cost function by adding another term (2.28) known as the regularization term [18].

$$J(w,b) = \ell(y,\hat{y}) + Regularization \tag{2.28}$$

Addition of this regularization term decreases the values of weight matrices. This happens because it assumes that a neural network with smaller weight matrices leads to simpler models. Therefore, it will also reduce overfitting to a significant extent. Assuming we reduce the value of λ to be close to zero, this will produce a simpler network and many nodes will be equal zeros. This will reduce the high variance and hence be less prone to overfitting.

1. L2 regularization is the most common type of regularization. In L2 regularization we add λ as regularization parameter. This hyperparameter value is optimized for better results (2.29). It also known as weight decay as it forces the weights to decay towards zero (but not exactly zero) [18].

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\ell(y,\widehat{y}) + \frac{\lambda}{2m} \times \|w\|^2 \tag{2.29}$$

2. In L1 Regularization, We penalize the absolute value of the weights (2.30). If we use L1 regularization W can end up being sparse which means w vector will have a lot of zeros. This can help to compress the model because the set of parameters are zero and we need less memory to store the model. Otherwise, we usually prefer L2 over it.

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\ell(y,\widehat{y}) + \frac{\lambda}{2m} \times \|w\| \tag{2.30}$$

- **Dropout** Dropout is one of the most powerful techniques to apply regularization. It produces excellent results and is consequently the most frequently used regularization technique in the field of deep learning.

  Figure 2.11 shows the idea of dropout; it randomly selects some nodes and removes them along with all of their incoming and outgoing connections. Dropout technique's randomness makes the models' results seem a more generalized model. For each node, we have a probability of some dropout percentage that this node will be removed with its connections. Therefore, we have some simpler network architecture. Also, dropout forces the algorithm not to rely on any feature. Hence, one must spread the weights and make the network try to learn a different type of features. Therefore, dropout is usually preferred when we have a large neural network structure in order to introduce more randomness.

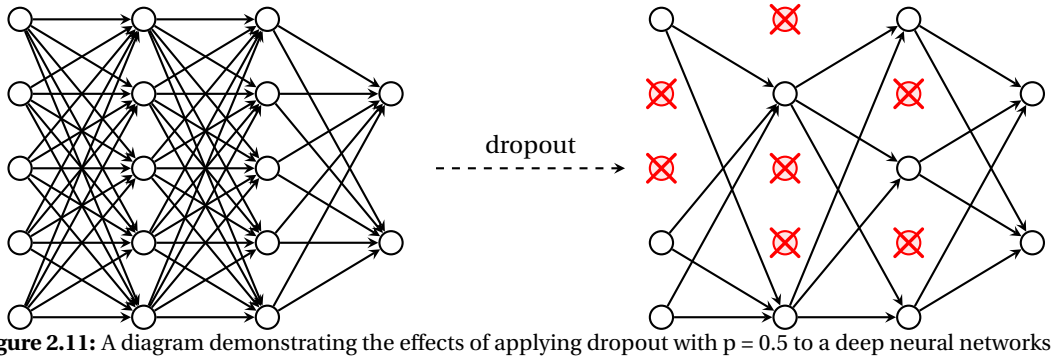


Figure 2.11: A diagram demonstrating the effects of applying dropout with p = 0.5 to a deep neural networks [3]

- **Data Augmentation** is a technique to augment your training data when there is no way to acquire more data for the more generalized model. Assuming we are working on Image classifier problem, for example, we can flip the images horizontally and also add that to the training set. Now, instead of this one example in our training set, we can add this to our training example. So by flipping the images horizontally, we could double the size of our training set. Also, we can do different techniques such as rotating the image, flipping, scaling, and shifting the images. All of these techniques can help to produce a better-generalized model.
- **Early stopping** is a type of cross-validation technique where we split some of our training data to be set as the validation set. When we show that our performance on the validation set is getting worse, we immediately stop the training on the model. This is known as early stopping.

After finalizing this part, we need to note that: In our problem based on text, we found the most effective regularization technique was *Dropout*. After we applied the dropout in our experiments most of the results increased by at least 1%, and some of the experiments increased by 3.5%.

2.2.4.7 Optimization Algorithms

As explained previously, we used Gradient Descent algorithm to find the minimum loss for our functions. However, Gradient Descent is not the only algorithm used for this purpose. In this section, we introduce another type of optimization algorithm, most commonly used for optimization problems particular to Deep Learning, named *ADAM Optimization Algorithm*.

ADAM Optimization Algorithm

ADAM stands for adaptive moment estimation. It is an adaptive learning rate optimization algorithm designed for training deep neural networks, published in 2014 at ICLR 2015 [19]. It is one of the best-designed algorithms for deep learning and proven to work well across a wide range of deep learning architectures. The Adam optimization algorithm takes Stochastic Gradient Descent with momentum (it uses a moving average of the gradient instead of gradient itself) and RMS prop (it uses the squared gradients to scale the learning rate) and puts them together. It derives its name from adaptive moment estimation, because Adam uses estimations. The first is β_1, computing the mean of the derivatives as the first moment. Second, β_2 is used to compute the exponentially weighted average of the squares and is called the second moment. We can show the algorithm pseudocode in Algorithm 1. The chosen Hyperparameter can be as follows:

1. α is the learning parameter and must be tuned based on the problem type.
2. β_1 the default choice is 0.9. This is the moving averages in momentum term for (dw,db).
3. β_2 The author of the Adam paper recommends 0.999; this computes the moving average of dw^2 and db^2 squares.
4. ϵ The author of the Adam paper recommends 10^{-8}.

Algorithm 1 ADAM Algorithm for Deep Learning Optimization.

Vdw = 0, Sdw = 0, Vdb = 0, Sdb = 0
for $t = 0$ to $num\_iterations$ **do** ▷ %: compute dw,db on mini-batches
$V_{dw} = \beta_1 V_{dw} + (1-\beta_1)dw, \quad V_{db} = \beta_1 V_{db} + (1-\beta_1)db$ ▷ %:Momentum Step
$S_{dw} = \beta_2 S_{dw} + (1-\beta_2)dw^2, \quad S_{db} = \beta_2 S_{db} + (1-\beta_2)db$ ▷ %:RMS prop Step)
$V_{dw}^{corrected} = \frac{V_{dw}}{1-\beta_1^t}, \quad V_{db}^{corrected} = \frac{V_{db}}{1-\beta_1^t}$
$S_{dw}^{corrected} = \frac{S_{dw}}{1-\beta_1^t}, \quad S_{db}^{corrected} = \frac{S_{db}}{1-\beta_1^t}$
$w := w - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}}+\epsilon}, \quad b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}}+\epsilon}$
end for

2.2.5 Recurrent Neural Networks (RNNs)

Deep Neural Networks shows its ability to solve many problems. However, in some use cases, Naive Neural Network architecture cannot work or achieve the expected results. One of the famous examples related to this issue in the NLP tasks when working on a text problem. For example, if we say Harry is the king and Elizabeth is the queen, and we need our model to understand from the sentence that, Harry is he and Elizabeth is she. Also, if this word appears again, we need the model to detect that Harry is a person.

This type of problem has a dependency on the input text and how to calculate the output prediction based on the information provided from the input.

As explained previously, most of the research in this area tries to simulate human brains, so we rarely find anyone trying to think about something start from scratch; it always starts from another related point; for example, what does a human do if he tries to connect the information to generate knowledge about something?

RNN shows its ability to work on sequence data and its related application problems such as natural language[20]. The effectiveness of RNN on language modeling is apparent. There are many problems based on this idea of dependency. For example:

- Time series anomaly detection.
- Speech recognition.
- Music composition.
- Image captioning.
- Stock market prediction.
- Translation.

So, what are the problems in the Naive Neural Network architecture?

- Input and output can be different lengths in different examples.
- The most important issue is that the Naive architecture cannot share features learned across different positions of text. In this case, we lose the learned feature; and the lake of dependency, in this case, affects the overall performance.

What is the new proposed architecture which can provide a way to share the features between the Network?

- First, assuming we have input features x_1, x_2, x_n in the old architecture, we input all these features to the Neural Network, but now we input for example x_1 and take the output activation from $a^{<1>}$ to be a feature input with x_2, then take the output activation from $a^{<2>}$ as input to x_3, similar to x_n. figure 2.12 shows an example. Hence, this new change allows us to share the learned feature between the networks input data. ,We can also think about it as multiple copies of the same network, each passing a message to a successor[4].

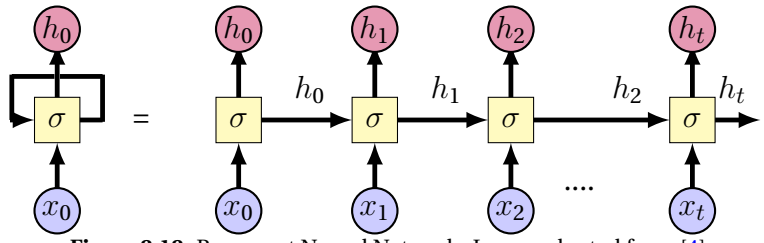


Figure 2.12: Recurrent Neural Networks Loops adapted from [4]

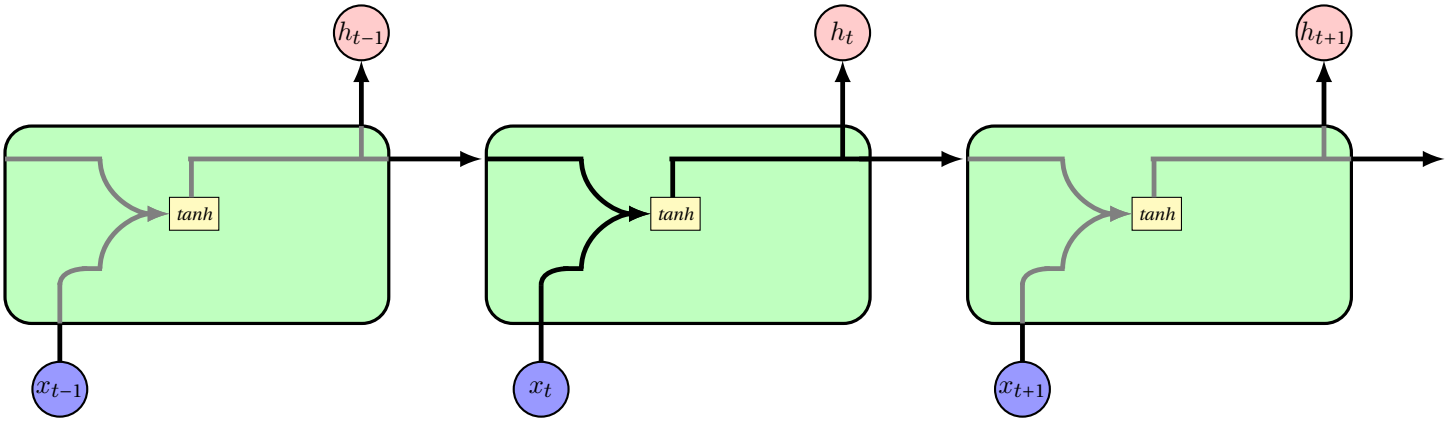


Figure 2.13: The repeating module in a standard RNN containing a single layer adapted from [4]

- Second, the feedforward computes for time t and then we calculate the loss at step t. The final loss is the sum of loss at every step; t (2.31) explains feedforward steps. The backpropagation is also calculated through time at every step.

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \tag{2.31a}$$
$$= g(W_a[a^{<t-1>}, x^{<t>}] + b_a) \tag{2.31b}$$
$$\widehat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y) \tag{2.31c}$$
$$\ell^{<t>}(\widehat{y}^{<t>}, y^{<t>}) = -(y^{<t>}\log\widehat{y}^{<t>} + (1-y^{<t>})\log(1-\widehat{y}^{<t>})) \tag{2.31d}$$
$$\ell(\widehat{y}, y) = \sim\sum_{t=1}^{T_m} \ell^{<t>}(\widehat{y}^{<t>}, y^{<t>}) \tag{2.31e}$$

2.2.5.1 Vanishing Gradient with RNNs

As explained, RNN works on sequential data, and the idea is to predict new output not only based on the input data vector but also, other input vectors. Due to the recurrent structure in RNNs, it tends to suffer from long-term dependency. To simplify this point, consider the following sentence:
Waleed Yousef, who is Associate Professor at Helwan University and teaches Data Science courses and its dependencies, ***got*** *a Ph.D. in Computer Engineering from GWU in 2006.*

In the previous example, predicting the word got depends on the sentence's tense whether it present or past to be consistent is correct. It also shows how some problems need the long-term dependencies handling.[Bengio et al.,1994] [21] showed that Basic RNNs have a problem in long-term dependency. Another problem which may happen in basic Neural Networks is gradient exploding. One of the side-effects of gradient exploding is an exponentially large gradient which causes our parameters to be so large.

Hence, the Neural Networks parameters have a severe problem. Another serious problem with Basic Neural Networks is overfitting problems [Zaremba et al., 2014] [22].

To solve this learning problem [Hochreiter and Schmidhuber, 1997] introduced Long Short-Term Memory which helps to reduce the dependency problem using memory cells and forget gates.

2.2.6 Long Short Term Memory networks (LSTMs)

Long Short Term Memory networks – "LSTMs" – are a special type of RNN, capable of learning long-term dependencies. To solve the vanishing gradient problem for long-term dependencies, [Hochreiter and Schmidhuber, 1997] [23] suggested new cell architecture for RNN by adding Long Short Term Memory which significantly reduced the long-term dependency problem using memory cells and forget gates.

LSTMs are designed to help solving the long-term dependency problem and to hold information in memory for long periods of time. They use the RNNs' sequential model but add gating mechanism structures to every cell.

Both Basic RNNs and LSTM have the form of a chain of repeating modules of neural network. The main difference is the structure of the Networks.

Basic RNNs have very simple structure for every layer with simple output function 2.13. However, LSTMs have four interacting layers Figure 2.14.

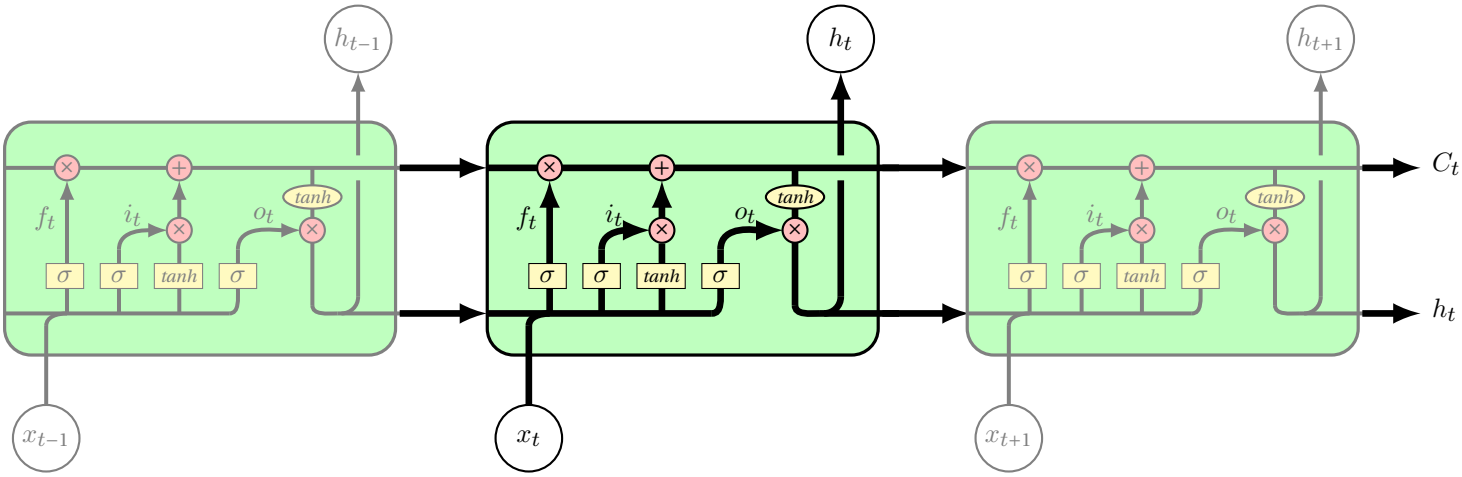


Figure 2.14: The repeating module in an LSTM containing four interacting layers adapted from [4]

2.2.6.1 LSTM Gate Mechanism

The main component of LSTM is the cell state; this allows the information to pass through it unchanged. In figure 2.15(b) the top line shows the information flow through the cell. The LSTM cell can add or remove information to the cell state using the Gating mechanism.

Gates's idea is a methodology to manage how and which information passes or not. It controls information flow through the cell. It has three of these gates. They consist of a sigmoid neural network layer 2.15(a) and a pointwise multiplication operation 2.15(b).

Sigmoid function output values range between zero and one. If the value is one, everything should pass, while if the value is zero nothing passes. Hence, the value output from the sigmoid function refers to the amount of each component should be passed.

2.2.6.2 How LSTM Works

We have explained LSTM has three gates:

- **Forget Gate Layer** a Sigmoid layer Figure 2.15(c) decides which information will be allowed to pass and which will not. It looks at h_{t-1} and x_t, and calculates the output from Sigmoid function between zero and one. As explained, if one *everything should pass*, while if zero *nothing passes*. The value zero or one depends on the value of the cell state; if it includes a gender

type and we need to predict the pronouns so, it will pass. Otherwise, it will ignore this state (2.32).

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_t) \tag{2.32}$$

- **Input Gate Layer** a combination between sigmoid layer which works to decide which values should be updated, and tanh layer creates a new vector of the new information $\tilde{C}_t$ which should be stored for the next state Figure 2.15(d). The previous combination controls the update state as shown in Equation (2.33). This layer is used when we have new input information. For example, if we have a new subject named Elizabeth we need to store it for the next input. The next step is the pointwise multiplication and addition operations.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2.33a}$$
$$\widetilde{C_t} = \tanh(W_C[h_{t-1}, x_t] + b_C) \tag{2.33b}$$

- **Multiplication and Addition operations** This step is to apply the actions recommended by the previous gates as shown in Equation (2.34). This step is the actions applying the forgetting of the old information and addition of the new information, as decided in the previous steps. Regarding the upper line in Figure 2.15(e), there are two operations:

  1. Multiplication Operation: This operation applies the forget gate step by multiplying the old state $\tilde{C}_{t-1}$ by the f_t.
  2. Addition Operation: This operation will add the output from the previous multiplication with the new input information scaled by how much we need to update each state value $i_t \times \tilde{C}_t$

$$C_t = f_t \circ c_{t-1} + i_t \circ \widetilde{C_t} \tag{2.34}$$

- **Output Gate** This gate is a combination of sigmoid layer and tanh layer. Sigmoid layer decides the information which should be output. Then the output of the sigmoid function will be multiplied by the output of the tanh layer of the cell state. This tanh will make the values between -1 and 1. The output of the multiplication of sigmoid and tanh will be the final output, as shown in Equation (2.35). In practice, this gate responsible for deciding which information should be the output. For example, if it saw a subject such as Elizabeth, it might want to output a verb to be relevant to her as a singular.

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \tag{2.35a}$$
$$h_t = o_t \circ \tanh(C_t) \tag{2.35b}$$

We have explained the normal LSTM. We also need to mention that much research proposes different modifications of the normal LSTM type. We do not explain all the types, but we give a small overview of one of these modifications named Gated Recurrent Unit (GRU) in the next part.

2.2.6.3 Gated Recurrent Units (GRUs)

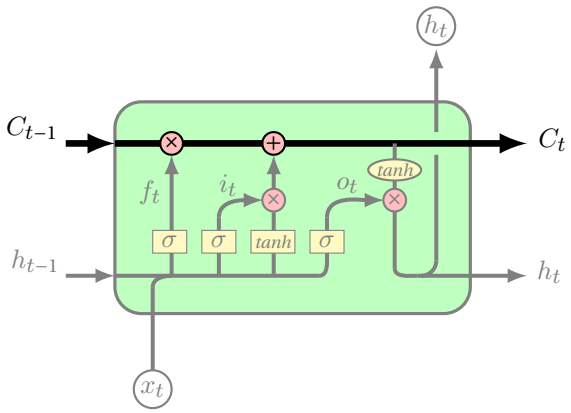
In RNN, Gated recurrent units (GRUs) are a gating mechanism, introduced in 2014 by Kyunghyun Cho et al. [24]. They work to overcome the problem for long-term dependencies. It also aims to solve the vanishing gradient problem from Basic RNNs.It proposed a new architecture Figure 2.15(g) similar to the LSTM but with some major variants as below:

- It combines the forget gate and input gates into a single gate named “update gate and reset gate.”
- The GRU unit controls the flow of information without having to use a memory unit. It simply exposes the full hidden content without any control.
- It also merges the cell state and hidden state.

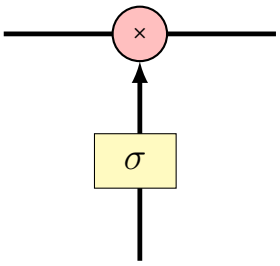
The result of this modifications is that GRUs are simpler and easier for modifications in design. GRUs trains faster and in some cases performs better than LSTMs on less training data, mainly in language modelling. However, LSTMs have some benefits over GRUs, with longer sequences in tasks requiring modelling long-distance relations.

2.2.6.4 BI-LSTM

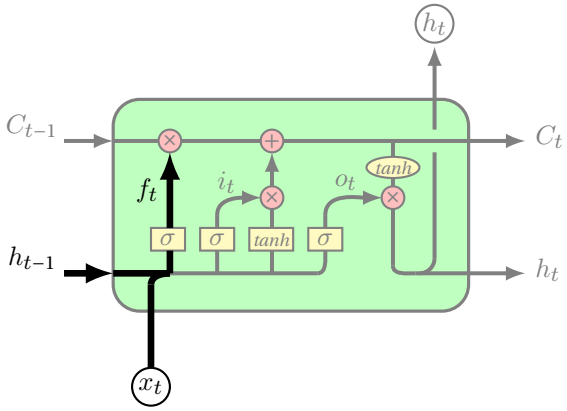
BI-LSTM refers to two LSTMs stacked on top of each other used to solve some problem where the information needs to be considered in both directions for LSTM. As the normal LSTM is working from left to right, the BI-LSTM adds the other directions into the learning information. Consider a motivation example regarding why we need BI-LSTM:



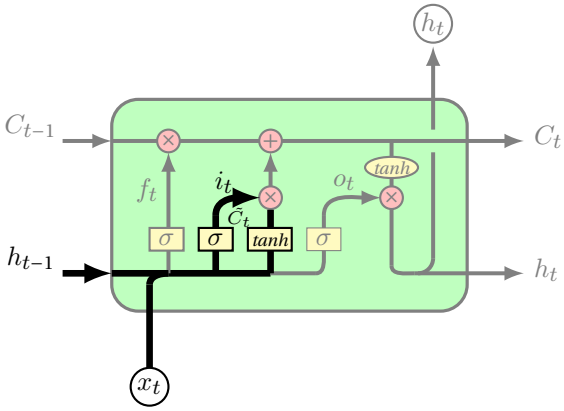
(a) Top line is the medium for information flow



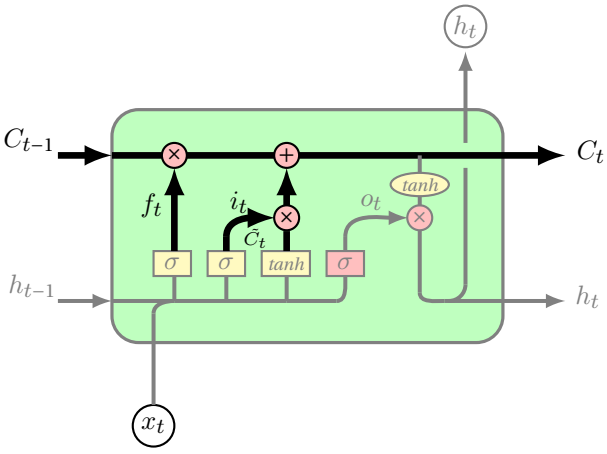
(b) Pointwise Multiplication Operation



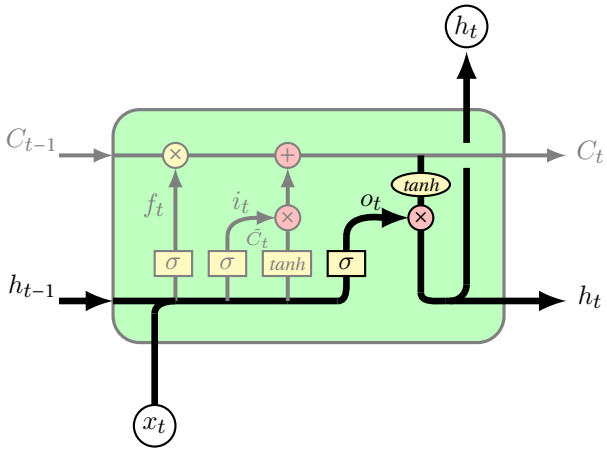
(c) LSTM sigmoid forget gate



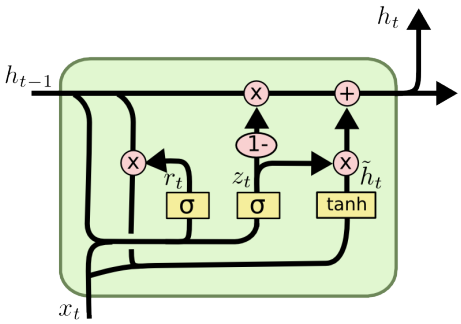
(d) LSTM Input gate



(e) Multiplication and Addition Operation in LSTM.



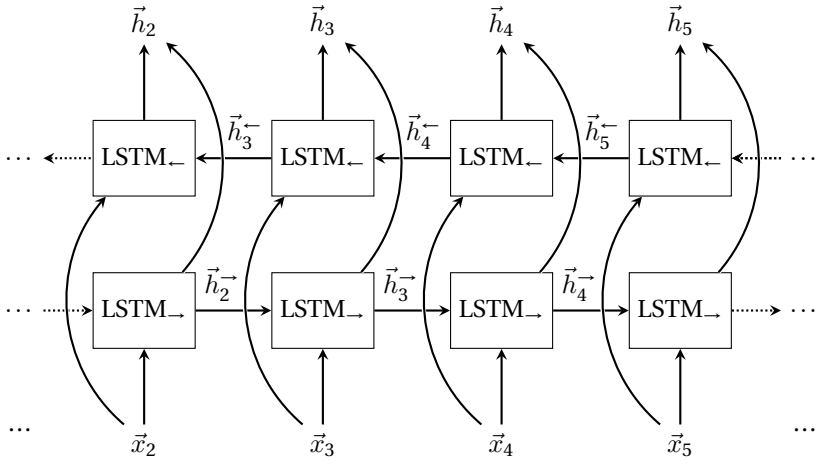
(f) LSTM output gate.



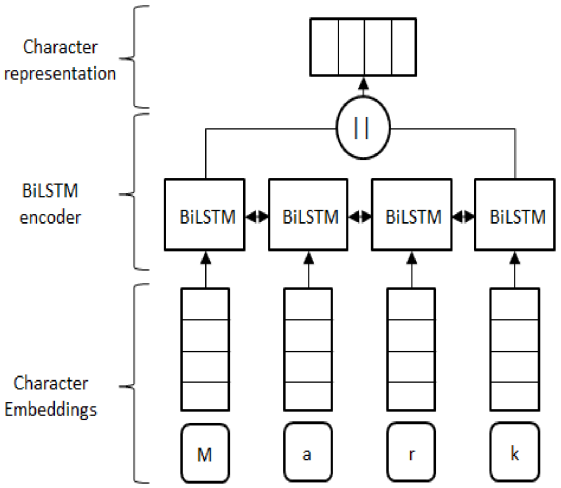
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

(g) GRU cell architecture.

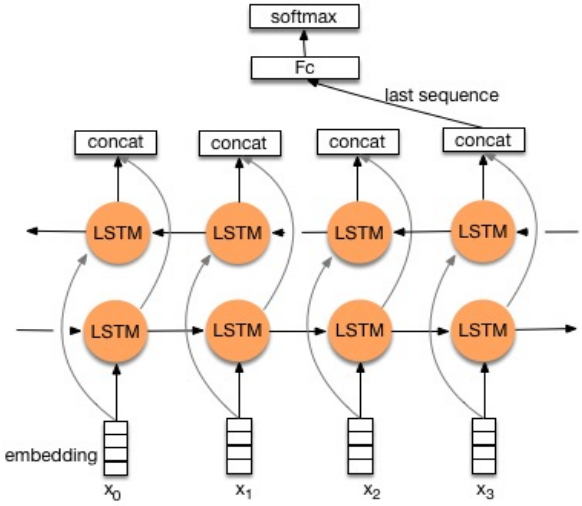
Figure 2.15: LSTM Gates and Configurations adapted from [4]



(a) bidirectional long short-term memory [3]



(b) character-based representation using BiLSTM [25]



(c) An example of the architecture of LSTM [26]

- *Harry is the king, and he will travel next week.*
- *The new book which makes the big sale is named Harry Potter..*

Harry in the first example refers to a person, but in the second example refers to a book. So, if we are working left to right, we do not get the type of word Harry represents in the second example.

The architecture in BI-LSTM is similar to what we discussed previously regarding Uni-LSTM. We can mention here that BI-LSTM is very slow compared to LSTM, and needs much time in the training phase but, as we see later in our research, it is impressive regarding the results and the effect on the language problems.

Figure 2.16(a) represents a recurrent neural network consisting of several LSTM blocks, processing the input sequence simultaneously forwards and backwards (to exploit both directions of temporal dependence). In Figure 2.16(b) we can observe a similar example to our encoding data, which feeds the networks using a sequence of char, then passes to the BI-LSTM cells, then produces the character representations to feed the next layer of the network. Figure 2.16(c) shows the simple example of the full network from char-level embedding (encoding) till the last sequence which outputs to softmax layer.

2.2.7 Machine Learning Model Assessment

As explained previously, machine learning cycle starts by Data preparation, Feature extraction, Model training and Model assessment 1.1. In this section we explain the meaning by model assessment. we also discuss different techniques for model assessment.

In Supervised machine learning problems, we have two types *Classification and Regression*. Classification of the output is discrete variables, for example, Spam vs Normal Email detection. In Regression, the output is a continuous variable: for example,

Label	Actual Number	Actual Spam (Positive)	Actual Normal (Negative)
Predicted as Spam (Positive)	200	160	40
Predicted as Normal (Negative)	300	10	290

Table 2.6: Spam vs Normal Email Classifier example

Predict Housing Price. Each type of problem has its methods or performance evaluation matrix. In this section, we focus on Classification problem, as our problem is a meter classifier application.

How can we measure the model performance? It is good when the difference between the predicted value and the actual is small and not overfitting on the development dataset.

There are many ways to measure classification performance. Before we explain every method, we give a simple example to allow us to understand the output of every method. Let us assume we have a binary classifier which detects Spam vs Normal Emails. we explain by example the definition for every type base on the example data in Table 2.6

2.2.7.1 Accuracy

Accuracy measures the corrected prediction over the dataset. It is calculated using the ratio between the corrected predicted sample from the test data over the total test data sample. Ex: in Table 2.6 *(the total positive predicted as positive + the total negative predicted as negative)/total number of test data* (2.36) which means 90%. An issue in Accuracy as a measurement is that it doesn't give us any sense of the results with respect to the actual performance of the positive, calculated as positive and vice-versa. One measurement which gives us some sense is the Precision and Recall, which we discuss it in the next section.

$$\frac{\text{the total positive predicted as positive + the total negative predicted as negative}}{\text{total number of test data}} = \frac{160+290}{500} = 0.9 \tag{2.36}$$

2.2.7.2 Precision and Recall

Precision and Recall are two measurements which answer questions related to our model performance. Some applications consider one of them and others require both or a combination. Before explaining it, we should prepare a data table named Confusion Matrix similar to the example in 2.6.

- Precision (also called positive predictive value) is used to answer the question, **Based on the test data how many items predicted correctly from the test data sample**. It can be calculated from Equation (2.37).
- Recall (also known as sensitivity) answers another question: **Based on the test data, how many of the truly predicted items are truly predicted**. We can calculate this using Equation (2.38).

$$precision = \frac{\text{true positives}}{\text{true positives + false positives}} = \frac{160}{160+40} = 0.8 \tag{2.37}$$

$$recall = \frac{\text{true positives}}{\text{true positives + false negatives}} = \frac{160}{160+10} = 0.941 \tag{2.38}$$

We need to highlight that some applications could require a focus on precision more than recall and vice-versa. We therefore need to choose the right measure based on our problem.

2.2.7.3 F_1 Score

In statistical analysis of binary classification, the F1 score is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the accuracy score. The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0 [27]. We can calculate it using Equation (2.39).

$$F_1 = 2 \times \frac{precision * recall}{precision + recall} \tag{2.39}$$

2.2.7.4 Per-Class Accuracy

Per class, accuracy is one of the important accuracy measures when we have multi-class. There is a difference between each accuracy calculation regarding the dataset classes distribution. Most of the performance measurement calculate the average accuracy per-class but the difference is some of them take into consideration the size of each class and some don't take the size. So, in case we have imbalanced dataset we should use the type which considers the class size for the accuracy of calculations. In case the data is imbalanced the results could hide information about how the model is performing. For example, assuming the model working perfectly with a class which has the most amount of our data and performing badly in the other class the accuracy can give us high results. However, if we use the average per-class, it will give us a more clear vision of how the model is performing regarding the dataset size for each class. Below is common types used in most of the framework to calculate accuracy per-class. Note: The below naming is followed the same as *Sklearn* Library.

- Weighted: Calculates the F1 score for each class independently but when it adds them together uses a weight that depends on the number of true labels of each class (2.40).

$$F_{1class1} \times W_1 + F_{2class2} \times W_2 + F_{3class3} \times W_3 \tag{2.40}$$

- Micro: Uses the global number of TP, FN, FP and calculates the F1 directly (2.41). It does not take into consideration the size for each class.

$$F_{1class1+class2+class3} \tag{2.41}$$

- Macro: Calculates the F1 separated by class but not using weights for the aggregation (2.42) which results in a bigger penalization when the model does not perform well with the minority classes.

$$F_{1class1} + F_{2class2} + F_{3class3} \tag{2.42}$$

Chapter 3

Literature Review

Poetry meter classification and detection have not addressed as a learning problem or similar our way for solving this problem. In this literature, we can see that they treat the problem mainly as deterministic. They are restricted by some static conditions and unable to deploy machine learning approach to address the problem. This approach can work but with other problems which have static rule-based approach. However, this problem requires learning approach; hence, the rule-based model is not the best option in our case.

3.1 Deterministic (Algorithmic) Approach

[28] present the work most related to our topic, classifying Arabic poetry according to its meters. However, they have not addressed it as a *learning problem*; they have designed a *deterministic five-step algorithm* for analyzing and detecting meters. The first and most important step is to have the input text carrying full diacritics; this means that every single letter must carry a diacritic, explicitly. The next step is converting input text into *Arud writing* using *if-else* like rules. *Arud writing* is a pronounced version of writing; where only pronounced sounds written. Then metrical scansion rules are applied to the *Arud writing*, which leaves the input text as a sequence of zeros and ones. After that, each group of zeros and ones is defined as a *tafa'il*, giving a sequence of *tafa'il*. Finally, the input text classifies the closest meter to the *tafa'il* sequence. 82.2% is the classification accuracy on a relatively small sample, of 417 verses.

[29] has taken a similar approach to the previous work, but it replaced the *if-else* by *regular expressions* templates. This approach formalized the scansions, Arud based on lingual rules relating to pronounced and silent rules, directly related to *harakat* as context-free grammar. Only 75% of128 verses were correctly classified.

[30] have taken a similar approach but worked on detecting and analyzing the arud meters in Ottoman Language. They convert the text into a lingual form in which the meters appear. Their first step is converting Ottoman text transliterated to Latin Transcription alphabet (LTA). Subsequently, they feed the text to the algorithm which uses a database containing all Ottoman meters to compare the detected meter extracted from LTA to the closest meter found in the database which saved the meters.

Both [28] and [29] have common problems:

1. **The size of the test data** which cannot measure the accuracy for any algorithms they have constructed because it is a very small dataset. In addition, a 75% total accuracy of 128 verses is even worse.
2. **The step converting verses into ones and zeros pattern** is probabilistic; it also depends on the meaning, which is a source of randomness. Treating such a problem as a deterministic problem will not satisfy the case study. It results in many limitations, including obligating verses to have full diacritics on every single letter before conducting the classification. This is also the case with [30] work: for their algorithm to work, the text must be transliterated into LTA.

We can summarize the results difference between our work and the previous work in Table 3.1

Ref.	Accuracy	Test Size
[29]	75%	128
[28]	82.2%	417
This article	96.38%	150,000

Table 3.1: Overall accuracy of this article compared to literature.

3.2 English Literature

[31] has used several machine learning models (Naive Bayes, SVM, averaged perceptron, CRF, HMM) to automatically perform metrical scansion on English poetry. The models are trained on handcrafted-feature like number of syllables and others which can be easily obtained by English NLP tools.

[32] This work is the ML handcrafted-features version of ours, they classify poems by meter, they experimented different classifier but the J48 design tree provided the most accurate results. Thanks to a program called Scandroid and pronunciation dictionaries, they were able to extract the features that effect the poetic pattern, features like the stresses position.

At this point, we have to notice that English poetry the Arabic poetry are two different patterns, they deceptively share the same name which can lead to false hidden assumption like the techniques done on English poetry can work on Arabic poetry, but what is true is non of the English techniques can work on Arabic poems due to differences of the languages grammar and the pattern nature. English pattern is much easier to be formalized for computers, which is not the case with Arabic.

[33] develop a CRF model to predict the metrical values of syllables in MHG epic verse. HandCraftedFeatures, Length of syllable in characters, Syllable characters Syllable weight and length.

[34] has used generative models (LSTMs) to generate rhythmic verses, what is special about this work is that they trained the models on both phonetic character encoding. As usual in this kind of works on English poetry, they got the phonetics from a dictionary (CMU pronunciation dictionary). And found out that phonetic encoding is more informative than character encoding.

[35] has extended his earlier feature-based models to featureless neural network models. He showed that the character-based models are more informative that the earlier feature-based models. He used Averaged Perceptron, Hidden Markov Models, Conditional Random Fields (CRFs) and Bidirectional LSTMs with a CRF layer, the Bi-LSTM+CRF-model outperformed them on scansion of poetry in two languages Spain and English.

Chapter 4

DESIGN DATASET

In this chapter, we discuss the Dataset Design done in this project. The Dataset Design steps started from acquisition and encoding, including the essential pre-processing steps, and the justification for their use. Pre-processing steps are data extraction, data cleansing, data format, and data encoding techniques employed. Additionally, it contains comparisons between the three techniques used.

4.1 Dataset Design

In this section, we introduce the dataset acquisition and encoding, including the essential pre-processing steps, and the justification for their need. Pre-processing steps are data extraction, data cleansing, data format, and the data encoding techniques used. This section also contains comparisons between the three techniques used.

The collection of the dataset was one of the most laborious tasks in this project, involving the search for these criteria:

- **Datasets availability:** There are old Arabic references which contain many poems, not all of which were available in PDF or Web pages format, and could be difficult to locate.
- **The Poem with diacritics:** Some resources have Arabic Poems, but it is difficult to find versions with diacritics.
- **The amount of the dataset:** To have a successful project with good results we need a massive amount of data. From the previous work, We did not find this amount of data. The maximum number found was 1.5k. However, We were searching for around 1.5M record of classified poetry.
- **Cleansing of this data:** The amount of the datasets which could be considered was limited; alternatively data could be scrapped due to the limited APIs or already-existing datasets in this context.

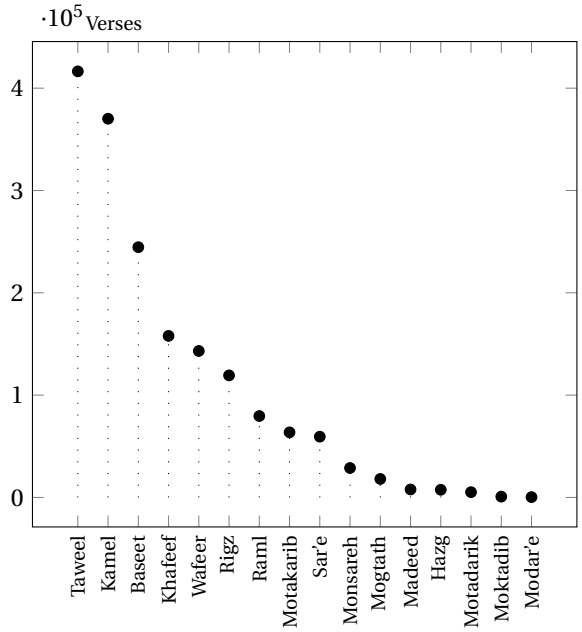
To meet the above criteria and overcome it, we applied following:

- **Datasets availability:** We have scrapped the Arabic datasets from two large poetry websites: الديوان [36], الموسوعة الشعرية [37]. Both were merged into one large dataset, which was open sourced online [38].
- **The size of the dataset:** The total number of verses is 1,862,046 poetic verses; each verse is labeled by its meter (class), the poet who wrote it, and the age which it was written. There are 22 meters, 3701 poets and 11 ages;these are Pre Islamic, Islamic, Umayyad, Mamluk, Abbasid, Ayyubid, Ottoman, Andalusian, the era between Umayyad and Abbasid, Fatimid, and modern. We are only interested in the 16 classic meters attributed to Al-Farahidi, which constitute the majority of the dataset with a total number of 1,722,321 verses. Figure 4.1 shows the distribution of the verses per meter.
- **Poems with diacritics:** We tried to get the most verses with the available diacritics, but these are inconsistent; a verse may contain full diacritics, partial diacritics or no diacritics. We analyzed the diacritics on the data and we found the following:
  - Out of 1,722,321 verses, we have 1,371,388 with diacritics.
  - Less than 0.001% of our data contains full diacritics.
  - The number of verses which contain 30% of their letters with diacritics is 302,515.
  - The number of verses which contain 40% of their letters with diacritics is 12,993.
  - The average percentage of the diacritics for each verse is 16.37
- **Cleansing of this data:** Dataset was not sufficiently cleansed for usage in this research, but we have applied cleansing rules explained in detail in the Data Preparation and Cleansing section Data Preparation and Cleansing. We also open sourced all the code scripts used in our online repository [39].

4.1.1 Data Scraping

To scrape the data from the website: الديوان [36],is to minimize the problem. Specifically, it should first be ascertained if any ”keywords” are set, if used. Then the entire preamble and the complete bibliography are printed. If the same error recurs, the problem might be in the preamble. If so, the preamble is reduced, until the bibliography is printed. Parts are slowly added back to the preamble, until the error occurs again. This may reveal the cause of the warning. We used custom Python scripts for each website, to acquire the verses' details. The script created with simple usage to pass the link we need to scrape, illustrated below with examples from both websites.

1. First example; if we need to scrape a meter from الديوان the website, for example Al-Tawil `https://www.aldiwan.net/poem.html?Word=%C7%E1%D8%E6%ED%E1&Find=meaning`, we will passed this link to the script and the output file name. The script started scraping and the output was saved in a CSV format. We can obtain an output similar to that in the output in table 4.1



(a) *Arabic Dataset*

Figure 4.1: Arabic dataset Meter per class percentage ordered descendingly on x axis vs. corresponding meter name on y axis all class in the left of the red line (less than 1% assume to be trimmed in some experiments).

2. Second Example; if we need to scrape the same meter from الموسوعة الشعرية the website (for example Al-Raml `https://poetry.dctabudhabi.ae/#/diwan/poem/126971`), we passed this link to the script and the output file name. The script started scraping and the output was saved in a CSV format. We can obtain an output similar than the output in table 4.2

   We scraped all the available datasets on both websites and merged them based on the common columns. Then we started the Data preparation tasks. It should be mentioned that not all diacritics were correctly available on all the websites. Nor did we work to generate the diacritics for those datasets. We therefore depended on available data, which remained unchanged. The following sections relate to correction, preparation, and cleansing of the current datasets.

4.1.2 Data Preparation and Cleansing

Data preparation and cleansing tasks were divided into multi-stages:

- All scraped datasets were merged into one CSV file with a selection of the common columns in each file.
- The duplicates rows were removed from the files in case of any joined rows between both websites.
- The datasets on the 16 meters required were filtered, as some data belonged to other non-famous or not original meters.
- Many unnecessary or useless white spaces were removed.
- Removal of non-Arabic characters and other web symbols.
- Diacritical mistakes were rectified, such as the removal of one of two consecutive harakats.
- Removal of any *harakah* occurring after a white space.
- Shadaa 1 was factored to its original format, explained in this example 2.1 previously.
- Tanween 2 was also factored to its original format, explained in this example 2.2 previously.[1]

We need to highlight that the last two points are not a handcrafted feature. They involve a factorization of the letter to its original format. This factorization will affect the size of the data in the memory, and the letter representation in the vector. We will explain this part in detail in the next chapter on the encoding mechanism and the impact of the encoding type in the model training time and performance.

4.1.3 Data Encoding

As explained, we collected the dataset and checked the data from any quality issues. The next step was to prepare the data representation for model training. This change in the data structure is named Data Encoding.

[1] *We ignored the factorization of Alef-Mad* (آ) *in our data preparation and transformation, thus saving more memory and shortening our encoding vectors.*

الشاعر	البحر	الشطر الأيمن	الشطر الأيسر	البيت
ابن نباته المصري	الطويل	رَجا شافع نسج المودّة بيننا	ولا خيرَ في ودّ يكون بشافع	رَجا شافع نسج المودّة بيننا ولا خيرَ في ودّ يكون بشافع

Table 4.1: Aldiwan scraping output example

#	البيت	الشطر الأيمن	الشطر الأيسر	البحر	القافية	الديوان	الشاعر	العصر
1	من يرد مورد حب ظمأ بالشوق يزدد	ظمأ بالشوق يزدد	من يرد مورد حب	الرمل	د	الديوان الرئيسي	يعقوب الحاج جعفر التبريزي	الحديث

Table 4.2: Al-Mosoaa Elshearyaa scraping output example

4.1.3.1 Encoding in English

- **Word embedding Encoding in English:** The concept of data encoding was first introduced by Bengio et al. (2003) [40]. They used an embedding lookup table as a reference and mapped every word to this lookup. They used the resulting dense vectors as input for language modeling. There are many works to improve the word embedding one of them. Collobert et al. (2011) [41] proposed improvement of word embedding task and proved the versatility of word embedding in many NLP tasks. Other works proposed by Mikolov et al. (2013) [42]; and Pennington et al. (2014) [43]] show the maturity of word embedding, which is currently the most used encoding technique in the neural network based natural language processing.
- **Character Level Encoding in English** : All the previous work focused on word embedding encoding, but in thisresearch problem, we work not on word level; instead we focus on character level encoding as an input feature of the model. There is much research based on character level encoding. Kim et al. (2015) [44] used character level embedding to construct word level representations to work on vocabulary problems. Chiu and Nichols (2015) [45] also used character embeddings with a convolutional neural network for named entity recognition. Lee et al. 2017] [46] used character embeddings for personal name classification, using Recurrent Neural Networks.

4.1.3.2 Character Level Encoding in Arabic

Working on Arabic language embedding based on the character level has attracted little attention from the research community. Potdar et al. (2017) [47] have undertaken a comparative study on six encoding techniques. Of interest is the comparison of *one-hot* and *binary*. They have used Artificial Neural Network for evaluating cars, based on seven ordered qualitative features. The accuracy of the model was the same in both encoding *one-hot* and *binary*. Agirrezabal et al. (2017) [48] show that representations of data learned from character-based neural models are more informative than the ones from hand-crafted features.

This research makes a comparative study between different encoding techniques *binary* and *one-hot*. We also provide some new encoding method specific to Arabic letters, and we will see the effect of this on our problem. We will show the efficiency of every technique based on performing model training and model running time performance.

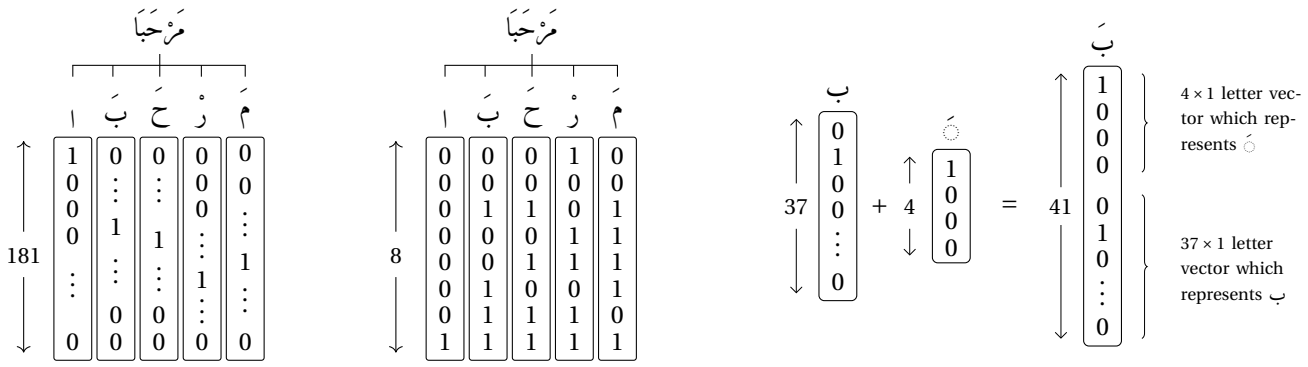
Generally, a character will be represented as an n vector. Consequently, a verse would be an $n \times p$ matrix, where n is the character representation length and p is the verse's length. Where n varies from one encoding to another, we have used *one-hot* and *binary* encoding techniques and proposed a new encoding, the ***two-hot*** encoding.

Arabic letters have a feature related to the diacritics; To explain this feature we will take an example based on *one-hot* encoding. This feature is related to how we will treat the character with a diacritic. Arabic letters are 36 + white space as a letter. So, the total is 37. Any letter represented as a vector 37×1. Let's take an example a work such as مرحبا having 5 letters encoded as a 37×5 matrix. If it came with diacritics such as مَرْحَبًا and we need to represent the letters as *one-hot* encoding we will consider every letter and diacritics as a separate letter. So, it will be 5 character and 4 diacritics. The vector shape will be 41×9.

One diacritical feature of Arabic letters can be explained by an example based on *one-hot* encoding. This feature is related to our treatment of characters with diacritics. There are 36 Arabic letters. Including the white space, the total is 37. Any letter represented as a vector 37×1. For example, a work such as مرحبا has 5 letters encoded as a 37×5 matrix. If it came with diacritics such as مَرْحَبًا and we needed to represent the letters using *one-hot* encoding, we would consider every letter and diacritic as a separate letter, thus 5 characters and 4 diacritics. The vector shape will be 41×9.

One of the main reasons for focusing on the encoding is the *RNN* training. Different numbers of time steps in *RNN* cells, and different input vector dimensions based on the input will lead to a standard architecture for the model and permit both work with and without diacritics to show the effect of the model learning on the same architecture.

To achieve the model architecture unification, we proposed three different encoding systems: *one-hot*, *binary*, and the novel encoding system developed in this project *two-hot*. All three are explained in the next three subsections.



(a) *One-Hot* (b) *Binary* (c) *Two-Hot*

Figure 4.2: Different encoding mechanisms

***One-hot* encoding** In this encoding system, we assume the letter with the diacritic as one unit. So, for example, دَ represented as letter differs than (دَ, دِ, دُ, دْ). Now every letter is represented 5 times (one without diacritic and four times with different diacritics), giving combinations 36×5 besides the white-space character. Hence, the total is 181×1. Henceforth, we have 181 Arabic alphabetic characters represented in the *one-hot* encoding, which we employ to encode verses. (Figure 4.2).

It should be noted that that *one-hot* encoding technique is one of the famous techniques in encoding problems. We will not compare encoding techniques in these sections. We will, however, discuss it in detail in model results. Moreover, the implementations of the *one-hot* is trivial. We need to focus on the size for every letter which is 181×1;meaning if we have a verse with 82 characters, it will result in a matrix 181×82. This would require very large memory.

Binary Encoding The idea is to represent a letter with an $n \times 1$ vector which contains a unique combination of ones and zeros. $n = \lceil \log_2 \rceil l$ where l is the alphabet length, and n is a sufficient number of digits to represent l unique *binary* combinations. For example a phrase like this مَرْحَبَا has 5 characters. figure 4.2 shows how it is encoded as a 8×5 matrix, which saves 22.6 times memory compared to *one-hot*, and reduces the model input dimensions significantly. On the other hand, the input letters share some features due to the *binary* representation as shown in figure 4.2.

***two-hot* encoding** This is an intermediate technique which takes the advantages of the previous two encoding techniques. In it, we encode the letter and its diacritic separately as two *one-hot* vectors, hence the letter is encoded as 37×1 *one-hot* vector and the diacritic is encoded as 4×1 *one-hot* vector, then both vectors are stacked to form one 41×1 vector (Figure 4.2).

We thus reduced the vector dimension from 181 to 41, and also minimized the number of shared features between vectors to the maximum one at each vector.

4.1.3.3 Categorical Values Encoding

Machine Learning uses the encoding techniques to transform the categorical data (class label or text) into numbers. This will make it easy for the predictive model to understand our label in an easy representation. Some encoding techniques are *label encoding, one hot encoder, and custom binary encoding*. All these encodings will convert the categorical to numerical values, but in different ways. For example, Label encoding can transform the following labels: (apple, orange, banana) to (0,1,2) as a numeric representation. As explained in the previous subsection, *one-hot* encoding could transform the labels to ([1,0,0],[0,1,0],[0,0,1]). In this research, we used the label encoding for label encoding to the meter's class, using the *Sklearn* library in Python [49]. The categorical encoding technique has no effect on the results. However, there is some difference in the execution and running time performance.

Chapter 5

Model Training

In this chapter, we discuss the training and experiment design. This phase starts by exploring the ratio of Training, Testing, and Validation. Choosing the correct percentage of training dataset compared to the Testing and Validation in Deep learning differs from normal machine learning structure, and affects the model performance. One must choose the correct Neural Network and RNN architecture.

In a normal machine learning project, the dataset would be divided into approximately 60% training, 30% testing and 10% validation. However, in The Deep Learning, the amount of data profoundly affects the model performance. Hence, the more data fed as training, the more performance results the model can achieve on test data (Assuming regularization and the required generalization were conducted in the model training phase). The main reason to change the split size is due to the size of the dataset we previously worked on; for example 1% of 1M sample is 10k, which is big enough for such experiments. However, if we work on a 10k sample, we need around 30% 3k sample to have confidence in our model. It therefore depends on the problem and size of the dataset.

The current research addresses Poem Comprehensive Dataset (PCD) [38]. We are interested in the 16 classic meters, attributed to Al-Farahidi. These meters comprise the majority of the dataset with a total number of 1,722,321 verses. Figure 4.1 shows an ordered bar chart based on the number of verses per meter. We trained all our models based on 80%, around 1,377,856 verses. Our testing data (development) is 10%, around 172,232 verses, and our Validation data around 172,232 verses.

We can show the training phase designed as a data representation configuration and an RNN configuration. The number of experiments is the cross product of both data representations and RNN configuration; for example, If we have 12 data representations and 12 RNN configurations, the total number of experiments will be 144. It should be emphasised that:

- Data representation feature is a general feature applied to the dataset, not a hand-drafted feature. More details on Parameters of Data Representation.
- Data representation feature is affected by Arabic language pronunciation, and some features provide more information than others.
- RNN configurations are the parameters related to the Network model development training, and are used after many experiments to find and tune the best configurations. This means we did more than the number of experiment written in this research but publish only the best results overall. More details are provided in sec Parameters of Network Configuration.
- The number of verses (344,464) used in testing and validation is significant, confirming that the model was tested on all types of verses and inspiring confidence in the results.

5.1 Parameters of Data Representation

Arabic language parameters have the following types: Diacritics, Trimming, and Encoding. Every type has its effect on the data and the performance on the model learning rate. Each is explained in detail in the following subsections.

5.1.1 Diacritics

The Arabic dataset contains the verse with diacritics. We can feed the network the characters with and without diacritics. With diacritics, it will be much easier for the network to learn, since they provide more information on the pronunciation. Moreover, they provide more information relating to the vowel and consonant sounds in the letters. However, as we discussed in Data Encoding Chapter Data Encoding, the inclusion or otherwise of diacritics has the same length in input vector size.

5.1.2 Trimming Small Classes

The Arabic poem dataset, as stated in Figure 4.1, is unbalanced. Hence, as part of our research, we consider the dataset representation as a Full dataset and Trimmed dataset. This allows us to study the effect of this unbalance. We not only explore the impact of the unbalanced dataset, but also apply a technique to solve this issue Working on Unbalanced data using Weighted Loss. The trimmed classes are five classes which have less than 1% of the total dataset. We present these classes as all the classes on the left side of the horizontal red line in Figure 4.1. Therefore, the classes after trimming total 11, and in the Full dataset all 16 meters are presented.

5.1.3 Encoding Techniques

As explained previously, there are three different encoding methods Data Encoding. Although all carry the same information, it was expected that every encoding has its own behaviors (as below):

- **Running Time**: It was expected that the running time would differ from one encoding type to others. This information is important if someone is conducting an experiment with limited time and needs the results as fast as possible.

#	Diacritic	Encoding Types			Trimming
1	With diacritic	*one-hot*	*two-hot*	*binary*	Full
2	Without diacritic	*one-hot*	*two-hot*	*binary*	Trimmed

Table 5.1: Data Representation Combination Matrix

- **Required Resources**: It was expected that different resources will be required from one encoding type to others. This information is important if someone has an experiment with limited resources.
- **Learning Rate**: Some encoding will learn faster than others (Note: Learning Rate not the overall performance) for example, one encoding can achieve 80% on training performance after four epoch, but another one can reach the same percentage after 20% epoch.
- **Overall Performance**: The final performance percentage which can be achieved with each encoding technique. (Note: the best learning rate may be the method which takes most time.) Therefore, the researcher who will use this encoding should decide which one will be used based on the criteria needed.
- **Overall Performance**: The final performance percentage which can be achieved with every encoding technique (Note: the best can be the worst in learning rate or the method which take much time). So, the researcher who will use this encoding should decide which one will be used based on the criteria needed.

5.1.4 Data Representation Matrix

The data representation matrix is the cross product of the Diacritics 2, Data Encoding 3, and Trimming 2: total 12 combinations table 5.1 shows this matrix combination. Example (With diacritic + *one-hot* + Full)

5.2 Parameters of Network Configuration

As explained previously, Recurrent Neural Networks (RNN) show the ability to solve language model problems in Section Recurrent Neural Networks (RNNs). We used RNNs with Long Short Term Memory (LSTM) Long Short Term Memory networks (LSTMs) as the main architecture for our experiments. We also used BI-LSTM, discussed in Section BI-LSTM, as an alternative way to test the effect of BI-Directional LSTM, to check the learning patterns with the two directions.

We also thought using BI-LSTM will support the model to learn the *tafa'il* for every class as it can be combined with the sound of music from both perspectives. We will explain our argument and the effects on the results in Section Results. In RNNs network configuration parameters, we took 100k sample of the data and applied numerous experiments and trials to find the best network configuration parameters which we show in the next part. However, due to the hardware limitations, we didn't apply much higher configuration parameters, due to the memory issue. We can mention that, for example, when we were trying to increase the unit size over 82 and the layers more than 7 we had insufficient memory exceptions and sometimes reduced for experiment stability. There are four parameters:

- **Cell Type**: We used LSTM and BI-LSTM.
- **Layers**: We tried many numbers of layers and we found the optimum number based on our problem to be 4 and 7 layers.
- **Cell Unit Size**: We also tried many numbers but the best results were achieved between 50 and 82.
- **Weighting Model**: As shown in Figure 4.1, classes are unbalanced. Hence, as an alternative to removing the small classes, we tried to keep all classes, weighting the loss function to account for the relative class size. We introduced a new weighting function, explained in the next sub-section Working on Unbalanced data using Weighted Loss to help work on all the dataset. We therefore have two combinations; one with weighting loss and one without weighting loss.

The total number of combinations is 16, which is 4 parameters: each one has 2 types. Thus, the total will be $2^4 = 16$; and hence, there are 16 different network configurations to run on each of the 12 data representations above. This results in $16 \times 12 = 192$ different experiments (or models). Hence, there are 96 different network configurations to run on each of the 2 data representations above. This results in $96 \times 2 = 192$ different experiments (or models), whose accuracy is presented on the y-axis of Figure 6.1. For all the 192 experiments, networks are trained using dropout of 0.2, batch size is of 2048, with Adam optimizer, and 10% for each validation and testing sets.

5.2.1 Working on Unbalanced data using Weighted Loss

Unbalanced dataset issue is one of the important problems we addressed in research. We strove to overcome the unbalanced dataset and to protect our model from this issue. We therefore introduced a Weighting function 5.1 where nc is the sample size of class c, c = 1,2,...C, and C is the total number of classes.

$$w_c = \left(\frac{\frac{1}{n_c}}{\sum_{c'} \frac{1}{n_{c'}}} \right) \tag{5.1}$$

The idea was to increase the loss for the small classes, given that the number of verses of small classes is small, so the output will be bigger than the loss in cases where the class has many verses. To have a clear explanation, we can show equation 5.2 [1].

$$w_c = \frac{\frac{1}{288}}{\sum \frac{1}{416428} + \frac{1}{370116} \cdots + \frac{1}{288}} \tag{5.2a}$$

$$= \frac{\frac{1}{288}}{0.00535} = 0.03 \tag{5.2b}$$

$$= \frac{\frac{1}{416428}}{0.00535} = 0.0004 \tag{5.2c}$$

[1] The equation numbers are rounded for simplicity; as an example the biggest class will have a smaller loss compared to the smallest class. We divided by constant, which is the sum of classes' density

5.3 Experiments

5.4 Hardware

We used a Dell Precision T7600 Workstation to conduct our experiments with Intel Xeon E5-2650 32x 2.8GHz CPU, 32GB RAM, 1 NVIDIA GeForce GTX 1080 ti GPU[2].

More details are available in the following website link: asus website Hard disk SSD 256; with Ubuntu OS, x86_64 Linux 16.04 LTS. We must emphasize that[3]. Beside the effect of the GPU for Deep Learning experiments, we utilized the memory and the processors to prepare the batches for the input model.

5.5 Software

During our Model development we used the following software and libraries,

- Python 3.7: *Used as main programming language.*
- Tensorflow: *Used as Deep learning backend framework*
- Keras: *Used as High level framework on top of the backend*
- Pyarabic: *Used in data pre-processing and cleansing.*
- pandas: *Used in data pre-processing and splitting.*
- sklearn: *Used to encode the classes using Label-Encoder and for model assessment phase.*
- pickle: *Used to save the encoder and the model as serialized pickle object.*
- h5py: *Used to save the encoded dataset matrix in h5 format.*

5.6 Implementation Outline

1. Data Reading and Cleansing:
    - We set the random seed and *Numpy* seed in the code to be reproducible.
    - We used PyArabic [50] to trim and strip some dummy letters [4].
    - We cleaned the data from any dummy char using Pandas and Numby libraries.
    - We obtained the maximum Bayt length for padding [5].
    - We removed the dummy letters or wrong diacritics letters.
    - We factorize *Shadaa and Tanween* to their original letters as explained in Chapter BACKGROUND.
    - We divided the data into Full (i.e. Including all the Meters) and Eliminated (i.e. Removing the meters which have less than 1% of the total data).
2. Data Encoding:
    - We encoded the Meters label using the label encoder in *Sklearn* and generated Full and Eliminated to output label encoder.
    - We save the encoder serialized in the desk for later usage.
    - We used *Pickle* Python library to save the serialized object with *H5 format* in *h5py* library.

[2] *GPU ASUS ROG STRIX GeForce GTX 1080 Ti Assassin's Creed Origins Edition AC-ORIGINS-ROG-STRIX-GTX1080TI, Memory Type: GDDR5X, Connectors: DisplayPort Output, DVI Output: HDMI Standard Output, Chipset/GPU Manufacturer: NVIDIA, Brand: ASUS NVIDIA GeForce GTX 1080 Ti, Compatible Port/Slot: PCI Express 3.0, Memory Size: 11GB 352-Bit GDDR5X, Core Clock 1594 MHz (OC Mode), 1569 MHz (Gaming Mode (Default)), Boost Clock 1708 MHz (OC Mode), 1683 MHz (Gaming Mode (Default)), 1 x DL-DVI-D 2 x HDMI 2.0 2 x DisplayPort 1.4, 3584 CUDA Cores.*

[3] we found a major impact using SSD hard disk when data reading

[4] The data has some dummy letters named Tatweel for example شعـــر is similar to شعر so we removed this letter as it has no meaning.

[5] Padding used zeros, which are not selected Arabic Char representations

- We used three methods of encoding *one-hot*, *binary* and *two-hot*, all of them developed using custom Python functions.
- We saved the data for each encoding with two combinations: *Full/Eliminated and With/Without tashkeel. tashkeel.*
- We saved these as *h5* format in the desk.

3. Model Training:

   - We used *Keras with Tensorflow* backend as Deep Learning Framework.
   - We created a custom function to handle the Data combinations *Full/Eliminated and With/Without tashkeel* with the Network combinations *Layers, Units and cell type (Bi-LSTM or LSTM).*
   - We faced a performance issue while training LSTM for Recurrent Neural Networks as we have a huge amount of data on the current known behaviour of RNN, as it is recurrent. We therefore took steps to reduce our RNN training time by following the next steps:
     - Saving the data cleaned and encoded in the hard disk, and reading it directly encoded (as explained in the previous steps). The data was read in batches and the hardware utilized to train the dataset in parallel with these.
     - We used Nvidia Cuda optimized LSTM cell, which significantly reduced our training time by up to 6x speedup.
     - We used a parallel GPU in our experiments to test the effect, by utilizing a Keras utils multi_gpu_model option.
     - We saved all the model output for later validations.
   - We saved all the model output for later validations.

4. Results and Validations

   - We calculated the confusion matrix on the testing data to calculate the results and carry out the validations.
   - We calculated the accuracy from the confusion matrix which was similar to the accuracy generated from *Tensorflow* on the testing data.
   - We used weighted accuracy to check the accuracy for Per-Class.

,

Chapter 6

Results

In this chapter, we discuss the results and discussion phase. It explains the results of all the 192 experiments on our dataset. Also, How we assess our model design. We discuss some aspect related to our results.

In this section we explain the results of all the 192 experiments in our dataset. We measure the results using the overall accuracy, then measure the performance accuracy of the model per class (meter). We start by presenting the results for every combination, and then discuss our findings relating to the topic.

As explained in Section Model Training we have a set of combinations requiring exploration. Most of our results therefore involve a combination. These are explored below:

1. Three data representations, *binary*, *one-hot*, and *two-hot*, are represented as ***BinE, OneE, TwoE*** respectively.
2. Two types of model loss functions, ***Weighting loss and no Weighting loss***, are represented as ***(1 and 0)*** respectively.
3. The Number of layers is represented as ***nL***; for example, 7 layers are 7L.
4. The Number of cell units is represented as ***nU***; for example, 82 units is 82U.

Therefore, if we need to explain a set of combinations, we write (4L, 82U, 0), which means 4 layers, 82 units, and no weighted loss function. Many figures are provided, all of which explain specific perspectives on the results .

6.1 Overall Accuracy

We present the overall accuracy score of the 16 neural networks configurations and at each of the 12 data representations (y- and x-axis respectively) in Figure 6.1. The x-axis is divided into 4 strips corresponding to the 4 combinations of trimming and diacritical parameters. Each strip includes the 3 different encoding values. Each point on the figure represents the overall accuracy score of one of the 192 experiments (some values are too small, and hence omitted from the figure). To explain the figure, we take as an example the most-left vertical list of points that represent the 16 experiments of the full (no trimming), diacritics, and *binary* encoding dataset representation. For each rug plot, the highest (Bi)LSTM accuracies are labeled differently, as circle and square respectively; and the network configuration of both is listed at the top of the rug plot.

To explain the figure, we take as an example the most-left vertical rug plot, which corresponds to (0T, 1D, BinE) data representation. The accuracies of the best (Bi)LSTM are 0.9025 and 0.7978 respectively. The configuration of the former is (7L, 82U, 0W). Among all the 192 experiments, the highest accuracy is 0.9638 and is possessed by (4L, 82U, 0W) network configuration on (1T, 0D, BinE) data representation.

6.2 Data Representation Effects

In this section we will explain the effect of the 12 data representation techniques explained previously.

1. **Trimming Effect:** The effect of trimming (removing the small classes from the training cycle) can be observed if we fix the other two parameters, diacritic and encoding. The score with trimming is consistently higher than that with no trimming, e.g. by looking at the two unshaded strips, the score at (1T, 0D, TwoE) is 0.9629, while that at (0T, 0D, TwoE) is 0.9411. The only

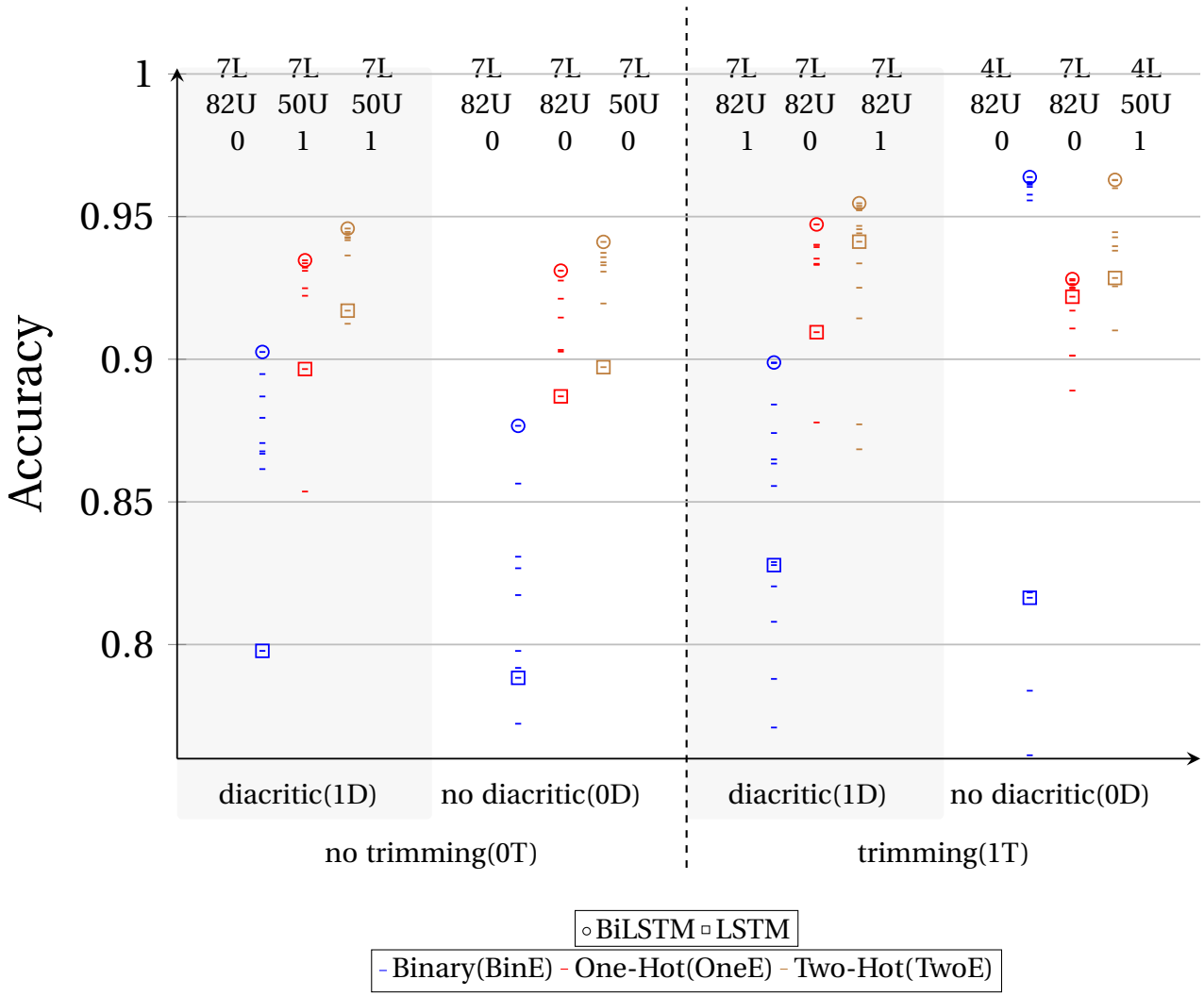


Figure 6.1: Overall accuracy of the 192 experiments plotted as 2 vertical rug plots (for the 12 different data representations: $\{Trimming, NoTrimming\} \times \{Diacritics, NoDiacritics\} \times \{OneE, BinE, TwoE\}$), each represents 16 exp. (for the 16 different network configurations: $\{7L, 4L\} \times \{82U, 50U\} \times \{0W, 1W\} \times \{LSTM, BiLSTM\}$). For each rug plot the best model of each of the four cell types —(Bi)LSTM labeled differently. Consistently, BiLSTM was the winner, and its network configuration parameters are listed at the top of each rug plot.

Experiment Id	Accuracy	Encoding Types	Diacritic	Trimming	Cell Type	Layers	Units	Weighted Loss
1	0.963847	*Binary*	D0	T0	BI-LSTM	*3*	*82*	*W0*
2	0.962883	*Two-Hot*	D0	T0	BI-LSTM	*3*	*50*	*W1*
3	0.962721	*Two-Hot*	D0	T0	BI-LSTM	*3*	*82*	*W0*
4	0.962105	*Binary*	D0	T0	BI-LSTM	*6*	*50*	*W1*
5	0.961955	*Binary*	D0	T0	BI-LSTM	*6*	*82*	*W0*
6	0.961189	*Binary*	D0	T0	BI-LSTM	*3*	*82*	*W1*
7	0.960404	*Binary*	D0	T0	BI-LSTM	*3*	*50*	*W0*
8	0.959949	*Two-Hot*	D0	T0	BI-LSTM	*3*	*50*	*W0*
9	0.957686	*Binary*	D0	T0	BI-LSTM	*3*	*50*	*W1*
10	0.955668	*Binary*	D0	T0	BI-LSTM	*6*	*50*	*W0*

Table 6.1: Top ten experiments results

exception, with very little difference, is (1T, 1D, BinE) vs. (0T, 1D, BinE). We need to highlight that this effect is logical, as the training will have fewer classes with a huge amount of data for these classes.

2. **Diacritics Effect**

   - *Without Trimming:* The effect of diacritics is obvious only with no trimming (the two left strips of the figure) where, for each encoding, the accuracy score is higher for diacritics than no diacritics.
   - *With Trimming:* The diacritics have no effect other than in the *one-hot* encoding, while other encoding has no effect on the model performance. This result is inconsistent with that anticipated, from the effect of diacritics. It is believed that

this result is an artifact due to the small number of network configurations.

3. **Encoding Effect:** The effect of encoding is clear; by looking at each individual strip, *overall accuracy score is consistently highest for two-hot then one-hot then binary* the only exception is (1T, 0D, BinE) which performs better than the other two encodings. It seems that *two-hot* encoding makes it easier for networks to capture the patterns in data. However, we anticipate that there is a particular network architecture for each encoding that can capture the same pattern while yielding the same score.

6.3 Network Configurations Effects

This section is to comment on the effect of the network configurations parameters.

- **Cell Type**: It is clear that BI-LSTM (large circle) has the highest accuracy score for each data representation. It is always higher than the highest score of the LSTM model (large square). This is what we expected; the more complex architecture, the more results we can achieve. However, we need to mention that the BI-LSTM is slower than LSTM in overall running time for all experiments, and also consumes much more resources than LSTM cell.
- **Layers Number**: Layers Number: As explained in Section Deep Learning Background, the idea behind the deep neural network comes from the multi-layers which make the network learn more details. Hence, the more complex the network (more layers), the more results we can achieve. In our experiments,therefore, we can show that 7 layers achieved higher scores than 4 layers. There is an exception for the trimming data without diacritics in (1T, 0D, BinE) and (1T, 0D, TwoE). The straightforward interpretation for this is that the reduction in dataset size occurred by trimming and no diacritics, which required a less complex network. Hence, reducing the complexity of our problem (the number of layers will not be effective).
- **Cell Units and Weighting Loss**: We cannot ascertain a consistent effect based on the number of cell units or the weighting loss, but we must mention that the highest results were achieved using both the highest cell units (82) and the weighted loss.

6.4 Per-Class (Meter) Accuracy

In this section, we explore the accuracy of each class. This addresses how our model can detect every class separately. It is similar to accuracy (see above) but involves per-class calculation. It is useful to check, as it will show how the model may understand each class, and which classes our model was unable to classify.

Similar to the previous section, we employ four combinations of *trimming and diacritic*: we investigate which models achieved the best results. We take the best four models (the first involve *two-hot* encoding, and the fourth *binary* encoding) from Figure 6.1, which represents the overall accuracy and shows the results of the per-class accuracy for each one.

In Figure 6.2 we have the previous four models display the per-class accuracy. We ordered The class names based on their data size per class. It explained previously in Figure 4.1 with the same order. If we compare the results for the four models accuracy scores were around 95% in Figure 6.1 and the per-class accuracy, we will find only 6 classes (which have around 80% of the total datasets) are around this value. But there are significant drops for some classes which make the figures line has dropped in the results.

Figure 6.2 shows the relation between the model accuracy results per-class and the dataset size per class; however, this trend was expected to be fixed from the weighted loss which has an inconsistent effect on the weighting loss for all the models. This inconsistent effect shows we need a new design which has an function which can solve this trending issue. The overall accuracy can be increased after trimming but there will be a gap between the accuracy per class the size of the data per class as the dataset is not balanced. Moreover, we can repeat this experiment, ensuring all classes have an equal size; we can therefore show the accuracy without the issue of data unbalance.This is developed in Section Discussion.

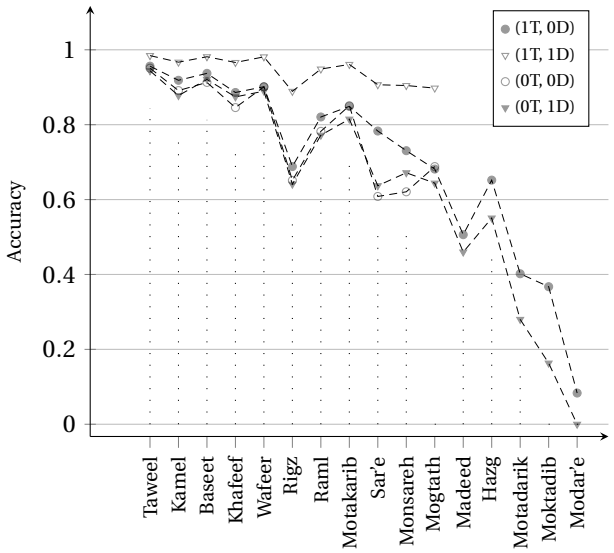


Figure 6.2: The per-class accuracy score for the best four models with combination of ({*Trimming*} × {*Diacritics*}); the x-axis is sort by the class size as in Figure 4.1. There is a descending trend in the class size, with an exception at *Rigz*

6.5 Encoding Effect

The difference between data encoding types is explained in Section Data Encoding. In this section, we will explore the effects of Data Encoding with respect to Accuracy, Learning Rate and Memory Utilization on the best model results (4L, 82U, 0W, 1T, 0D, BinE).

During our experiments, we found no consistent effect on the model encoding type and the model accuracy 6.3-a. However, in most cases the accuracy of the *two-hot* was found to be slightly better than *binary* and then *one-hot*.

Figure 6.3 shows the effect of encoding on learning rate, with no difference in convergence speed between the encoding types; However, we found some encoding starts learning faster than others between epochs [1:5]; but overall they converge with the same learning curve at the end.

Memory Utilization is not similar for each model encoding. We found that each encoding has it is own vector size representation with different memory utilization, based on the vector size; for example, the vector size of *one-hot* is $181 \times 8(bits)$, which will output $1,448$. If we compare this encoding with *two-hot* we find $41 \times 8(bits)$, WHICH will output 328. Comparing the previous two encodings with *binary*, we find it is the lowest memory consumption $8 \times 8(bits)$, outputting 64. It is apparent that *two-hot* is in the middle between the two encodings with respect to memory consumption, which gives more meaning for data encoding, as explained in Section Data Encoding

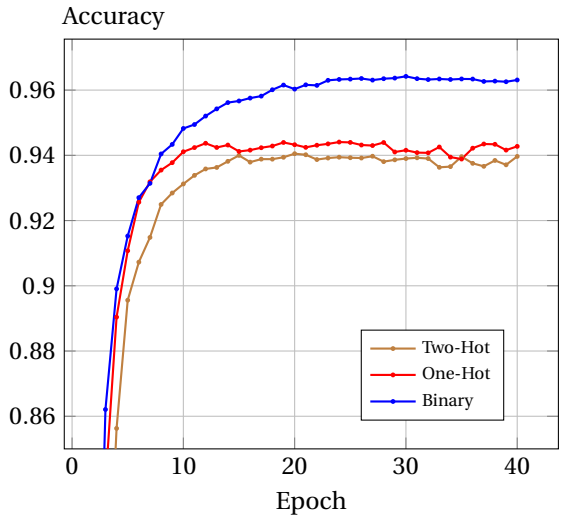


Figure 6.3: Encoding effect on Learning rate with the best model (1T, 0D, 4L, 82U, 0W, BinE) and when using the two other encodings instead of BinE.

6.6 Comparison with Literature

As explained previously, one of the advantages in our research work is our very large dataset, which affords us a good subset for testing. This provides us confidence regarding our results.

If we compare our work approach results, the best model scored 0.9638 with the highest two in the literature. We find that our model results are significantly higher than the others, as illustrated in Table 3.1. Moreover, our approach is a learning approach, not a hand-crafted algorithmic approach, which suggests our model is mature enough for these types of problems, as explained in Chapter Literature Review.

6.7 Classifying Arabic Non-Poem Text

In this section, we present the results of classifying non-poems' text (for example prose articles) to either the nearest meter or no meter if the former could not be matched with some scores to any meters. The motivation behind this trial is to check how well the model's network trained? We also need to check if the model can identify the pattern of the meter in any text (partially or fully following the meter pattern). We need to highlight that our model will identify the meter with some additional or missing characters. As we lack a clear definition correctly classifying or not classifying meters, we apply it based on probability scores.

We took a random sports article and passed each sentence to the Model to classify it according to the Arabic meters. The results for sample sentences are shown in Table 6.2. As explained previously, Arud rules are not concerned with meaning, only with the

Text	Meter (البحر)	Probability
خلال المباراة التي جمتهما مساء السبت بالجولة ال26 من المسابقة	الطويل	0.9949
والثالث بشكل عام في آخر 16 موسم بعد ستيفن جيرارد في موسمي	البسيط	0.9947
واصل صلاح هوايته المفضلة بالتسجيل أمام بورنموث للمباراة الرابعة على التوالي محققا العلامة الكاملة أمامهم	البسيط	0.9954
مبتعدا عن أقرب ملاحقيه الجابوني بيير إمريك أوباميانج الذي سجل 15 هدا	الطويل	0.9378
يعد بورنموث بوابة صلاح للعودة للتسجيل هذا الموسم في بريميرليج	الطويل	0.9988
عزز صلاح صدارة هدافي الدوري الإنجليزي راف رصيده إلى هدف	الطويل	0.7316
وهاري كين وسيرجيو أجويرو (14 هدفا لكل منهما)	المتقارب	0.8327
أصبح صلاح أول لاعب يسجل 20 هدفا أو أكثر في موسمين متتاليين مع ليفربول منذ الأوروجواياني لويس سواريز	البسيط	0.7075
ليتساوى بورنموث مع واتفورد كأكثر الفرق استقبالا لأهداف صلاح في بريميرليج برصيد 6 أهداف	الرمل	0.8422
لم يكمل صلاح ثلاث مباريات متتالية مع ليفربول في الدوري الإنجليزي بدون تسجيل هدف	الوافر	0.5246
، منذ انضمامه للريدز قبل انطلاق الموسم الماضي قادما من روما الإيطالي	البسيط	0.7561
قاد الدولي المصري محمد صلاح فريقه ليفربول للعودة إلى صدارة الدوري الإنجليزي الممتاز	الرمل	0.7385

Table 6.2: Sample Article Classification

music of the text. Hence, we can find if the text in the table matches one of our meters classes' full or partially with small issues. Thus, it does not follow the full rule for the meter. However, the model was able to identify it. We fed the model with text without *tashkeel*, which is harder than with *tashkeel*, but the model was able to establish all the nearest patterns and classify them correctly. An example from the table using the Arud writing style shows how the model was able to classify it. Consider the first row from the Table which belongs to *Al-Taweel* meter.

The example below shows the data cleansing applied to removing 26 from the text (Note: if the numbers are written with letters, it will be part of the text; otherwise the model will remove it). Note that the model was able to classify the text to the correct class. We can show the sequence of the example as follows; the first line is the text, the second line is the Arud writing style, the third line is the Arud writing equivalent music shapes as a sequence of *Harakat and Sukun*. The fourth line is the correct meter music sequence, and the last line is *Al-Taweel* meter *Tafa'il*.

Analyzing the example, there is only one issue in the meter *tafa'il*, highlighted in with red to show the additional *harakah*. The model can still identify the meter with some issues. However, if we remove this *harakah* the probability for this sentence to be classified to *Al-Taweel* will increase around 1%. This means the model can know the extent to which the text is relevant and close to the nearest *tafa'il*. We need also to highlight that *Al-Taweel* can appear with different *tafa'il* example فعولن ,فعول both are correct and the meter will be assumed correct with either of these shapes.

Example:

خلالِ المباراة التي جمعتهما مساء السبتِ بالجولة الـ26 من المسابقة
خلالْ مُباراتل لتيجْ معتهما مساءَسْ سبتْبلْجو لتلْمنلْ مسابقه
0/0// 0/0/0// /0// 0//0// 0/0/0// 0/0/ 0//0// 0/0/0// 0//0// 0/0/0// 0/0//
0/0// 0/0// /0/ 0//0// 0/0/0// 0/0// 0/0// 0/0// 0/0// 0/0// 0/0//
فعولْن مفاعيلْن فعول مفاعلن فعولْن مفاعيلْن فعولن مفاعيلن

In Figure 6.4 we see the score results for our testing data contain 159,998 correctly classified over 169,164 rows. These scores show that around 153,186 of the corrected results are above 0.94. Hence, if we filter any text with probability less than 0.94, we can identify if the text is related to the 16 classic meters or not. If we apply the same threshold to Table 6.2 we find all the sentences which above 0.94 will be correctly classified to the corrected meters (with small issues in its *Tafa'il*).

Another interesting example shows how the model can understand the pattern without *tashkeel*, and the effect of adding *tashkeel* to the text. We take the second row from Table 6.2. In the case without *tashkeel*, the model automatically tried to find any sequence of characters that follows any of the meter patterns. The model registered only the pattern including the *tafa'il*, not the mean of the text. The model could therefore successfully detect a sequence of characters which follows *Al-Taweel* with score 0.9988. In the case with *tashkeel*, the model understood the new pattern of *tafa'il* and gave the text low probability, below the threshold, and classified it as non-poem (not related to the 16 classic meters) with score 0.8390. This illustrates how *tashkeel* can add a feature for model understanding to the *tafa'il* pattern of the poem.

Example:

يَعِدْ بُرْنْمُثْ بَوَّابَةَ صَلَاحِ لِلْعَوْدَةْ لِتَّسْجِيلِ هَذَا الْمَوْسِمِ فِي بِرِمْرِجْ
يعدبر ثْمُثْبوا بةَصلا حلْلعودة لَتَّسْجي لهَذَاالْمو سمفي برمرجْ
0/0// 0/0/0// 0//// 0/0/0// 0/0// 0/0/0// /0/// 0/0/0// 0/0/0// /0/// 0/0/0//
0/0// 0/0/0// 0/0// 0/0/// 0/0// 0/0/0// 0/0// 0/0// 0//0// 0/0// 0/0///

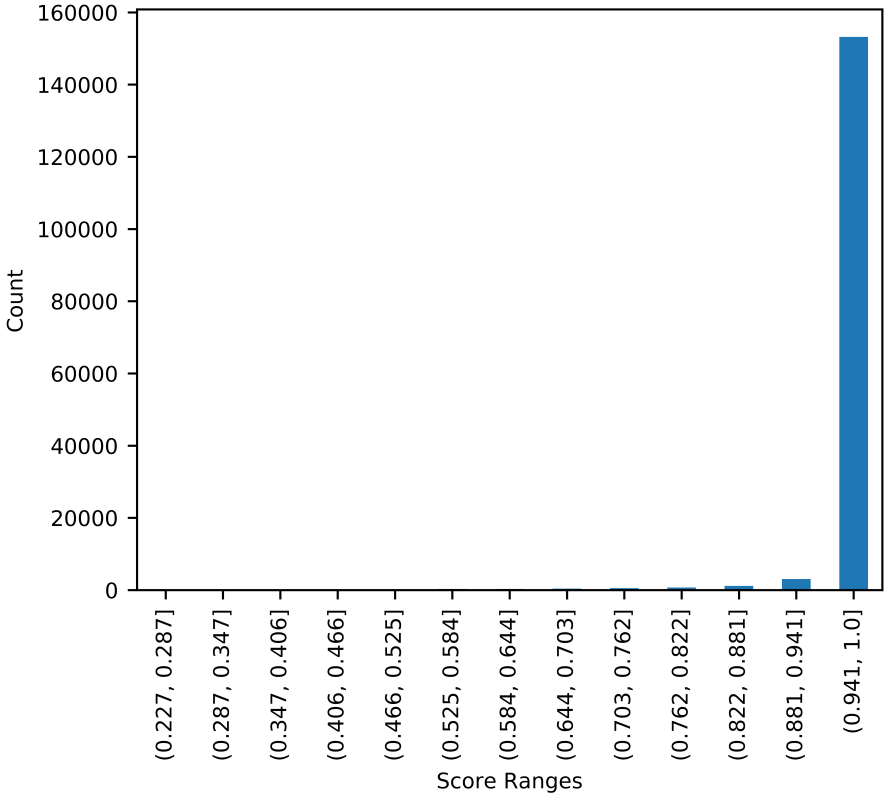


Figure 6.4: Testing data score ranges distribution.

مَفَاعِيلُنْ فَعُوْلُنْ مَفَاعِيْلُنْ فَعُوْلُنْ مَفَاعِلُنْ فَعُولُنْ مَفَاعِيْلُنْ فَعُوْلُنْ

6.8 Discussion

In this section, we discuss some points regarding our experiments and results, highlighting aspects we think it need more discussion or exploration.

6.8.1 Dataset Unbalanced

Our dataset was unbalanced, which certainly affected the results. Some significant drops are apparent in per-class accuracy, most of which concern the data size issue. We contend further work is required regarding this point, to reconstruct the experiments with balanced data, for example, 10k samples per class, and the results should be checked. Another approach could be to increase the size of the small classes to be at least 5% of the overall classes' percentage; this would enhance the learning accuracy of these classes.

6.8.2 Encoding Method

Although in theory all the encoding methods which carry the same information should produce the same results, in practice, Deep Neural Networks showed this is not the case. To explain the reason, consider how Neural Network works with different encoding mechanism.

The encoding method is a transformer function $\mathcal{T}$. This function transforms a discrete input values X. We can denote to the values as a transformed feature $\mathcal{T}(X)$, the output of this transformer method. The output $\mathcal{T}(X)$ of this transformer in the new encoding space will be input to the Neural Network model. The model should be able to "decode" this type of encoding. Since the lossless encoding is inevitable, it is clear that for any two functions and any two encodings $\eta_1(\mathcal{T}_1(X)) = \left(\eta_1 \cdot \mathcal{T}_1 \cdot \mathcal{T}_2^{-1}\right)(\mathcal{T}_2(X))$. This means that if the network η_1 is the most accurate network which can "decode" the encoding function (transformer) $\mathcal{T}_1$, this network η_1 is not a general network which can understand any encoding function. Moreover, to design this network requires a very complex architecture. Hence, if we have another encoding function $\mathcal{T}_2$ and we attempt to use the same network for the $\mathcal{T}_2$, designing another network is required $\eta_2 = \eta_1 \cdot \mathcal{T}_1 \cdot \mathcal{T}_2^{-1}$. However, this network may be need complicated architecture to "decode" the complicated pattern of $\mathcal{T}_2(X)$.

In general, any encoding function $\mathcal{T}$ requires a special network η to obtain the correct decoding (learning) for the dataset. Our comparison between the encoding methods in the same Neural Networks architecture is therefore not accurate, as each requires different network design. However, all will reach the same results but over a different time; or there can be a small difference due to the inaccurate network architecture. Moreover, our work illustrates clearly the effects of the encoding methods; and comparing them, we believe the *two-hot* encoding is the more suitable method to work with character level problems. It is the middle approach between *one-hot*, which needs a huge amount of memory, and *binary* which loses some meaning with the Arabic language diacritics effect.

6.8.3 Weighting Loss Function

Our weighting loss function does not solve the small classes issues (although the best model accuracy is achieved with weighting loss, but this is not a consistent result). The weighting loss function needs to be redesigned to solve this issue with the combination of learning rate and the batch size.

6.8.4 Neural Network Configurations

During our work, we show the effect of different network configurations on the model learning and accuracy. We conducted a range of experiments to find the best development architecture to facilitate make our experiments. In the beginning, the experiments took around 1.5 hours. Secondly, we proposed the multi-batch training to utilize parallel processing and prepare the data to the model faster, as our data was voluminous. We used enhanced an LSTM cell which reduced our experiments' duration to 7-9 minutes per epoch, based on the architecture of the network.

We also showed the effect of network layers on learning and accuracy results. Hence, conducting more experiments with more deep layers and more complex architecture can facilitate the acquisition of more language knowledge and build a more complex model which will enhance both the per-class accuracy and the overall accuracy.

6.8.5 Model Assessment

In our work, we proposed the overall accuracy score as the model assessment method for the results. However, we need to highlight that the overall model accuracy produced from the Deep Neural Networks was very close to the overall accuracy score calculated from the confusion matrix, and in some experiments was almost the same. We also explored various statistical ways, for example F_1 score, to assess our model and we find the model results will be the same.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis aimed to train Recurrent Neural Networks on Arabic Poems at character level to recognize their meters. This can be considered a step forward for language understanding, synthesis, and style recognition. This step aimed to understand how Deep Learning models can understand the Arabic Poem Meters and its rules and grammar. The datasets were crawled from several non-technical online sources; then cleaned, structured, and published to a repository that is made publicly available for scientific research. To the best of our knowledge, using Machine Learning (ML) and Deep Neural Networks (DNN), in particular for learning poem meters and phonetic style from a written text, along with the availability of such a dataset, is new to the literature. We described different encoding mechanisms, each with its own structure and understanding mechanism. We also experimented different RNN configurations including number of layers, cell units and types, with different language settings (with and without *diacritics*) We endeavoured to use different ways to handle the imbalanced dataset, showing the different accuracy and the effect of each one. The classification accuracy obtained on our Arabic Poem dataset was remarkable, especially if compared to that obtained from the deterministic and human-derived rule-based algorithms available in the literature.

7.2 Future Work

In this section, we outline future work which can be undertaken based on this research. We will divide the proposed future work into two parts; the first related to this idea and how can we enhance it, and the second, related to the new research area which can be built on our dataset.

- Enhancement on the current work:
  1. 1. The classification results could be enhanced to the same level as a human expert. We have many areas of enhancement; first, enhancing the network configurations with more layers with combinations of cell units and batch size. Secondly, there is an open area to solve the accuracy drops in the per-class performance issue in the small classes as follows:
     - New design of weighting loss function.
     - Increasing the dataset for the small classes, which will affect the learning and understanding of their patterns. \{}item Training the dataset with equal amounts of classes, known as desampling.
     - Duplicating some poems in the small classes.
  2. This problem can be treated as unsupervised learning, which will be a different approach to the problem-solving.
- Build new work based on the current dataset:
  1. Classify the correct and incorrect poems, respecting the nearest meter.
  2. Identify the issues in the written poems regarding the meter rules, and attempt to address them.
  3. Classify the poetry's meaning, as this paper did not work for this idea.
  4. Generate a new poem from learning the current classes and patterns.
  5. Analyze the historical impact on the poems and the poetry, for example for a specific period, if the poetry is affected by this period, or if there are patterns of writing between the poetry.

Bibliography

[1] Stathis, "How to read: Character level deep learning," 2017. [Online]. Available: https://offbit.github.io/how-to-read/

[2] N. Buduma and N. Locascio, *Fundamentals of Deep Learning: Designing Next-generation Machine Intelligence Algorithms*. O'Reilly Media, 2017. [Online]. Available: https://books.google.pl/books?id=EZFfrgEACAAJ

[3] "Collection of latex tikz figures." [Online]. Available: https://github.com/PetarV-/TikZ

[4] Colah, "Understanding Lstm Networks," 2015. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[5] G. F. Simons, "The 20th Edition of Ethnologue," 2017. [Online]. Available: https://www.ethnologue.com/ethnoblog/gary-simons/welcome-21st-edition

[6] M. Al-Metari, "القواعد العروضية وأحكام القافية العربية , محمد بن فلاح المطيري," 2004. [Online]. Available: https://archive.org/details/waq63764

[7] A.-K. A. tabrisi, الكافي في العروض والقوافي. Elkhangy, Cairo, 1994, vol. 3, http://waqfeya.com/book.php?bid=2373.

[8] A. Almuhareb, W. A. Almutairi, H. Al-Tuwaijri, A. Almubarak, and M. Khan, "Recognition of modern arabic poems," *JSW*, vol. 10, pp. 454–464, 2015.

[9] "Verse (poetry) wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Verse_(poetry)

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[11] D. R. Cox, "The regression analysis of binary sequences," pp. 213–230, 2005.

[12] "Convex function wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Convex_function#cite_note-1

[13] W. Rudin, *Principles of Mathematical Analysis*. McGraw-Hill Higher Education, 1976.

[14] "Wolfram alpha." [Online]. Available: http://mathworld.wolfram.com/ConvexFunction.html

[15] "Concave function wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Concave_function

[16] M. R. D. G. Richard, *The Secret Life of the Brain*. Joseph Henry Press, 2001.

[17] "Overfitting, wikipedia page." [Online]. Available: https://en.wikipedia.org/wiki/Overfitting

[18] "An overview of regularization techniques in deep learning." [Online]. Available: https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[20] T. Mikolov, M. Karafi, L. Burget, J. ernock, and S. Khudanpur, "Recurrent neural network based language model," in *Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH 2010)*, vol. 2010, no. 9. International Speech Communication Association, 2010, pp. 1045–1048. [Online]. Available: http://www.fit.vutbr.cz/research/view_pub.php?id=9362

[21] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, March 1994.

[22] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *CoRR*, vol. abs/1409.2329, 2014. [Online]. Available: http://arxiv.org/abs/1409.2329

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[24] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *CoRR*, vol. abs/1409.1259, 2014. [Online]. Available: http://arxiv.org/abs/1409.1259

[25] N. Reimers and I. Gurevych, "Optimal hyperparameters for deep lstm-networks for sequence labeling tasks," *CoRR*, vol. abs/1707.06799, 2017. [Online]. Available: http://arxiv.org/abs/1707.06799

[26] "An example of the architecture of lstm." [Online]. Available: http://www.paddlepaddle.org/documentation/docs/en/0.12.0/howto/rnn/rnn_config_en.html

[27] "F1 score wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/F1_scorex

[28] B. Abuata and A. Al-Omari, "A rule-based algorithm for the detection of arud meter in classical arabic poetry," *International Arab Journal of Information Technology*, vol. 15, 12 2017.

[29] M. Alnagdawi, H. Rashaideh, and A. Aburumman, "Finding arabic poem meter using context free grammar," *J. of Commun. & Comput. Eng.*, vol. 3, no. 1, pp. 52–59, 03 2013.

[30] A. Kurt and M. Kara, "An Algorithm for the Detection and Analysis of Arud Meter in Diwan poetry," pp. 948–963, 2012.

[31] M. Agirrezabal, I. Alegria, and M. Hulden, "Machine learning for metrical analysis of English poetry," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, Dec. 2016, pp. 772–781. [Online]. Available: https://www.aclweb.org/anthology/C16-1074

[32] C. Tanasescu, B. Paget, and D. Inkpen, "Automatic classification of poetry by meter and rhyme." in *FLAIRS Conference*, Z. Markov and I. Russell, Eds. AAAI Press, 2016, pp. 244–249.

[33] A. Estes and C. Hench, "Supervised machine learning for hybrid meter," in *Proceedings of the Fifth Workshop on Computational Linguistics for Literature*, 2016, pp. 1–8.

[34] J. Hopkins and D. Kiela, "Automatically generating rhythmic verse with neural networks," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 168–178. [Online]. Available: https://www.aclweb.org/anthology/P17-1016

[35] M. Agirrezabal, I. Alegria, and M. Hulden, "A Comparison of Feature-Based and Neural Scansion of Poetry," *Ranlp*, 2017.

[36] "الدّيْوَانُ," 2013. [Online]. Available: https://www.aldiwan.net

[37] "المَوسُوعَةُ الشِعْرية," 2016. [Online]. Available: https://poetry.dctabudhabi.ae

[38] W. A. Yousef, O. M. Ibrahime, T. M. Madbouly, M. A. Mahmoud, A. H. El-Kassas, A. O. Hassan, and A. R. Albohy, "Arabic Poem Comprehensive Dataset," 2018. [Online]. Available: https://hci-lab.github.io/ArabicPoetry-1-Private/#APCD

[39] "Hcilab, arabic poetry," 2018. [Online]. Available: https://github.com/hci-lab/ArabicPoetry-1-Private/

[40] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *J. Mach. Learn. Res.*, vol. 3, pp. 1137–1155, Mar. 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=944919.944966

[41] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12, pp. 2493–2537, Nov. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1953048.2078186

[42] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 3111–3119. [Online]. Available: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

[43] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[44] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-aware neural language models," *CoRR*, vol. abs/1508.06615, 2015. [Online]. Available: http://arxiv.org/abs/1508.06615

[45] J. P. C. Chiu and E. Nichols, "Named entity recognition with bidirectional lstm-cnns," *CoRR*, vol. abs/1511.08308, 2015. [Online]. Available: http://arxiv.org/abs/1511.08308

[46] J. Lee, H. Kim, M. Ko, D. Choi, J. Choi, and J. Kang, "Name nationality classification with recurrent neural networks," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017, pp. 2081–2087. [Online]. Available: https://doi.org/10.24963/ijcai.2017/289

[47] K. Potdar, T. S., and C. D., "A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers," pp. 7–9, 2017.

[48] M. Agirrezabal, I. Alegria, and M. Hulden, "A Comparison of Feature-Based and Neural Scansion of Poetry," 2017.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

[50] T. Zerrouki, "pyarabic, an arabic language library for python," 2010. [Online]. Available: https://pypi.python.org/pypi/pyarabic