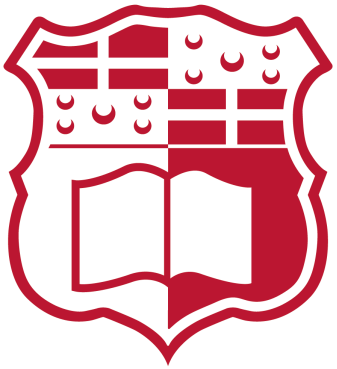
A Dependency Parser for the Maltese Language using Deep Neural Networks



Andrei Zammit

Supervisor: Dr. Claudia Borġ

Department of Artificial Intelligence

University of Malta

Submitted in partial fulfilment of the requirements for the degree of

Master of Science in Artificial Intelligence

June 2018

Dedicated to the Maltese language; may its legacy be a beacon to those who nurture this unique Semitic language, written in Latin letters, influenced by Romance Languages, spoken by just 400,000 people.

Declaration

STUDENT: Andrei Zammit 487677M

FACULTY: Department of Artificial Intelligence

COURSE: Master of Science in Artificial Intelligence

TITLE: A Dependency Parser for the Maltese Language using Deep Neural Networks

1. AUTHENTICITY OF DISSERTATION

   I hereby declare that I am the legitimate author of this Dissertation and that it is my original work.

   No portion of this work has been submitted in support of an application for another degree or qualification of this or any other university or institution of higher education.

   I hold the University of Malta harmless against any third party claims with regard to copyright violation, breach of confidentiality, defamation and any other third party right infringement.

2. RESEARCH CODE OF PRACTICE AND ETHICS REVIEW PROCEDURES

   I declare that I have abided by the University's Research Ethics Review Procedures.

   As a Master's student, as per Regulation 58 of the General Regulations for University Postgraduate Awards, I accept that should my dissertation be awarded a Grade A, it will be made publicly available on the University of Malta Institutional Repository.

Andrei Zammit

June 2018

Acknowledgements

I would like to express my deepest gratitude to my supervisor Dr. Claudia Borġ for her patience, guidance and encouragement which made this dissertation possible. I am honoured for having the opportunity to work with her and hope that in the future we can work together again. I would like also to thank Dr. Lonneke van der Plas and Dr. Slavomír Čéplö, two awesome individuals of great inspiration.

I am immensely grateful to my family and friends with their huge hearts, who have been of great support.

Finally, I would like to thank Kenneth, for deciding together to start this adventure during a dull night out over a couple of drinks. His friendship kept me sane during the late lonely dark nights.

It was a good decision!

Abstract

Tasks such as information retrieval, sentiment analysis and question answering require the processing of text analysis and natural language processing. Sentence parsing is one of the tasks performed in NLP to analyse the grammar structure of a sentence, with the aim of determining the relationships between the words in a sentence.

Whilst there are several parsers for many European languages, Maltese remains a low-resourced and low-researched language and currently there are no parsers for the Maltese language. This work investigates computational parsing of Maltese by using novel Deep Learning and source bootstrapping techniques, with the aim of contributing not only to the increase in computational resources for Maltese, but also to dependency parsing.

The evaluation of the parser was performed according to the Conference on Computational Natural Language Learning (CoNLL) standards and metrics. Experiments were conducted using datasets provided during CoNLL 2017 except for the Maltese language dataset which is provided directly by the author.

Results show an Unlabelled Attachment Score of 90% and Labelled Attachment Score of 86% by using a Quasi-Recurrent Neural Network (QRNN) with a bootstrapped data source of Maltese and other Romance languages. Bi-directional LSTM Neural Networks outperform QRNN by less than 0.2% in both metrics however, QRNN achieve a three-fold runtime performance over bi-LSTM. To our knowledge, this is the first time that QRNN is applied to the task of dependency parsing. The use of bootstrapped data sources is not documented in the published papers and proceedings of the 2017 shared task we reviewed.

Contents

List of Figures **ix**

List of Tables **xi**

1 Introduction **1**

1.1 Motivation 2

1.2 Aims and Objectives 4

1.3 Approach 5

1.4 Chapter Overview 7

2 Background and Literature Review **10**

2.1 Traditional methodologies 10

2.1.1 Dynamic programming and Eisner's algorithm 12

2.1.2 Constraint satisfaction 13

2.1.3 Transition-based approaches 14

2.1.4 Graph-based approaches 16

2.2 Neural Networks 17

2.3 CoNLL 2017 18

2.4 Deep Learning methodologies 21

2.4.1 Neural Network Optimizers 22

2.4.2 Word embeddings 25

2.4.3 Deep neural network architectures: RNN and LSTM 29

2.5 A bootstrapping approach for Maltese 33

2.6 The Universal Dependencies . . . 37
2.7 The Maltese Universal Dependencies . . . 41
2.7.1 Sample sentence from MUDTv1 . . . 46
2.8 Evaluation . . . 47
2.9 Conclusion . . . 48

3 Methodology 50
3.1 Maltese Word Embeddings . . . 50
3.2 Using Quasi-Recurrent Neural Networks . . . 53
3.3 The Parser . . . 56
3.4 The Bootstrapped Multi-source Treebank . . . 59
3.5 CoNLL 2017 Evaluation standard . . . 61
3.6 Conclusion . . . 63

4 Evaluation and Results 64
4.1 Evaluation Procedure . . . 64
4.2 Evaluation metrics . . . 66
4.3 Experiments . . . 66
4.4 Neural Network Optimization algorithms evaluation . . . 69
4.5 External Word Embeddings evaluation . . . 72
4.6 Bootstrapped Multi-source Treebank evaluation . . . 76
4.7 Neural Network Architecture evaluation . . . 80
4.8 Alternate Languages evaluation . . . 82
4.9 Summary of Experiments and Results . . . 83
4.10 Conclusion . . . 85

5 Discussion 86
5.1 Neural Network Optimization algorithms . . . 86
5.2 External Word Embeddings . . . 88
5.3 Bootstrapped Multi-source Treebank . . . 92

5.4 Neural Network Architecture 96
5.5 Alternate Languages 97
5.6 Contributions 99
5.7 Limitations 101
5.8 Conclusion 102

6 Conclusion **103**
6.1 Achieved Aims and Objectives 103
6.2 Future work 104
6.3 Final remarks 105

Bibliography **107**

Appendix A Universal Dependencies specifications **113**

List of Figures

1.1 Dependency structure for Maltese and English sentence 3

2.1 Word embeddings of capitals mapped to the countries after projection [40] . 27
2.2 Feed Forward Neural Network architecture 30
2.3 Recurrent Neural Network architecture 30
2.4 Elman's Network [23] 31
2.5 LSTM architecture 32
2.6 Dependency structure for sample English sentence. 41
2.7 Dependency structure for sample Maltese sentence. 47

3.1 Architecture of a traditional Convolutional Neural Network [33]. 54
3.2 QRNN architecture as compared to LSTM and CNN architectures [9] . . . 55
3.3 Graph-based parser architecture as proposed by Kiperwasser and Goldberg [31] 59

4.1 Evaluation using fasttext external word embeddings ID: 9 73
4.2 Evaluation using GloVe external word embeddings ID: 10 74
4.3 Evaluation using no external word embeddings ID: 11 75
4.4 UAS during training of models for experiments ID: 12 to 17 77
4.5 LAS during training of models for experiments ID: 12 to 17 78
4.6 Weighted LAS during training of models for experiments ID: 12 to 17 . . . 79
4.7 Training of model for experiment ID: 20 81

5.1 Optimizer performance through training phase for thirty epochs 87

5.2 Prediction metrics using different optimizers 89
5.3 Prediction metrics using different external word embeddings 91
5.4 Word embeddings plotted on three dimensional scatter-plot 92
5.5 Prediction metrics using single-source and multi-source treebanks 95
5.6 Prediction for LAS metric using alternate languages 100

List of Tables

2.1 Results for the best three language treebanks [58]. 35
2.2 Multi-source datasets [58]. 36
2.3 Multi-source projection models [58]. 36
2.4 CoNLL-U format [48]. 38
2.5 Universal POS tags [43]. 39
2.6 Universal features [43]. 40
2.7 Sample sentence from UD English treebank. 40
2.8 The Maltese Tagset v3.0 [48]. 44
2.9 The Maltese Tagset v3.0 mapped to Universal POS tags. 45
2.10 Sample sentence from UD Maltese treebank. 46

3.1 Sample sentence from MLRS. 52
3.2 Sample sentence from UD Maltese treebank with offset boundaries. 62

4.1 GPU Server 65
4.2 Software and frameworks 65
4.3 Experiments 68
4.4 Neural Network Optimization algorithms experiments 69
4.5 Prediction metrics using different optimizers 69
4.6 Loss of Optimization algorithms during training 70
4.7 Loss of Optimization algorithms during training 71
4.8 External Word Embeddings experiments 72
4.9 Prediction metrics with different external Word Embeddings 72

4.10 Bootstrapped Multi-source Treebanks experiments 76
4.11 Evaluation metrics using Bootstrapped Multi-source Treebanks 76
4.12 Performance using Bootstrapped Multi-source Treebank 80
4.13 Neural Network Architecture experiments 80
4.14 Evaluation metrics using different Neural Architecture 82
4.15 Runtime performance using bi-LSTM Neural Architecture 82
4.16 Alternate Languages experiments 82
4.17 Evaluation metrics for Alternate Languages 83
4.18 Experiments and Results 84
4.19 Treebank reference 84

5.1 Experiments reference for Figure 5.5 94
5.2 Comparing results from single-source and multi-source treebanks 94
5.3 Comparing results from multi-source treebanks to Tiedemann and van der Plas [58] 95
5.4 Prediction metrics using QRNN and bi-LSTM neural architectures 97
5.5 Runtime performance of QRNN and bi-LSTM neural architectures 97
5.6 Predicted metrics for English Language 98
5.7 Predicted metrics for Spanish Language 98
5.8 Predicted metrics for Uyghur Language 99
5.9 Predicted metrics for Kazakh Language 99

Chapter 1

Introduction

Parsing is a functionality that allows us to analyse the structure of a sentence and to check whether it is expressed according to a specified grammar. In computer science, parsing is used to analyse the structure and syntax of code prior to it being compiled and/or run, thus alerting the developer of any errors that require adjustment. In natural language, parsing allows us to see whether a sentence is appropriately structured [29]. Although humans might not necessarily learn specific grammar rules, they naturally have a sense of whether a sentence 'sounds' right or not. In the computational treatment of any language, parsing can provide information with relation to which part of a sentence is the subject, and which is the object. This type of information can then be used by other Natural Language Processing (NLP) tools which can analyse say particular relations between words.

There are a number of computational parsing approaches that can be used - the main two approaches are constituent-based parsing and dependency parsing. Constituent-based parsing analysis a sentence by splitting it up into sub-phrases like a noun phrase, verb phrase, etc. These phrases become the constituents in a sentence and a grammar would generally specify the order in which these constituents can occur [29]. Dependency parsing on the other hand focuses on the actual relations between words, such as the subject and the object, or words that modify other words. Which type of parser should be used depends very much on the type of application and end-goal of the NLP task at hand. For many NLP tasks, dependency

parsing provides sufficient information without the need to look at the full syntactic structure of a sentence.

The computational treatment of the Maltese language has lagged behind when compared to the development of other major European languages [50]. However, there have been a number of efforts aimed at improving the computational resource for Maltese, including the development of a part-of-speech tagger [25] and further research at the development of a morphological analyser [8, 7]. More recently, the development of a manually annotated set of sentences with their respective dependency parse trees (Maltese UD Treebank, [48]) means that it is now possible to experiment with machine learning techniques and create a dependency parser based on this annotated data. This research will be the first of its kind, looking specifically at the computational dependency parsing of Maltese by using this treebank. It will also be the first time that a particular deep learning architecture will be applied to parsing, with the aim to contribute not just to Maltese dependency parsing, but also to the broader field of dependency parsing in general.

This chapter provides an introduction to the research area and the particular goals and objectives of this project. It will then provide a brief overview of the work carried out and an outline of the remainder of the document.

1.1 Motivation

Modern Dependency Grammar theory is attributed to the work of the French linguist Tesniére [56]. Since then, a number of different grammar schemes have been proposed and evolved. However, the main idea behind this theory is that the syntactic structure consists of *words* linked by asymmetrical relations called *dependencies*. A dependency relation holds between a syntactic subordinate word, called *dependent* whilst the other on which it depends, called the *head* or *modifier*. A sentence will always have a *root*, a word which has no *head* itself, meaning that it is independent of all the words composing the sentence. An example of a dependency parse tree is illustrated in Figure 1.1 for a Maltese sentence. The English equivalent in this case also has the same parse tree, even though the languages are different.

This is one of the advantages of dependency parse trees since relations between words might be the same across some languages. Usually, the closer two languages are historically (e.g. both derived from Latin), the more similarity they would share in sentence structure and relations [45].

The dependency relations are represented with arrows originating from the **HEAD** pointing to the **DEPENDENT**. Each of these arrows is labelled, denoting the type of dependency relation between two words. For example, the noun *Ġanni* is dependent on the verb *tefa'*, and the label shows that the type of relation between the two words is that *Ġanni* is the subject of the word *tefa'*. By contrast, the noun *ballun* is a dependent of the verb *tefa'* and also acts as a head for the determiner *l-*. The dependency structures can be formally defined as labeled directed graphs where the words are represented by the nodes and the typed dependencies by labeled arcs.

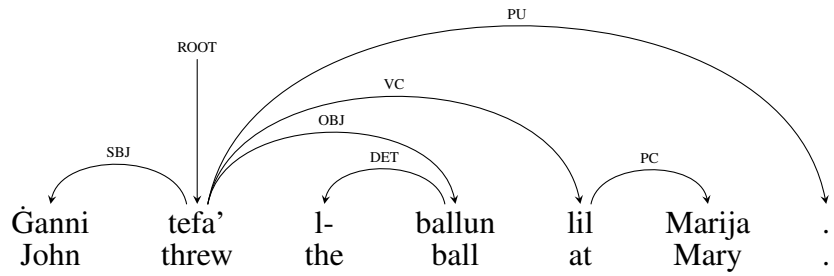


Figure 1.1 Dependency structure for Maltese and English sentence

Modern approaches to dependency parsing generally use an annotated treebank to train their models on, using machine learning techniques. Once a model is built, it is then possible to test the model on a portion of data which was not used in the training phase, referred to as the test data or unseen data. Treebanks are generally regular corpora, with the exception that they include an additional layer of annotation that specifies the dependencies between the words and their labelled relations.

Recent efforts by Čéplö [48] focused on the creation of a Universal Dependency Treebank for Maltese. A treebank is simply a corpus of sentences, and each sentence is annotated with its respective dependency tree — in this case, following the annotation guidelines of the Uni-

versal Dependency project[1]. The Maltese treebank consists of over 2000 sentences annotated with dependency parse trees. This recent development came at the opportune time for this research to take a machine learning approach for the development of a dependency parser for Maltese. We will also be exploring the possibility of using a novel neural architecture for the parsing algorithm and at the same time experiment with bootstrapping the learning of the model by using multi-lingual treebanks — thus looking at how other related languages to Maltese could contribute to the modelling of a parser for Maltese.

Recent developments in deep learning have also seen a shift in the type of machine learning algorithms used to train a dependency parser. The most recent research in dependency parsing has shown that the application of deep learning techniques improved results over other machine learning techniques, and this is something that this project aims to harness. The use of deep learning, coupled with resources such a word embeddings, might prove to be advantageous for the development of a dependency parser. In particular, word embeddings allow a neural network architecture to represent the semantic relations between words as vectors, thus providing the opportunity to represent words as something that a neural network can actually work with.

1.2 Aims and Objectives

The main aim of this work is to create a dependency parser for Maltese using machine learning techniques. A secondary aim is to investigate the use of novel techniques in deep learning to examine their effectiveness in the problem of dependency parsing. Thus, the proposed implementation of a dependency parser for Maltese is based on the latest Deep Learning technologies. The current state-of-the-art methodologies and architectures are reviewed in this report, and a Quasi-Recurrent Neural Network (QRNN) is chosen as the architecture since it seen both as a potentially novel approach to dependency parsing, but also that it will provide the necessary dependency parser model. The main contributions of this research are both in terms of the computational treatment of Maltese, as well as

[1] http://universaldependencies.org/introduction.html (Accessed: 2018-10-31)

dependency parsing in general. The stated aims will be accomplished by achieving the following objectives:

1. Design and implement a dependency parser for the Maltese language.
2. Generate word embeddings for the Maltese language and create a visualisation site that allows users to see the semantic relations between two words.
3. Evaluate and investigate the performance of the parser using a dedicated Maltese annotated treebank by applying standard evaluation procedures.
4. Determine the effectiveness of using a multi-lingual treebank to parse Maltese.
5. Compare and contrast results with published research.

1.3 Approach

The aims of this project are twofold. First, the development of the first dependency parser for Maltese, and secondly, to understand effectiveness of a Quasi-Recurrent Neural Network on parsing, a novel deep neural network architecture for the task of dependency parsing. Initial experiments were carried out to help determine the expectations of the processing time in order to better plan and schedule the experiments. Since this is the first time that a dependency parser is being built for Maltese, a number of experiments were carried out not only in terms of architecture, but also in terms of the input data. Two approaches are used to structure the input data; a Maltese language only training dataset and secondly training datasets composed of multiple language supporting the Maltese-only dataset. This bootstrapping approach using a multilingual training dataset does indeed provide better results than the Maltese-only dataset, since it provides richer information and more data sample points to the machine learning algorithm.

In the preparatory stage, features must be extracted from any given input sentence. The sentence is tokenised and word embeddings are generated for each word. Each word and punctuation mark is considered as a token forming the input. *Word embeddings* are a language

feature modelling technique whereby words from a corpus are mapped to vectors of real numbers. Part of the research is to create and experiment with creating a word embedding model based on the Maltese corpus. However, since Maltese is a morphologically rich language [8], problems arise when encountering word formations that were not originally part of the training corpus. For this reason, apart from experimenting with the Maltese model, experiments also looked at the effectiveness of using *fastText* [6], a word embedding model created by Facebook Research[2] and trained on multilingual corpora. The input sentence is also augmented with information pertaining to its part-of-speech category, thus providing further linguistic information about a word.

From a dependency parsing perspective, the experiments focus on two major strands. The first set of experiments simply use the dependency parse trees as annotated by Čéplö [48] — thus having a Maltese-only dataset. The second strand focuses on the multilingual approach, similar to the experiments carried out by Tiedemann and van der Plas [58]. The purpose is to compare whether augmenting the datasets with other languages related to Maltese would actually improve the results. The experiments focused on using data from Arabic, Hebrew, English, Italian and Spanish.

From a technical perspective, the experiments focused on the use of a QRNN — a type of deep learning architecture that is still relatively new in this field and to the best of our knowledge, has never been applied to dependency parsing. In this approach, the composition of the input sentence/features is fed into *tensors* which are multidimensional matrices on which the deep neural network can operate. With each token and its representation fed, the neural network would then proceed to output four vectors representing:

1. the token as a dependent searching its respective head
2. the token as a head searching its dependent tokens
3. the token as a dependent determining the lexical label
4. the token as a head determining the lexical labels of its dependent tokens

[2]https://github.com/facebookresearch/fastText (Accessed: 2018-10-31)

Once a sentence is completely processed, the resulting vectors of each token are fed into a neural classifier to compute a score. The dependency tree is then generated according to the highest scores of these input vectors. The approach is very much based on a similar parsing process used by Dozat et al. [18], which obtained the best performing results during the Conference on Computational Natural Language Learning (CoNLL) 2017.

The evaluation of the parser will be performed according the CoNLL standards and metrics. The experiments will be conducted using datasets provided during CoNLL 2017 except for the Maltese language dataset which is provided directly by the author. To perform the evaluation, the CoNLL 2017 evaluation script[3] was used. This ensures that this work follows a defined standard by an institution and the results can be compared with those achieved during CoNLL 2017. Results show that bi-directional LSTM Neural Networks outperform QRNN by less than 0.2% in the main CoNLL metrics. However, rather than using bi-LSTMs, this work focuses on experiments with QRNNs because of the superior runtime performance over the traditional bi-LSTMs.

A visualisation component is also included as part of the output for this research, with the aim to make the word embedding model for Maltese available as a a three dimensional scatter-plot. This results in a cloud with dense and less dense points indicating the semantic proximity of the Maltese words to each other.

1.4 Chapter Overview

This dissertation is structured as follows:

Introduction This chapter provided a brief overview of the work carried out in this research project, including the motivation behind undertaking this research, the aims and objectives of this work and the scope under which the research is carried out. It also provides a brief overview of the approach taken to tackle the set aims and objectives.

Background and Literature Review This chapter first delves into the theoretical aspects and approaches to grammar, sentence structure and dependency parsing, providing a

[3] http://universaldependencies.org/conll17/eval.zip (Accessed: 2018-10-31)

brief historical overview of how parsing evolved from the early traditional techniques to dependency parsing and to the adoption of neural networks and deep learning paradigms. This chapter then focuses on the shared task in dependency parsing, in particular highlighting the rise of deep learning techniques and their applied success in the field of dependency parsing. The only published work on dependency parsing for the Maltese language is also reviewed in great detail. Finally, the literature review also presents the annotation standards and the de facto framework for dependency parsing, with a particular focus on the annotations used for the Maltese Universal Dependency Treebank.

Methodology This chapter reports on how the research problem was tackled in order to achieve the aims and objectives by using the acquired knowledge from the Background and Literature Review chapter. Each method used and decision taken is supported with pertinent arguments. The whole process is throughly documented to ensure that the reader can replicate this work.

Evaluation and Results This chapter describes the experiments conducted to measure the performance of the different models and setups, according to a pre-defined evaluation plan. The evaluation is based on the same metrics as proposed by the CoNLL shared task in dependency parsing, so as to ensure the possibility to compare results, albeit using Maltese.

Discussion This chapter focuses on comparing and contrasting the results obtained against published work. It provides a technical overview in terms of performance and optimisers used. But it also provides an overview of the chosen approach, how it contrasts to other work, and how it fares when applied to other low-resourced languages. The final part of this chapter provides an overview of the contributions that this thesis makes to the field of dependency parsing in Maltese.

Conclusion The concluding chapter revisits the fundamental aspects in this dissertation and the methods used, together with the relevant experiments performed. The main results are highlighted in order to summarise the achievements of this work. Finally,

the chapter outlines how this dissertation met the aims and objectives initially set out, and proposes future work that can be undertaken to further research in this field.

Chapter 2

Background and Literature Review

In this section, we report on the different techniques used for dependency parsing. First, we explore the early approaches to dependency parsing and then progress to the use of neural networks and deep learning architectures. We also review a recent shared task in dependency parsing, CoNLL2017, which highlights the current challenges in dependency parsing and the prevalent use of deep learning to improve upon the results of the previous state of the art in dependency parsing. We evaluate in detail the only work performed on dependency parsing specifically for the Maltese language. Finally, we report on the defacto standard framework for dependency parsing — the Universal Dependencies framework and the Maltese treebank which has been released under this framework.

2.1 Traditional methodologies

Generally, traditional approaches to dependency parsing can be split into two distinct categories — *data-driven* or *grammar-based*. An approach is considered grammar-based when the methods it applies rely on a formal grammar specification which represents a particular language. An algorithm then detects if an input sentence adheres to the rules of the language defined by its grammar. An approach is data-driven when machine learning is used to learn automatically the grammatical relations between words from sentences annotated with their respective parse trees. Although these two categories are distinct, it is possible for depen-

dency parsing methods to adopt machine learning with the use of formal grammar and hence such parsing methods are both data-driven and grammar-based. This type of approach is referred to as a supervised technique since the machine learning algorithm is provided with labelled data as part of its training phase.

There are two computational problems to be solved in this process. The first is called the *learning problem*, which is the function of constructing a *parsing model* from a subset of the sentences and their corresponding dependency structures. The other is the *parsing problem*, which is the function of inferring the constructed model to an input sentence. In machine learning, the parsing problem is also known as the *inference problem* or the *decoding problem* which are terms used to describe the application of the model to data. The two problems are represented as:

- Learning: For a given training set of sentences D with annotated dependency structures, construct a model M to parse sentence S.
- Parsing: For a model M and sentence S, determine the dependency graph G for S by inferring M.

There are two classes of data-driven approaches called *transition-based* and *graph-based*. These two classes are distinguished by the algorithms used to construct the model from the input sentences and annotated data, the type of model constructed and the parsing algorithm adopted.

For most data-driven approaches, any input sentence is considered valid and the aim of the parser is to retrieve the best possible dependency structure for the input based on the model; regardless how much improbable it may result. By contrast, grammar-based approaches use a formal grammar as a core component of the model. Only a subset of the possible sentences are accepted by the model. Early approaches used manually created grammars, however later approaches were also data-driven and hence the grammar was learned from linguistic annotated data. Grammar-based parsing is also divided into two classes: context-free and constraint-based. Context-free dependency parsing uses a map between the dependency

structures and a Context Free Grammar (CFG) with the same parsing algorithms applied for the CFG. Constraint-based approaches tackle parsing as a constraint satisfaction problem.

The four classes are reviewed in detail and the most important research and results highlighted in the following sections.

2.1.1 Dynamic programming and Eisner's algorithm

One of the traditional approaches to describe a grammar is through the use of Context-Free Grammars (CFGs). A grammar is specified using a particular specification, and then a number of algorithms, such as Cocke-Younger-Kasami (CKY) [62] and Earley's algorithm [20], would process the grammar specification and output a parse tree for any given sentence. Initially, the aim of CFGs was to describe the structure of a sentence in terms of noun phrases, verb phrases, *etc.*, without going into the dependency relations between the actual words. However, parsing algorithms used to process a CFG can also be adapted for dependency parsing. Eisner's dynamic programming algorithm [22] is such a parsing algorithm that transforms a grammar specified in the CFG convention and extracts the type of word relations that are central to dependency parsing.

One of the problems with parsing of sentences is that ambiguity can easily arise — a typical example is: *'I saw the man with binoculars'*. This sentence is ambiguous because it is not clear who is with binoculars, whether 'I' or 'the man'. Techniques like the CKY or Eisner's algorithm use dynamic programming to deal with ambiguity since it provides all possible parse trees. An extension of the CKY is a probabilistic one which includes statistical information for each possible sub-parse-tree. There is also the problem of repeated parse trees during the processing. Top-down parsing (naïve search) is inefficient because each sub-tree might be created over each iteration. Dynamic programming offers an efficient way to record a particular sub-tree predicted over a specific range of the input sentence. Eisner's algorithm uses dynamic programming with a chart to keep track of partial derivations so no trees have to be re-derived. Chart parsing uses dynamic programming with a chart to keep track of partial derivations so nothing has to be re-derived.

The main idea behind Eisner's algorithm is to use a split-head representation in order to enable the chart cells represent half-trees instead of full trees. The two indices for the lexical heads are replaced by boolean variables indicating whether the head is at the left or right location of the respective half-trees. Two operations are required; the first for adding a dependency arc between the heads of two half-trees to form an incomplete half-tree. Hence, combining this incomplete half-tree to a complete half-tree to form a larger complete half-tree. The incomplete solutions are created and stored once. Furthermore, the algorithm never explores trees that are not potential solutions [22].

2.1.2 Constraint satisfaction

Constraint satisfaction uses a dependency grammar where every rule is assigned as a constraint on word-to-word relations. This constraint based parsing approach was introduced by Maruyama [35]. This work showed that a Constraint Dependency Grammar (CDG) parsing can be formalised as a constraint satisfaction problem. To lower structural ambiguity without the need to construct the individual parse trees, constraint-propagation algorithms are used where the transitional parsing result is represented as a data structure called a constraint network. The possible solution that satisfies all constraints concurrently is hence represented as a parse tree. To reduce disambiguation, new constraints are added to the network which are propagated using constraint propagation [35].

Menzel and Schröder [38] implemented a CDG parser for German using the same technique but added weights to the constraints. The German grammar was developed manually and consisted of nearly 700 constraints. Since this is a difficult and laborious task, this was extended by Schröder et al. [52], who used a machine learning approach based on genetic algorithms to assign weights. In the initial experiment Schröder et al. attempted to improve the weights of the manually crafted grammar which were achieved in the previous study by Menzel and Schröder [38]. Schröder et al. hence compared the results and acquired an increase of the f-measure from 96.9% to 98.4%. In the subsequent experiment the authors attempted to discover the required weights without referring to the weights discovered during the study by Menzel and Schröder [38]. The final result from this experiment is an f-measure

of 97.4%. Schröder et al. state that even if this is a significant result, it was not possible to attain the quality of the grammar achieved during the first experiment [52].

2.1.3 Transition-based approaches

A transition-based system is an abstract machine composed of states and transitions between the states. The simplest example of a transition system is a finite state machine, which consists of a finite set of states and a list of transitions which cause the machine to move from one state to another. The machine can be in only one state at any given point in time.

In dependency parsing, transition-based systems have complex states and the transitions correspond to the stages of a dependency tree derivation. The sequence of a valid transition for a given sentence starts from an initial state and ends in one of the possible final states. Such a full path traversed by the machine defines a valid dependency tree for a given sentence.

The oracle is an important component of transition-based parsers. The aim of the oracle is to predict the optimal sequence of transitions that will derive a specific gold tree for a sentence. There are two categories of oracles; static and dynamic. In static oracles, a set of rules is specified on which a single static sequence of transitions is produced. The major disadvantage is that a parser would often get deviated from the gold standard sequence and hence transitions to states which might not lead to the correct tree. On the other hand, a dynamic oracle permits all valid transition sequences leading to the gold standard tree rather than restricting a single sequence of transitions. Furthermore, a dynamic oracle is correct for all possible states even if such states do not reach the gold standard tree. In such scenarios, the dynamic oracle allows all possible states leading to a tree which has minimum loss compared to the gold tree.

A transition-based dependency parser is typically composed of:

1. the stack where to store the processed words of the input sentence
2. the queue where the rest of the words of the input sentence are stored
3. the transition actions which determine which transitions have to be performed according to the history of the stack and queue

The transition actions are defined by the arc standard transition system. An arc is a dependency between words in a sentence. The three transition operations as defined by Collins [13] are:

- LEFT-ARC - create a relation (arc, dependency) between the word at top of the stack and the second word on the stack. Remove the second word from the stack.
- RIGHT-ARC - create a relation between the second word of the stack and word on top of the stack. Remove the word on top of the stack.
- SHIFT - remove the word from queue and push it onto the stack.

One of the most successful strategies in parsing is to use a data-driven approach by applying classifiers on a treebank corpus. Classifier-based parsing is a very important component of transition-based dependency parsing. Parsing using this technique is a greedy search through the transition system, guided by the treebank trained classifier. This approach was first proposed by Yamada and Matsumoto [61] who achieved state-of-the-art results using the English language. The authors used the annotated Penn treebank [34] for the experiments. Yamada and Matsumoto used two main metrics to evaluate their work; dependency accuracy and root accuracy. The dependency accuracy is the number of correct parents in a tree divided by the total number of parents. Root accuracy is the number of correct roots divided by the number of sentences. Yamada and Matsumoto achieved a dependency accuracy of 90% and a root accuracy of 92%. The main disadvantage of this technique is that it requires multiple passes over the input.

Later, Nivre [41] developed the Nivre's algorithm which is an evolved transition system [41] based on Collins [13]. In Nivre's algorithm, the transition actions are defined by the arc-eager transition system which is defined as:

- LEFT-ARC - create a relation between the word in the queue and the word on top of stack and perform a REDUCE operation.
- RIGHT-ARC - create a relation between the word on top of the stack and the word in the queue and perform a SHIFT operation.

- SHIFT - remove the first word from the queue and push it onto stack.
- REDUCE - remove the word on top of stack.

Nivre further improved the algorithm by having the system perform a single deterministic pass over the input [42]. This improved system was evaluated during the shared task on dependency parsing of CoNLL 2007 on ten different languages [44].

2.1.4 Graph-based approaches

Transition-based approaches are based on a state machine for mapping a sentence to its dependency graph. The learning problem is to build a model that, given the state's history, is able to predict the next state. The parsing problem is to construct the optimal transition sequence for the input sentence.

In contrast, graph-based methods define a search space for possible dependency graphs for the input sentence. The learning problem is to compose a model for scoring the possible dependency graphs for a sentence. The parsing problem here is to locate the highest scoring dependency graph. This technique is called the Maximum Spanning Tree (MST) since the problem of finding the highest scoring dependency graph corresponds to the problem of finding the MST in a dense graph. The score represents the likelihood that a specific tree is the correct one for the given input sentence. The most essential property of graph-based parsing is that this score is assumed to propagate proportionally through all subgraphs of the dependency tree.

The most common graph-based approach is the Chu-Liu-Edmonds algorithm [12, 21] and a graph-based parser is typically built on the following four components:

1. the definition of the graph for the given dependency tree
2. the definition of the parameters
3. a method for learning the parameters from the labelled data
4. a parsing algorithm

Projective trees are the set of trees which match to the set of nested trees under the root node. Projective dependency parsers are strongly related to CFGs and hence a large part of the CFG parsing algorithms can be modified to parse projective trees. A dependency tree is non-projective if it contains at least a tree which is not projective. In a projective dependency tree, it is possible to graphically illustrate all arcs of the tree without any arcs crossing. This property is known as the planar property. The first extensive work on graph-based dependency parsing is attributed to McDonald et al. [36].

2.2 Neural Networks

Chen and Manning [11] were the first to propose and implement a neural network classifier for greedy, transition-based dependency parsing. Traditional parsers perform feature extraction based on templates. Lexicalized features are highly sparse, which is a common problem in many NLP tasks. However, in dependency parsing the problem is worse since parsing critically depends on word-to-word interactions. The problem of incompleteness is a problem in all hand-crafted feature templates. Even with expertise involved, it is impossible to include every conjunction of every useful word in the template. Feature generation and extraction is computationally highly expensive. During experiments Chen and Manning discovered that 95% of the processing time was consumed by the feature computation.

The sparsity problem was solved using low-dimensional, dense word embeddings. A word embedding is a function mapping words to a multi-dimensional vector. Similar words were expected to have close vectors. For example, the words ‘was’, ‘were’ and ‘is’ were represented as close vectors since they share a lot of similarity between them. The part-of-speech tags (POS) and dependency labels were also represented as a multi-dimensional vectors. The tags and labels are small discrete sets, however it was found that these still show semantic similarities like words. For example, in POS tags, NNS (plural noun) should be close to NN (singular noun) and in dependency labels, NUM (numerical modifier) should be close to AMOD (adjective modifier).

Incompleteness was solved by the neural network classifier. The classifier did not require to enumerate all possible combinations available of the features. Chen and Manning [11] employed a novel function called *cube activation function* in the neural network instead of the traditional sigmoid functions. Hence, every hidden unit was computed by a non-linear mapping. The cube activation elements were sourced from the three different embeddings: word, POS tags and dependency labels embeddings.

Matrix multiplication with low-dimensional vectors solved the expensive processing requirements. Chen and Manning precomputed the matrix multiplication of the top 10,000 most frequent words, all POS tags and dependency labels. Hence, rather than performing matrix multiplications, only a lookup in a table was performed at each iteration. This pre-computation step increased the speed of their parser by 8 to 10 times and was able to parse 1013 sentences per second with a 92% accuracy [11].

Chen and Manning also addressed the three stated problems and were the first to show that neural dependency parsing can outperform conventional parsers. Their experimental evaluations showed that their parser is superior to other greedy parsers, such as MaltParser [45] by Nivre et al., in both accuracy and speed.

Weiss et al. [60] at Google adopted this paradigm to improve the state-of-the-art parsing with the release of SyntaxNet[1] which was considered as the world's most accurate parser with over 94% accuracy on well-formed English text. Trained linguists on this task agree between 96 to 97% of the cases, indicating that parsers are approaching human performance [60].

2.3 CoNLL 2017

The Conference on Computational Natural Language Learning (CoNLL) is a yearly conference organised by the Association for Computational Linguistics (ACL) which focuses on statistical, cognitive and grammatical inference. One of the shared tasks of CoNLL 2017, called 'Multilingual Parsing from Raw Text to Universal Dependencies' [67], was dedicated to dependency parsers for an extensive number of languages that can operate in a real world

[1] https://research.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html (Accessed: 2018-10-31)

setting. This task was possible because in the previous years, de Marneffe et al. [15] started an initiative called Universal Dependencies (UD) which was setup with the aim to develop cross-linguistically consistent treebank annotation for many languages and hence facilitating multilingual parser development and cross-lingual learning.

UD developed 64 treebanks in 45 languages where 15 languages have two or more associated treebanks. For CoNLL the 2017 shared task of dependency parsing, amongst the treebanks eight were small and hence the whole dataset could be used for training. Out of the 45 languages, four were considered as surprise languages implying that these languages have not been previously released in UD and were only available one week prior to the evaluation stage. The aim of the surprise languages is to inspire participants to develop real multilingual techniques to parsing, utilising data from other languages [67].

For CoNLL 2017 two baseline parsers were used; UDPipe by Straka et al. [55] and SyntaxNet by Kong et al. [32]. The aim of the baseline parsers is to provide pre-trained models for all languages which participants could improve upon. Both parsers are open source and can be used without any restriction.

The ranking of each system was based on the main evaluation metric, Labeled Attachment Score (LAS), averaged over all test sets representing all languages. Other secondary metrics were used to evaluate the systems highlighting their strengths and weaknesses. The secondary metrics included tokenization, sentence and word segmentation F_1 scores. In the 2017 shared task, the top ranked system was a neural dependency parser based on Long-Short Term Memory (LSTM) networks submitted by the University of Stanford [18]. However, this system placed fourth on the surprise languages and second on the small treebanks classifications. The second ranked system was submitted by Cornell University, which was an ensemble of three parsers: one graph-based and two transition based [54]. This system ranked first on the surprise languages and small treebanks classifications. The third ranked system, by the University of Stuttgart, was also an ensemble of two transition-based and one graph-based parsers [4]. An important conclusion from CoNLL 2017 was that the surprise languages and those languages which had small treebanks were difficult to parse with the best accuracy under 50% [67].

The process employed by Shi et al. [54], representing the University of Cornell, consisted mainly of four phases; preprocessing, feature extraction, unlabelled parsing and arc labelling. Their system solely focuses on dependency parsing. Hence, the tasks of tokenisation, sentence boundary detection, POS tagging and morphological features were handled by baseline models generated by UDPipe.

According to Shi et al., there were two major challenges in CoNLL 2017. The first was that a large part of the datasets represent morphologically rich languages. Secondly, a considerable fraction of the languages have limited training data. These two challenges were encountered and tackled during the feature extraction phase. In this phase, the input sentence was split into words and each word had its features extracted twice; one based on character level and the other on word level. Amongst the most popular methods for word representation is through word embeddings however, this methodology does not provide enough information for morphologically rich languages. In order to overcome this challenge Shi et al. adopted bi-LSTM vectors to obtain character level representation which often result in better information coverage. The second challenge was tackled by transferring delexicalized information from more resourced language datasets to a target lower resourced language. For languages with low training data, the most linguistically similar languages were selected and delexicalized models were trained and applied to the target language.

The unlabelled parsing phase was also composed of two stages. In the first stage, the ensemble of parsers analyzed the input sentence, each producing a syntactic structure in parallel. In the second stage, a parsing algorithm was applied to the original sentence by also inferring the analysis produced by each parser in the first stage. This methodology is called reparsing which was defined by Sagae and Lavie [51]. The graph-based parser was based on Eisner's algorithm [22] and used MST [12, 21] for scoring. The transition-based parsers were based on dynamic programming; one arc–eager and the other arc–hybrid. For these parsers, two bi-LSTM vectors were used to reduce the large search space, one from the top of the stack and the other from the top of the buffer. The scoring during the reparsing stage was performed using Dozat and Manning [17], which was the same scoring algorithm used

by Stanford's University's winning entry [18]. For the arc-labeling phase, a labeler proposed by Kiperwasser and Goldberg [31] was used.

For the arc-labeling phase, a labeler proposed by Kiperwasser and Goldberg [31] was used. A predicted arc would have a head and a modifier. These tokens were concatenated and passed through a multi-layer perceptron (MLP). The output with the highest score from the MLP would be the potential label.

This system ranked second in the overall classification and first in the surprise languages and small treebanks classifications. The better results of this system in the surprise languages and small treebanks classifications over the system presented by the Stanford University[18] were attributed to the feature extraction phase [67].

Most teams used a single parsing model except for four teams which submitted ensemble systems. An ensemble system is composed of a set of individual parsers operating together. The two top ranked parsers used Recurrent Neural Networks (RNN) whilst the third used Convolution Neural Networks (CNN). Both of these architectures form part of a class of Machine Learning techniques called Deep Learning where neural networks are composed of several layers rather than the tradition single hidden layer. For this reason, we will now provide a brief overview and review of Deep Learning.

2.4 Deep Learning methodologies

The thought process of humans does not start from scratch every time. The human thought process is based on past experience and we use that experience to infer knowledge on current decisions to be taken. We humans can do this because our thought process is persistent. Traditional shallow neural networks cannot replicate this process.

The term 'deep' in deep learning refers to the number of layers which compose the neural network. In deep learning, each layer performs a transformation on the input data according to a function into an abstract representation such as a matrix. This representation is then fed into the subsequent layer to achieve more featured representation. The process of feeding the output of a layer as input to the next layer will continue until the neural network produces

a final result. Three of the most popular deep learning technologies are Recurrent Neural Networks (RNN), Long Short Term Memory (LSTM) networks and Convolution Neural Networks (CNN).

RNNs tackle the issue of persistent memory by simply implementing loops making use of sequential information and hence allowing information to persist. A RNN can be thought of as multiple copies of the same network, each passing a message to a successor, with the output being dependent on the previous computations. RNNs have a memory which captures specific information called dependency about what has been calculated previously. In theory, RNNs should be capable of handling long-term dependencies; information collected and attributed to the early computation stages of the network. However, in practice, RNNs fail in this problem. LSTM networks are a special kind of RNNs, capable of learning long-term dependencies and remembering information for long periods of time as their default behaviour. In CoNLL 2017, many of the top ranked parser systems made extensive use of LSTMs.

CNNs are typically used to classify images although these type of networks were also successfully used for NLP tasks. CNNs are composed of convolution and classification layers. A convolution is a function which measures the overlap of two distinct functions. The convolution layers perceive input and output as a three dimensional representation. Before the classification step, the three dimensional representation is flattened to a two dimensional.

More detailed information about these architectures is given in the upcoming sections. The Methodology chapter will describe how a compound of these architectures was used to achieve a novel dependency parsing system.

2.4.1 Neural Network Optimizers

A Neural Network Optimizer is an optimization algorithm which is designed to maximise or minimise an objective or error function. Typically, when the aim is to maximise, the term objective is used whilst when to minimise the term error is preferred. The optimization algorithm is a mathematical function based on the internal learnable parameters of a model.

The parameters also also known as *hyperparameters* which are fundamental for training efficiently and effectively the model to generate accurate results. Three of the most common parameters are the learning rate, the weight and bias. Hence, the use of an neural network optimizer is required to constantly fine-tune the values of such parameters which determine the model's training phase and its resultant output.

There are two major classes of optimization algorithms:

- First Order Optimization Algorithms which are algorithms that minimise or maximise a function using the gradient acquired from the function with respect to the parameters.
- Second Order Optimization Algorithms which use a second order derivative to minimise or maximise a function.

The derivative of a function is a scalar that measures the rate of change of the function's value. A second order derivative of a function is the derivative of a derivative of that function. The gradient is the vector representation of the derivative. The main disadvantage of Second Order Optimization algorithms is that such algorithms are very expensive to compute and hence are only used in particular cases. On the other hand, First Order Optimization algorithms are much more easy to compute and require less resources.

Gradient Descent is the most important and popular First Order Optimization algorithm. The aim is to find the minimum loss of the error function. The learning rate is the size of the step to be performed with the aim to reach a minimum. In simple terms, follow the downhill direction of the slope of a U-shaped graph until the lowest point (minimum) is achieved. This process of slowly trying to reach the minimum is known as convergence. This algorithm has four main challenges:

1. Optimal learning rate - to choose the most appropriate learning rate is difficult. A learning rate which is too small leads to slow convergence. A learning rate which is too high can prevent convergence.
2. Learning rate schedules - This is the schedule of updates for the learning rate during the training phase. Such schedule must be predefined prior to training phase and cannot adapt to the training dataset characteristics.

3. Learning rate updates - the same learning rate must be applied to all parameter updates.

4. Convergence - typically the error functions are non-convex (not a U-graph) which have a number of minima with different values. It is difficult to converge a non-convex function and the algorithm can get trapped in a sub-optimal minimum.

Various algorithms were developed to optimize Gradient Descent and overcome these challenges. The following is a description of the mostly used optimization algorithms in practice.

AdaGrad

This algorithm allows the learning rate to adapt based on the parameters by performing small updates for frequent and larger updates for sporadic parameters and hence this algorithm fits well when training sparse data. The main advantage is that the learning rate does not need any adjustments but on the other hand it has the disadvantage that the learning rate is always decreasing. The constant decrease of the learning rate is known as the *vanishing learning rate* problem. This algorithm has been proposed by Duchi et al. [19].

AdaDelta

This is an evolution of the AdaGrad algorrithm where the learning rate does not tend to decrease, thus solving the vanishing learning rate problem. This problem is solved by using an exponentially weighted moving average over a window of the history of updates. AdaDelta was proposed by Zeiler [64].

Adam and Adamax

Adaptive Moment Estimation (Adam) is an algorithm similar to AdaDelta with the addition that it keeps a history of the mean of the past gradients, consequently, Adam records two distinct histories. The moment is defined a specific point in the recorded history. In practise, Adam performs very well when compared to other similar adaptive algorithms because it converges in a significant short period of time, in an efficient way. Adamax is a modified

version of Adam which is more suitable for sparse parameter updates. Both Adam and Adamax were proposed by Kingma and Ba [30].

SparseAdam

This is a variant of Adam which is suitable for sparse training data. SparseAdam restricts the parameters updates only to the specific moments which caused an update in the gradient. Rather than an algorithm in itself, it is more of a specific implementation by PyTorch[2], the deep learning framework used for this study. This implementation was reviewed because it forms part of the experiments conducted.

SGD and ASGD

Stochastic Gradient Descent (SGD) performs a parameter update of each training data point. These frequent updates should help to discover better minima, however, this advantage may result in a problem because the frequent updates can cause the convergence to oscillate and never reach a minimum. The Averaged Stochastic Gradient Descent (ASGD) is an extension of SGD where the algorithm keeps a history of the average of the updated parameters. ASGD was proposed by Polyak and Juditsky [49].

RMSProp

RMSProp is an algorithm very similar to AdaGrad which is unpublished and available as an online resource[3]. To solve the vanishing learning rate, RMSProp dismisses the older gradient history and divides the learning rate by a running average of the recent gradients only.

2.4.2 Word embeddings

Natural language is highly effective for us humans to relate to the world. Together with emotions and body language, we can easily convey a message with little or no ambiguity.

[2]https://pytorch.org/docs/master/_modules/torch/optim/sparse_adam.html (Accessed: 2018-10-31)

[3]http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (Accessed: 2018-10-31)

This is because we have senses which enable us to implicitly describe and structure such complexity into our language. For a machine to learn a natural language it needs to understand how we humans observe and relate to the world. Word embeddings is a learned representation of natural text as a vector which is capable to capture morphological, semantic and contextual information.

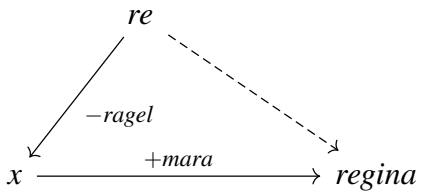
A vector is a series of real numbers and the amount of numbers in the series is called dimension. The dimension determines how much detail of the information is stored. Typically, word embeddings are generated using 100 dimensions; implying one vector is composed of 100 numbers, although 300 dimension is also common in research. The larger the dimension the more computational resources are required. The vectors offer the unique capability of performing mathematical operations on words. In a word embedding corpus, similar words should converge to the same locations in a three dimensional space. Similarity is calculated using the cosine distance between two vectors. The result of these operations is always a vector which has to be fetched from the embedding corpus. From the embedding corpus the closest, most similar vector has to be fetched. In practice, more than one vector is fetched because the distance between vectors (similarity) is very close.

Consider the words *re* (king), *reġina* (queen), *mara* (male) and *raġel* (female); the word embeddings should be located in the same vector space since these words are similar and related. The word embeddings are:

```
re -0.39504 -0.65514 0.39498, ..., -0.17863
reġina -0.36358 -0.9966, 0.51852, ..., 0.13695
mara 0.4302, -0.23856, 0.7076, ..., 0.74989
raġel 0.57179 0.21929 -0.43828, ..., -1.0287
```

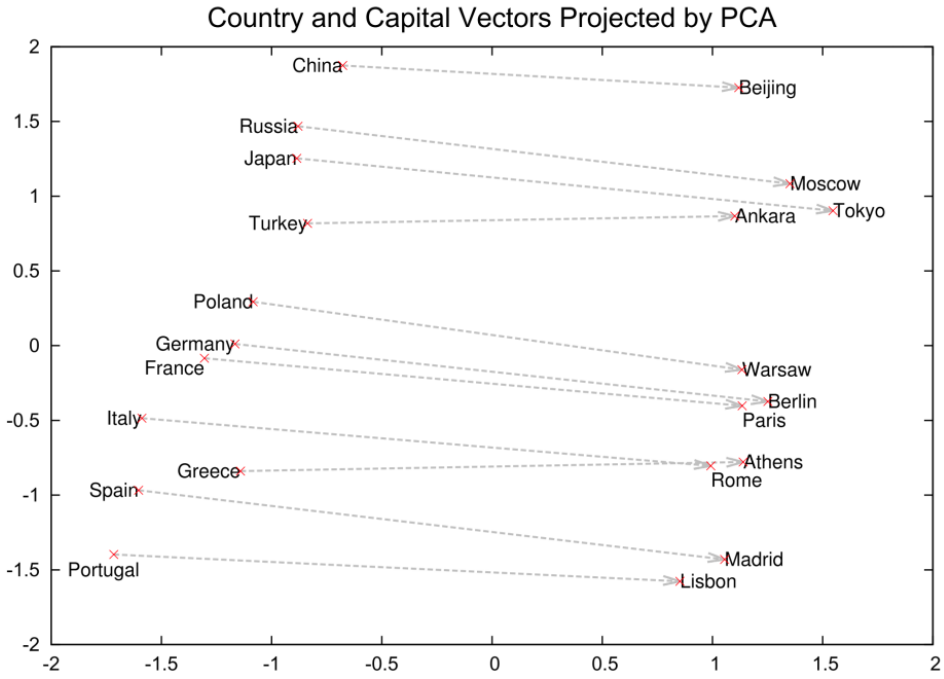
The following mathematical operation can be performed:

```
re - raġel + mara ≈ reġina
```



Mikolov et al. state that the directions and distances between vectors in the same vector space detail semantic relationships; for example male-female, verb tense such as walking-walked and even country-capital, relationships between words, as illustrated in the Figure 2.1. The authors also claim that the model is capable to organise concepts and learn the relationships between them. The model was trained in completely unsupervised manner and no information was given about what capital city means or relates to [40].

Figure 2.1 Word embeddings of capitals mapped to the countries after projection [40]



The three most important and popular implementations to generate word embeddings are *word2vec* by Mikolov et al. [39], *GloVe* by Pennington et al. [47] and *fasttext* by Bojanowski et al. [6]. Word2vec is the very first implementation with GloVe similar to it but uses a different algorithm.

Word2vec is primarily a predictive model whilst GloVe is a count-based model. In a predictive model, the algorithm tries to minimise the loss of predicting the target words from the context words using their vector representations. Count-based models build the

vectors by performing dimensionality reduction on co-occurrence counts. Co-occurrence counts are constructed in the form of a matrix which record the frequency of a word as a row and the context of that word as a column. Since the contexts are large by definition, the matrix is factorised to achieve lower dimensional version. In the case of GloVe, this process is performed by first normalising all counts and then performing a log function on the normalised counts. According to Pennington et al., this should increase the quality of the learned vectors.

Fasttext is the latest word embeddings implementation created by the Facebook Research for learning of word representation and sentence classification [6]. This library is currently being used by Facebook to deliver targeted adverts based on posts and status updates. Fasttext is an evolution of word2vec where each word is the aggregation of the character *n*-grams. According to Bojanowski et al., this should give more context to the word embeddings and produce more accurate results. Consider the word *reġina*; the word embedding is the sum of the vectors of *n*-grams:

```
<re, reġ, reġi, reġin, reġina>
```

One of the major advantages of fasttext is that better word embeddings for rare words can be generated because all words are decomposed to *n*-grams and these *n*-grams of rare words are shared with those of frequent words. In word2vec and GloVe, a rare word has very few neighbours to be associated with and hence it lacks context.

Out of Vocabulary Words (OVW) are words that do not appear in the training corpus. Fasttext can construct the word embeddings of such words because of the use of *n*-grams whilst word2vec and GloVe are incapable because a word is considered as one atomic entity. This is one of the reasons why Facebook is still capable of delivering targeted advertisements when the posts or status updates contain spelling mistakes.

One of the drawbacks of fasttext is that it requires more time to generate word embeddings than the other two implementations because each word in the corpus has to be decomposed into *n*-grams. Furthermore, as the corpus size increases, the computational resources required greatly increase because *n*-grams generation is exponential.

2.4.3 Deep neural network architectures: RNN and LSTM

Recurrent Neural Networks (RNN) are a type of powerful and robust neural networks especially adopted for sequential data. RNNs are the base of some of the most technologically advanced services such as Apple Siri and Google Voice. It is an important architecture because it was the first of its type which is capable of remembering the input by design of its internal memory.

Due to this design, RNNs are capable of precisely predicting future data patterns based on deep understanding and context and hence they are the preferred architecture for problems where sequential data is involved such as text, speech and financial data.

RNNs are based on Feed Forward Neural Networks (FFNN) which are more basic as architecture. In FFNN, the information moves in one direction only; from the input layer, through the hidden layers and finally out from the output layer as illustrated in Figure 2.2[4]. The information is channelled through a sequence of mathematical operations performed at the nodes. In a FFNN, information always touches a particular node just once. In such neural networks, there is no memory and the network only considers the current feed and hence a FFNN is not capable of determining the upcoming input. The input is transformed to output with supervised learning and the output is a label. A label is a name which is given to certain input. For example, a FFNN is trained on labelled images to categorise cats and horses. The FFNN would be trained until the error is minimised as much as possible when predicting the categories. The labelled images are fed into the FFNN in a sequential manner, meaning an image of *cat* a subsequent of a *horse*, the trained FFNN is not capable to perceive the next image. A FFNN needs to always process the current input with the consequence that it does not have a notion of order in time. The RNN architecture tries to solve this problem by using an internal short term memory. This is achieved by simply looping the input which was perceived previously as illustrated in Figure 2.3[5]. Therefore an RNN has two inputs; the current feed and the recent past feed.

[4]https://tex.stackexchange.com/questions/364413 (Accessed: 2018-10-31)
[5]https://tex.stackexchange.com/questions/364413 (Accessed: 2018-10-31)

Figure 2.2 Feed Forward Neural Network architecture

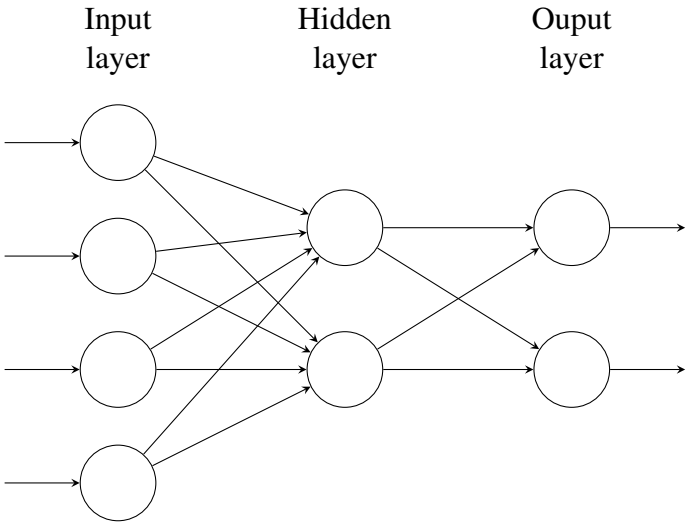
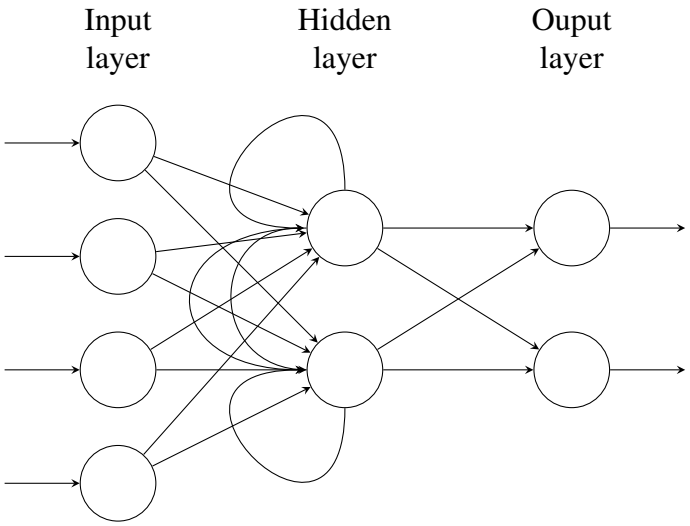


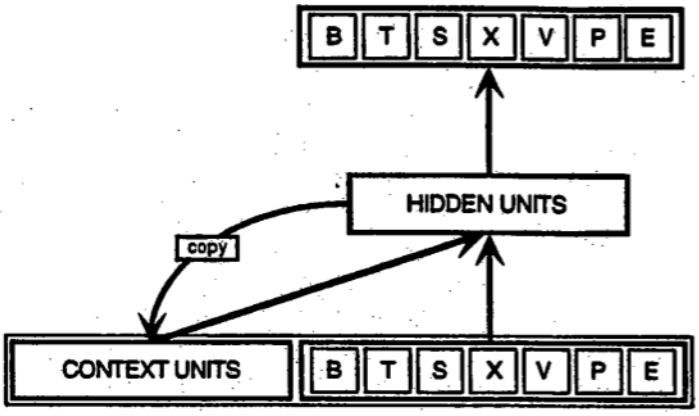
Figure 2.3 Recurrent Neural Network architecture



The main purpose of adding a memory to a neural network is that the sequence of data being fed into the network is information in itself and RNNs use this information to perform tasks; which FFNNs are not capable of. The Elman network was proposed Elman [23] which

is an early implementation of RNN which are also known as Simple Recurrent Network (SRN).

Figure 2.4 Elman's Network [23]



Such network is three-layered with a set of context units. This hidden layer is connected to the context units. Figure 2.4 illustrates Elman's implementation, where the text string BTSXVPE is the input string at time step *t*. The decision which was reached at time step *t-1* is contained in the context unit and will influence the hidden layer at time step step *t*. Like FFNNs, RNNs assign weights to the inputs but RNNs also assign weights to the context. These weights are modified through gradient descent and Backpropagation Through Time (BPTT).

Backpropagation is a process which channels back the final error through all the layers assigning a portion of the weights by calculating derivatives. BPTT is a series of backpropagation linking one time step to another.

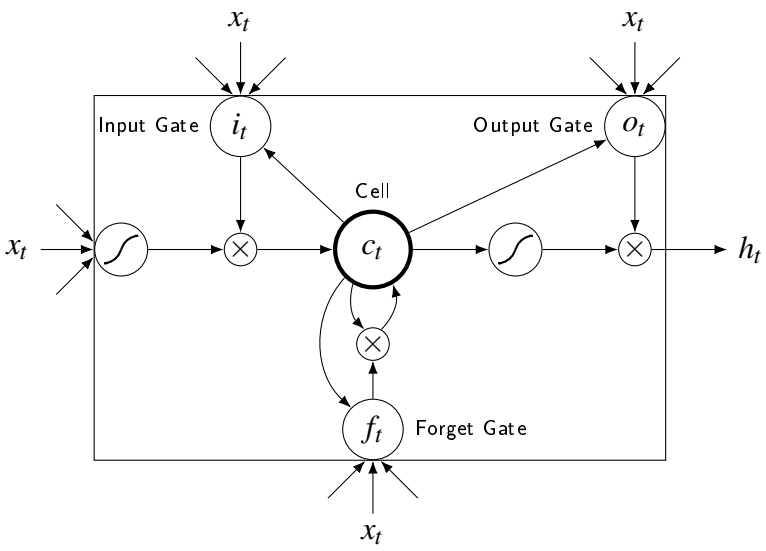
A FFNN is capable only to map one input to one output whilst RNN can map one to many, many to many and many to one. Example of applications of many to many are translations from one language to another and for many to one is voice classification and recognition.

Long Term Short Term Memory (LSTM) networks are an extension of RNN proposed by Hochreiter and Schmidhuber [27]. The main advantage of LSTMs over RNNs is that they are capable to maintain memory for a longer period of time through the iterations of the neural networks; thus the applied term *Long Term*. LSTMs can achieve this because it

is a memory unit which is capable to determine what to store, when to read and when to drop information. Figure 2.5[6] illustrates the LSTM unit decomposed with gates and flow of information. These operations are achieved by opening and closing three gates:

1. the input gate which determines if the input should pass through.
2. the forget gate which drops information which is not considered important.
3. the output gate which influences the output at time step *t*.

Figure 2.5 LSTM architecture



The gates are sigmoidal form meaning that their values range from 0 to 1. The forget gate was introduced by Gers et al. [26]. One major problem of the work by Hochreiter and Schmidhuber [27] was that the state of the LSTM network may grow indefinitely and eventually fail. Gers et al. solved this problem by introducing the forget gate which drop the information at appropriate times and hence gives resources back to the neural network.

Bidirectional Recurrent Neural Networks (BRNN) were proposed by Schuster and Paliwal [53]. The main idea behind BRNN is that the output at time step *t* may also depend on

[6]https://tex.stackexchange.com/questions/332747 (Accessed: 2018-10-31)

the future inputs. This is achieved by stacking two independent RNNs together which are trained simultaneously. For one RNN, the input is fed in normal time known as positive time direction. The other RNN is fed in reverse time and is known as negative time direction. Typically, the outputs of both RNNs are concatenated at each time step, with the possibility of other mathematically operations. The aim of this architecture is to capture more context which in turn improve the networks' output results [53].

This section explained in detail how the basic Feed Forward Neural Networks evolved to Recurrent Neural Networks which were in turn extended to Long Term Short Term Memory networks to improve efficiency and performance. Finally, Bidirectional Recurrent Neural Networks were discussed as another evolution.

2.5 A bootstrapping approach for Maltese

The work by Tiedemann and van der Plas [58], to our knowledge, is the only published work which evaluated methods of dependency parsing for the Maltese language. The authors focused their work on three widely used bootstrapping techniques; annotation projection, model transfer and translated treebanks. The authors considered the Maltese language to be an excellent test case to evaluate the performance and practicality of cross-lingual methods. Maltese is a computationally low resourced language with no parsers and an annotated dataset was not yet available at the time of this work. This meant that the authors had to bootstrap annotations for Maltese using resources from other languages, making the task harder than normal machine learning when annotated data is actually available.

Statistical approaches to natural language processing tasks require large annotated datasets in order to be able to perform reliably and accurately. The task of annotating datasets is highly time consuming, expensive and infeasible for large corpora. Automatic dataset annotation often fails because the annotation required is the information to be discovered and fetched. Bootstrapping techniques provide a viable way to manual annotation [59].

Early bootstrapping techniques concentrated on annotation projection which is also known as data transfer [28]. In this approach, the annotations from well-sourced languages

such as English and Spanish are projected to the target low-resourced language. Tiedemann and van der Plas used the heuristics as proposed by Hwa et al. [28] that make it possible to project the annotations from one language to another through bitext word alignment. Bitext word alignment is the process of identifying the translation relationships between words the bi-lingual parallel text. This process will result in a graph between the opposing sides of the parallel text. Projection rules are applied to the resultant graphs to create the tree structures required for the training process of the target language. The trees were further enriched with morphological features and POS information.

In model transfer, models are trained on annotated language sources and implemented onto the target language. This method works reasonably well only for closely related languages. To achieve good performance, the source and target languages must have significant lexical overlap else the models have to be delexicalised as part of the preprocessing stage. This method is considered to be the simplest transfer approach.

The third technique which was evaluated was translation of treebanks using Statistical Machine Translation (SMT) models trained on parallel datasets. This cross-lingual parsing would result in synthetic training data with the projected annotation from the source treebank. This technique was proposed by Tiedemann et al. [57] and evaluation results proved that this approach performed better than the annotation projection [58]. The two main problems of this approach is the lack of quality translation and sufficient training data for creating the models.

The Maltese treebank used by Tiedemann and van der Plas was the initial private version of research conducted by Čéplö [48] to release the first Maltese annotated treebank. The parser used in the experiments is a graph-based parser which forms part of the Mate Tools[7] developed by Bohnet [5]. The two principal approaches of the study; annotation projection and treebank translation depend on parallel translated datasets. This requirement was satisfied by using publicly available translated legal documents. The European Union's (EU) legislative documents, Acquis Communautaire (AC), is a large corpus of official documents accumulated over a long period of time. The Directorate General for Translation

[7]https://code.google.com/archive/p/mate-tools (Accessed: 2018-10-31)

(DGT-Translation) Memory is a large aligned translation memory of the AC covering the twenty-two official EU languages and their 231 language pairs where each sentence of the documents was professionally human translated. Hence, the DGT-Translation Memory was used to create these necessary parallel datasets.

For the evaluation, Tiedemann and van der Plas used a subset of 19 languages, including Maltese which had sufficient parallel translations. The corpus used for evaluation was composed of over one million translated sentences and a range of 19 to 26 million tokens per language. Maltese was tokenized using an in-house tokenizer whilst the rest of the languages were tokenized using UDPipe [55]. No specific details were published on the in-house tokenizer however, both tokenizers follow the standards of the UD treebanks. The best performing models were achieved using the English, Spanish and Italian treebanks. The translation treebank approach for the Spanish treebank achieved the highest Labeled Attachment Score (LAS) of 60.50% and highest Unlabeled Attachment Score (UAS) of 70.32% as illustrated in Table 2.1. From the results, it can be noted that Italian and Spanish are the languages which have the most lexical overlap with Maltese. The translated treebanks approach generally performed better than the other two approaches.

Table 2.1 Results for the best three language treebanks [58].

Treebank	Projection		Transfer		Translation	
	LAS	UAS	LAS	UAS	LAS	UAS
English	59.39	**69.53**	51.11	62.14	59.62	68.88
Spanish	**59.78**	69.41	55.54	**65.88**	**60.50**	**70.32**
Italian	57.70	66.74	**56.04**	65.11	60.35	68.80

Further to these experiments, Tiedemann and van der Plas tried to determine if multi-source models can be used to overcome the individual deficiencies of the projected data sets. Using the annotation projection technique, four further experiments were performed using two other datasets. The two datasets were composed of an aggregation of treebanks from English and Romance Languages and the another dataset was an aggregation of all treebanks as illustrated in Table 2.2.

Table 2.2 Multi-source datasets [58].

Dataset 1	Dataset 2
English	Bulgarian
Spanish	Czech
Italian	English
Portuguese	Spanish
Romanian	Italian
	Slovenian

The authors did not provide any reasons why the *annotation projection* technique was used instead of the *translated treebanks* method which performed better. One of the experiments was conducted with the inclusion of inflection information from the *Korpus Malti*. Inflection information details how words in language are modified in different contexts such as grammatical categories and tenses. These experiments resulted in minimal improvement over the previous experiments as illustrated in Tale 2.3.

Table 2.3 Multi-source projection models [58].

Method	Languages	LAS	UAS
Projection	All languages	62.51	71.54
Projection	Multi-source dataset 1	62.52	71.28
Projection	Multi-source dataset 2	62.77	**71.80**
Projection with inflection info	Multi-source dataset 2	**63.03**	71.54

At the conclusion of the final set of experiments, the authors stated that adding lexical information without contextual disambiguation provided insignificant increase in performance. The scores achieved in all experiments have limited practical value despite interesting for the research aspect. Tiedemann and van der Plas stated that cross-lingual parsing was still lagging behind fully supervised models. As possible future work, the authors recommend to study cross-lingual methods and their practicality over large-scale experiments which involve many more languages because the successful use of such techniques is still unproven. Such study and detailed analyses can comparatively explore the similarity between languages on specific linguistic levels.

2.6 The Universal Dependencies

Universal Dependencies (UD) is an open community initiative with the aim to develop a framework for cross-linguistically consistent grammatical annotation for a wide range of languages. This framework should promote research on multi-language parsers, parsing algorithms and cross-language learning [16]. The annotation scheme used in UD is the evolution of previous efforts by the University of Stanford, Google and Interset [48].

The first effort to standardise dependencies was by the University of Stanford. In 2005, Stanford developed a parser to support their Recognizing Text Entailment (RTE) systems. RTE systems are systems which are capable to map a directed relation between pairs of text expressions using inference. The Stanford Dependencies was the back-end component of the parser used in this system. During that period, the Stanford Dependencies evolved as the de facto standard for dependency analysis of English. At a later stage, these were adopted for other languages such as Chinese, Italian and Spanish [16].

The Google universal tag set was developed as part of the cross-linguistic analysis of the CoNLL-X shared task by McDonald and Nivre [37]. The tag set was used for the first time for unsupervised part-of-speech (POS) tagging by Das and Petrov [14]. Subsequently this work by Das and Petrov was used as a process for aligning the diverse tag sets to a standard by De Marneffe et al. [16].

The Interset Interlingua is a tool developed by Zeman which maps and converts different morphological tag sets amongst various languages [65]. This tool was later adapted as a morphological layer in HamletDT. The morphological layer determines how the individual words of a particular language are constructed. HamletDT is a collection of dependency treebanks and dependency conversions annotated with a standard tag set [66].

UD is the consolidation of these three works into one consistent clear framework based on the CoNLL-X format. The CoNLL-X format was defined during the previous editions of the CoNLL shared task by Buchholz and Marsi [10]. The format of the UD treebanks was later updated and named CoNLL-U with the initial guidelines published in October 2014. Subsequently, there were a number of releases of treebanks for various languages. In March 2017, the second version of the UD specification was released and in July 2018, the

second revision of this version will be published. Currently, UD is composed of more than 100 treebanks in over 60 languages with a number of upcoming treebanks for low resourced languages, including Maltese.

The CoNLLU-U format represent the dependency trees of that particular language in text format. The plain text file is UTF-8 encoded in order to support the large variety of characters of the different languages. Each entry must be stored in one line and there are three types of lines defined:

- Word line which is a word of a particular sentence with tab delimited fields.
- Empty line which marks a sentence boundary.
- Hashtag line which indicates a comment.

The word line contains the actual annotation defined by ten fields which are described in Table 2.4 and documented by Nivre et al. [43].

Table 2.4 CoNLL-U format [48].

Field	Layer	Description
1	ID	Word index, integer starting at 1 for each new sentence; may be a range for tokens with multiple words.
2	FORM	Word form or punctuation symbol.
3	LEMMA	Lemma or stem of word form.
4	UPOSTAG	Universal part-of-speech tag drawn from the revised version of the Google universal POS tags.
5	XPOSTAG	Language-specific part-of-speech tag.
6	FEATS	List of morphological features from the universal feature inventory or from a defined language-specific extension.
7	HEAD	Head of the current token, which is either a value of ID or zero (0).
8	DEPREL	Universal Dependency relation to the HEAD (root iff HEAD = 0) or a defined language-specific subtype of one.
9	DEPS	List of secondary dependencies (HEAD-DEPREL pairs).
10	MISC	Any other annotation.

The UD schema defines morphological and syntactical representations. The morphological representation is indicated through grammatical notions whilst syntactically through dependency relations. The three levels of morphological representation defined by UD are:

- Lemma which is the root of the word defined by the field LEMMA.
- POS tag which marks a sentence boundary defined by fields UPOSTAG and XPOSTAG.
- Feature set which show the characteristics of the word defined by the field FEATS.

The lemma is determined by the specific natural language dictionary whilst the POS tags and features are defined by UD. Some languages do not use all Universal POS tags whilst others require specific POS tags which are listed under XPOSTAG. The Universal POS tags are described in Table 2.5.

Table 2.5 Universal POS tags [43].

Tag	Class	Description
1	ADJ	adjective
2	ADP	adposition
3	ADV	adverb
4	AUX	auxiliary verb
5	CONJ	coordinating conjunction
6	DET	determiner
7	INTJ	interjection
8	NOUN	noun
9	NUM	numeral
10	PART	particle
11	PRON	pronoun
12	PROPN	proper noun
13	PUNCT	punctuation
14	SCONJ	subordinating conjunction
15	SYM	symbol
16	VERB	verb
17	X	other

The aim of features sets is to categorize words in a more precise way by offering additional annotations about the word which give more specific information on its speech and morphological properties. A feature is of the form *name=value* and every word can have any number of features assigned to it. The UD framework provides the list of features in Table 2.6 and each feature is described in detail in Appendix A. For a treebank to be officially considered as a valid UD treebank, fields 1, 2, 4, 7 and 8 are required whilst the remaining can be left empty marked by an underscore [43].

Table 2.6 Universal features [43].

Lexical features	Inflectional features	
	Nominal	Verbal
PronType	Gender	VerbForm
NumType	Animacy	Mood
Poss	Number	Tense
Reflex	Case	Aspect
Foreign	Definite	Voice
Abbr	Degree	Evident
		Polarity
		Person
		Polite

Listing 2.7 shows a sample sentence from the English UD treebank which can be downloaded from the Universal Dependencies repository[8].

Table 2.7 Sample sentence from UD English treebank.

# sent_id = reviews-187266-0007									
# text = This chef knows what he is doing.									
1	This	this	DET	DT	Number=Sing	2	det	2:det	_
2	chef	chef	NOUN	NN	Number=Sing	3	nsubj	3:nsubj	_
3	knows	know	VERB	VBZ	Mood=Ind	0	root	0:root	_
4	what	what	PRON	WP	PronType=Int	7	obj	7:obj	_
5	he	he	PRON	PRP	Case=Nom	7	nsubj	7:nsubj	_
6	is	be	AUX	VBZ	Mood=Ind	7	aux	7:aux	_
7	doing	do	VERB	VBG	Tense=Pres	3	ccomp	3:ccomp	_
8	.	.	PUNCT	.	_	3	punct	3:punct	_

The first two lines are hashtag lines which contain comments describing the sentence. The first comment shows the source of the sentence which is 'reviews' whilst the second comment is the actual text of the sentence. Although this is not a defined standard of commenting, many authors have adopted this style. The first word line has an ID value of '1' with word value of 'This' and a POS tag of type DET. From Table 2.5 it can be deducted that this POS tag is a 'determiner'. For display purposes, the feature set of each word line was reduced to

[8]https://github.com/UniversalDependencies/UD_English-EWT (Accessed: 2018-10-31)

one feature. The full feature set for this word line is 'Number=Sing|PronType=Dem' indicate features 'Number' and 'PronType' separated by a pipe. This word line depends on word line 2 which determines part of the dependency tree of the sentence. The full dependency tree is illustrated in Figure 2.6.

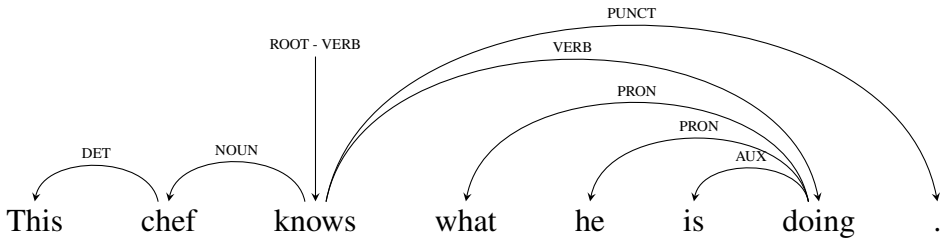


Figure 2.6 Dependency structure for sample English sentence.

2.7 The Maltese Universal Dependencies

For this work, a private first version of the Maltese UD Treebank (MUDTv1) was provided by Čéplö [48]. Private communication with the author reveals that this first version of the Maltese treebank was not intended to be fully UD complaint. The upcoming version, MUDTv2, due to be officially released in November 2018 will be fully UD complaint and distributed via UD website[9] with the other languages' treebanks.

Čéplö states that the UD framework was adopted to create the Maltese treebank because it has emerged as the de facto tree annotation standard for NLP functions. UD is well organised and coherent which can be confirmed by its adoption by the industry and growth of the available treebanks.

The source of MUDTv1 is two independent corpora which were published from earlier initiatives. The Maltese Language Resource Server (MLRS) hosts a number of Maltese corpora. The Korpus Malti v3.0 (2016)[10] is the latest available and was one of the corpus

[9]https://github.com/UniversalDependencies/universaldependencies.github.io (Accessed: 2018-10-31)

[10]http://mlrs.research.um.edu.mt/index.php?page=corpora (Accessed: 2018-10-31)

used by Čéplö. This corpus was tagged with the Maltese Tagset v3.0 (MTSv3) and has an accuracy of approximately 97%. This tagset is not compatible with the UD standard but is well documented[11]. Both the corpus and tagset were developed by Albert Gatt at the University of Malta. The other corpus is called bulbulistan maltiV3 (BCv3) which was developed by Čéplö, the same author of MUDTv1. Currently, Korpus Malti is provided as the main Maltese corpus whilst BCv3 is maintained for legacy purposes. Although these two corpora were developed independently, both works were published under one study by Gatt and Céplö [25] themselves.

The fields used in the Maltese UD treebank are the ID, FORM, LEMMA, UPOSTAG, XPOSTAG and FEATS. These fields are previously described in Table 2.4. It is important to note that the UPOSTAG and XPOSTAG tagsets are not UD standard and it is the use of these tagsets cause MUDTv1 not to be fully UD complaint. The following describe these fields as adopted by Čéplö with respect to the UD framework.

ID

This is the consecutive incremental integer used as the identification number of the *word* of the sentence. According to the UD framework, a word is a syntactic unit but also allow ranges to indicate multi-words. This is applicable to Maltese, however it was not implemented in MUDTv1 with the main reason being complications arising from the morphological analysis of the Maltese verbs.

FORM

The FORM field is based on the tokenization work of MLRSv3 and BCv3. Since Maltese is a morphological rich language, it was a difficult task to perform tokenization which fully follows the rules of the Maltese language. Hence, tokenization was performed as regular expression followed by rile based error corrections [48]. This process is described in detail by Čéplö in [48] Section 5.3.3.4.

[11] http://mlrs.research.um.edu.mt/resources/malti03/tagset30.html (Accessed: 2018-10-31)

LEMMA

In MUDTv1, this field was not used and was populated with an underscore as per UD specification.

UPOSTAG

The tagset used for this field is the Maltese Tagset v3.0 (MTSv3), the same used in MLRSv3 and BCv3 and not the UPOS tagset as described in Table 2.5. This is one of the reasons why MUDTv1 is not 100% complaint with UD. Part-of-speech (POS) tagging was manually applied to a subset of the two source corpora whilst the rest was applied using annotation software. Detailed information of the whole process is documented by Čéplö in [48] Section 5.4.1. Table 2.8 illustrates the MTSv3 as used in MUDTv1. Table 2.9 illustrates the possible mapping between UPOS and MTSv3. It is important to state that this map was not documented in [48] and the author left it to the reader to build it. Čéplö also notes that the most difficult part of building the map is when MTSv3 combines two word classes such as LIL_DEF and PRON_INT. This process of constructing the map must be performed to update MUDTv1 and make it fully complaint with the UD specifications. However, due to the fact that this map was not documented by Čéplö, there is the possibility that the map described in this work will not follow the reasons and decisions that will be adopted to release MUDTv2.

Table 2.8 The Maltese Tagset v3.0 [48].

ID	Tag	Description	ID	Tag	Description
1	FIX_THIS	Make corrections to this token	26	NUM_WHD	number *one*
2	_IGNORE_	ignore	27	PART_ACT	active participle
3	ADJ	adjective	28	PART_PASS	passive participle
4	ADV	adverb	29	PREP	preposition
5	COMP	complementizer	30	PREP_DEF	preposition with article
6	CONJ_CORD	coordinating conjunction	31	PREP_PRON	preposition with pronoun
7	CONJ_SUB	subordinating conjunction	32	PROG	progressive particle
8	DEF	article	33	PRON_DEM	demonstrative pronoun
9	FOC	focus particle	34	PRON_DEM_DEF	demonstrative pronoun with article
10	FUT	future particle	35	PRON_INDEF	indefnite pronoun
11	GEN	genitive particle	36	PRON_INT	interrogative pronoun
12	GEN_DEF	genitive particle with article	37	PRON_PERS	personal pronoun
13	GEN_PRON	genitive particle with pronoun	38	PRON_PERS_NEG	personal pronoun with negative suffix
14	HEMM	existential verb	39	PRON_REC	reciprocal pronoun
15	INT	interjection	40	PRON_REF	reflexive pronoun
16	KIEN	the verb kien	41	QUAN	quantifier
17	LIL	oblique particle	42	VERB	verb
18	LIL_DEF	oblique particle with article	43	VERB_PSEU	pseudoverb
19	LIL_PRON	oblique particle with pronoun	44	X_ABV	abbreviation
20	NEG	verbal negator	45	X_BOR	unclassified
21	NOUN	noun	46	X_DIG	digits
22	NOUN_PROP	proper noun	47	X_ENG	English words
23	NUM_CRD	cardinal numeral	48	X_FOR	other foreign words
24	NUM_FRC	fractions	49	X_PUN	punctuation
25	NUM_ORD	ordinal numeral			

Table 2.9 The Maltese Tagset v3.0 mapped to Universal POS tags.

ID	MTSv3 tag	UPOS tag	ID	MTSv3 tag	UPOS tag
1	FIX_THIS	X	26	NUM_WHD	NOUN
2	_IGNORE_	X	27	PART_ACT	PART
3	ADJ	ADJ	28	PART_PASS	PART
4	ADV	ADV	29	PREP	AUX
5	COMP	DET	30	PREP_DEF	PART
6	CONJ_CORD	CCONJ	31	PREP_PRON	PART
7	CONJ_SUB	SCONJ	32	PROG	PART
8	DEF	DET	33	PRON_DEM	PRON
9	FOC	PART	34	PRON_DEM_DEF	PRON
10	FUT	PART	35	PRON_INDEF	PRON
11	GEN	PART	36	PRON_INT	PRON
12	GEN_DEF	ADP	37	PRON_PERS	PRON
13	GEN_PRON	ADP	38	PRON_PERS_NEG	PRON
14	HEMM	ADV	39	PRON_REC	PRON
15	INT	INTJ	40	PRON_REF	PRON
16	KIEN	VERB	41	QUAN	ADJ
17	LIL	ADP	42	VERB	VERB
18	LIL_DEF	ADP	43	VERB_PSEU	ADV
19	LIL_PRON	ADP	44	X_ABV	SYM
20	NEG	VERB	45	X_BOR	X
21	NOUN	NOUN	46	X_DIG	NUM
22	NOUN_PROP	PROPN	47	X_ENG	X
23	NUM_CRD	NUM	48	X_FOR	X
24	NUM_FRC	NUM	49	X_PUN	PUNCT
25	NUM_ORD	NUM			

XPOSTAG

This field was populated with the same value of **UPOSTAG**.

FEATS

In MUDTv1, this field is empty and was marked with an underscore. Some work related to features was already performed by Čéplö and in the upcoming version, MUDTv2, the treebank will contain features as described in Section 2.6.

2.7.1 Sample sentence from MUDTv1

Table 2.10 shows a sample sentence from MUDTv1. The sentence in Maltese reads *'Qaltlu li mhux vera li x-xmajjar kollha għandhom memorji koroh.'* which translates to English as *'She told him that it is not true that all rivers carry unpleasant memories.'* Figure 2.7 shows the dependency tree of the sample Maltese sentence.

Table 2.10 Sample sentence from UD Maltese treebank.

1	Qaltlu	_	VERB	VERB	_	0	root	_	_
2	li	_	COMP	COMP	_	4	mark	_	_
3	mhux	_	PRON_PERS_NEG	PRON_PERS_NEG	_	4	neg	_	_
4	vera	_	ADJ	ADJ	_	1	ccomp	_	_
5	li	_	COMP	COMP	_	9	mark	_	_
6	x-	_	DEF	DEF	_	7	det	_	_
7	xmajjar	_	NOUN	NOUN	_	9	nsubj	_	_
8	kollha	_	QUAN	QUAN	_	7	det	_	_
9	għandhom	_	VERB_PSEU	VERB_PSEU	_	4	ccomp	_	_
10	memorji	_	NOUN	NOUN	_	9	dobj	_	_
11	koroh	_	ADJ	ADJ	_	10	amod	_	_
12	.	_	X_PUN	X_PUN	_	1	punct	_	_

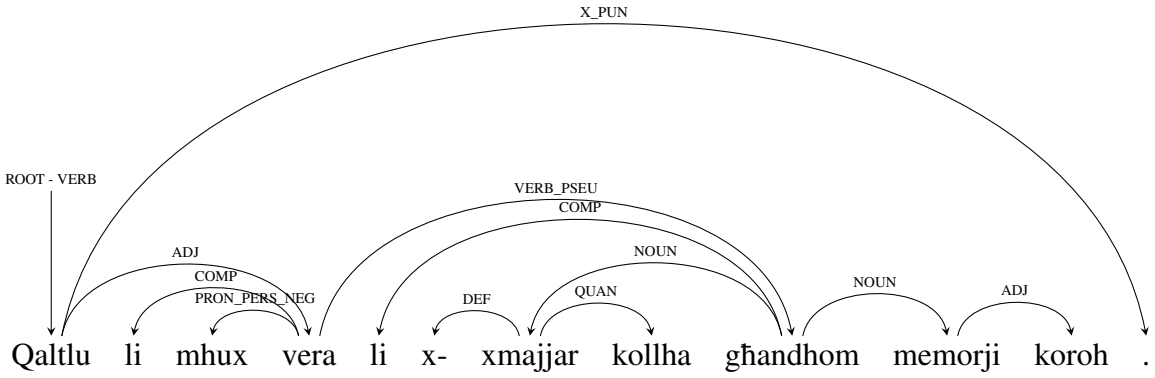


Figure 2.7 Dependency structure for sample Maltese sentence.

2.8 Evaluation

The standard evaluation procedure for dependency parsers is similar to other types of parsers; to apply the parser being evaluated to a test set. The test set is a subset taken from the annotated treebank which was not seen or used by the parser. The output of the parser is compared to the gold standard of the treebank.

The Maltese treebank will be split into a training and a test dataset with an approximate ratio of 80:20. The training dataset will be used to train the model and get the empirical risk. The empirical risk is the measurement of the performance of the model on the training data. The true risk is the accuracy of the model on the test data which is computed after building the model. The empirical risk must be in line with the accuracy of the model on the test dataset. Overfitting occurs when the empirical risk is almost zero and the accuracy of the test data is low. In such cases, the model is too specialised on the training data and does not generalise well. Overfitting is reduced through cross validation techniques such as random sub–sampling and k-folds approaches. The most widely used metrics are:

1. EXACT MATCH which represents the percentage of the parsed sentences with the exactly matched of the gold standard.

2. ATTACHMENT SCORE is defined as the percentage of words that have the correct head of the dependency tree. This metric is possible in depends parsing since the trees always have one head. On the other hand, evaluation of constituency-based parsing is based on precision and recall since it is not possible to exact match the constituents in the parser's output to those in the treebank.

3. PRECISION is the percentage of dependencies from the parser's out that were correctly parsed.

4. RECALL is the percentage of dependencies in the test set that were correctly parsed.

5. F-MEASURE is the result of the harmonic mean of the precision and recall.

These metrics can be used for evaluation in a labelled or unlabelled approaches. For this work, the following two metrics will be used for evaluation:

1. LABELED ATTACHMENT SCORE (LAS) represents the percentage of words that are assigned both the correct head and correct dependency label. This is the main evaluation metric for dependency parsing and the main evaluation metric for CoNLL [67].

2. UNLABELED ATTACHMENT SCORE (UAS) is the percentage of words that are assigned only the correct head. In every evaluation, UAS should always score better than LAS.

2.9 Conclusion

In this chapter, we reported in detail the various approaches for dependency parsing. Considerable attention was given to the techniques involving deep learning architectures, their use in CoNLL 2017 and the published work on dependency parsing for the Maltese language.

Further to these detailed analysis, the methods proposed by Kiperwasser and Goldberg seem to be the most appropriate choice for the current state of the art for Maltese dependency parsing. Given the time and hardware resources for this work, these methods are the most feasible. The parser by the University of Stanford [17] employed these methods and placed

first in CoNLL 2017 shared task. Another advantage that it also makes it possible to compare the evaluation of any techniques implemented with those obtained by Kiperwasser and Goldberg. Since the aim of this work is to also contribute to the field of dependency parsing, we will be using a Quasi-Recurrent Neural Network [9] as the main deep learning architecture, instead of the traditional bi-LSTM. To our knowledge, this architecture was never applied to dependency parsing.

Experiments will be conducted using the Maltese UD treebank and the results reported in detail in the upcoming chapters. The Maltese UD treebank will be further enriched with trees from other languages' treebanks with the aim to improve results. This process of bootstrapping of the source treebank was not documented by any of the participants of CoNLL 2017. Shi et al. [54] in their work employed similar techniques but by employing parallel datasets and inferring models on lower resourced languages. Our process is easier to implement and execute but should still achieve better evaluation metrics.

Chapter 3

Methodology

This chapter will detail the methods and processes employed to achieve the stated aims and objectives. These are based on the knowledge acquired during the previous chapter and are all justified with the necessary reasons.

3.1 Maltese Word Embeddings

One of the first tasks required for this type of work is the ability to represent words in a description that can be used by a neural network architecture. Word embeddings offer this functionality since words are represented as vectors. Moreover, once the word embeddings are generated, they can be mapped into a three dimensional scatter plot. This section describes the sequence of tasks required, starting from the source text from which the word embeddings will be generated. From the word embeddings, a model is created and finally the model is mapped to the scatter plot.

The first step in generating word embeddings for the Maltese language is to acquire a source text in Maltese. Bojanowski et al. [6] published pre-trained word embeddings for 294 languages, including Maltese using `fasttext`. The vectors are in 300-dimension and were generated using the skip-gram model and parameters as detailed by Bojanowski et al. [6]. The authors used Wikipedia articles as their textual source through which the word embeddings were modelled. There are some disadvantages in using this corpus. Since the vectors are

in 300-dimension, more computing resources are required and furthermore training will be more time consuming. Analysis of the corpus reveals that it contains several html tags and keywords which also ended up as part of the word embeddings model. Bojanowski et al. performed the necessary cleaning procedures of the Wikipedia articles. However, in practice, it is impossible to automatically acquire a completely clean text source from these articles and the only way would be to manually check for these type of occurrences that need to be removed manually. Another disadvantage is that the use of Wikipedia as the sole source for training limits the diversity of the text — especially when working with Maltese which has a limited number of entries/article[1]. Diversity is an important aspect for creating a word embedding model since it gives different context to words. In this research, a word embedding model was created using the MLRS corpus as a source for the Maltese text. The corpus is more diverse that the content of the Maltese Wikipedia and has more text, thus making it more appropriate as a textual source for word embeddings.

The MLRS corpus (also known as *Korpus Malti*, Gatt and Céplö [25]) was developed using a variety of texts as source including fiction and non-fiction works, academic writings, legal documents and news-website articles. This variety of sources should cover a wide range of subjects and contexts. A sample annotated sentence from MLRS is illustrated in Table 3.1.

To recreate the source, the tokens of each word line have to be extracted and sequenced into one sentence. For many tokens, MLRS also includes the lemma for that token. A lemma is the root of a word from which the word originates. The lemma does not originate from the sources of MLRS but was added as part of the annotation task. The lemma can be exploited for better word embeddings. As discussed in Section 2.4.2, similar words will result in closely related vectors. The lemma should be located close to the centre of a vector space and should also enhance context by adding more neighbours.

In the process used, the lemma of a word is inserted after the word token to form a sentence which grammatically and syntactically does not make sense but which should result in an enriched vector. Taking the sample sentence shown in Table 3.1, the composed sentence with the lemmas inserted would be as follows:

[1] As of May 2018, there were 3,368 articles on the Maltese Wikipedia — https://stats.wikimedia.org/EN/ChartsWikipediaMT.htm

Table 3.1 Sample sentence from MLRS.

Word	Tag	Lemma	Root
Libset	VERB	null	null
iż-	DEF	il-	null
żarbun	NOUN	żarbun	ż-r-b-n
tat-	GEN-DEF	ta’	null
takkuna	NOUN	null	null
għolja	ADJ	null	null
u	CONJ-CORD	u	null
rqiqa	ADJ	rqiq	r-q-q
u	CONJ-CORD	u	null
għamlet	VERB	għamel	għ-m-l
żewġ	NUM-CRD	żewġ	null
passi	NOUN	passa	p-s-j
.	X-PUN	.	null

```
Libset iż- il- żarbun tat- ta’ takkuna għolja u
rqiqa u għamlet għamel żewġ passi passa.
```

Note that if the lemma is the same as the word token, the lemma is simply not inserted. To process the MLRS corpus and generate all of the sentences, a C# tool was developed. This tool opens a stream to MLRS and reads one annotated sentence per step. The word lines of the MLRS are parsed for the word token and lemma, which are then queued sequentially in an array. At the end of the step, the array is iterated and a sentence formed. Finally, the sentence is flushed to a text file. This process is performed for all sentences of the MLRS corpus. The final result is a text file which is over four gigabytes large. This was the textual source on which the word embeddings were generated.

As discussed in Section 2.4.2, the word embedding models will be generated using fasttext [6] and GloVe [47]. Implementations of both algorithms are available from public repositories. Furthermore, the default parameters were used in this research, with only the 100 dimension parameter being specified. Zeman et al. state that for CoNLL 2017, a 100 dimension was chosen after a thorough discussion between the organisers and authors. This value is expected to yield good results and previous work by Andor et al. showed that it is

possible to achieve state-of-the-art results with just a 64 dimension. Both implementations generated two files each; one text and the other binary based. For this work, the text-based corpus is used since a client library would be required to load the binary format. Furthermore, using the text-based word embedding corpus gives the possibility to interchange between the corpora without any changes to the code. GloVe also generates a vocabulary file which lists the token words and their respective frequencies.

One of the stated objectives is to map the word embeddings corpus to a three dimensional scatter plot. TensorFlow and Tensorboard [1] are used in this research to achieve this objective. One of the tools of Tensorboard is the Projector[2] and this tool is used for visualisation. Three files are required for a successful mapping:

1. a 'tsv' file which contains the labels. In this case the labels are the actual words.
2. a 'ckpt' file which is the model file that actually contain the vectors.
3. a 'log' file which contains the meta data of the model.

A Python tool was developed to create the files. The first step was to load the whole word embeddings corpus into a list structure. From this list a tensor is created. A Tensorflow Interactive Session[3] is initialised and the created tensor loaded. The labels (words) are written to the 'tsv' file during the session. A Tensorflow Summary Writer[4] was used to write the 'ckpt' model file to disk and the respective meta data written to the 'log' file. The log file is required to have a complete model. To load the Projector with the map, Tensorboard is called with a parameter indicating the directory where all three files should be located.

3.2 Using Quasi-Recurrent Neural Networks

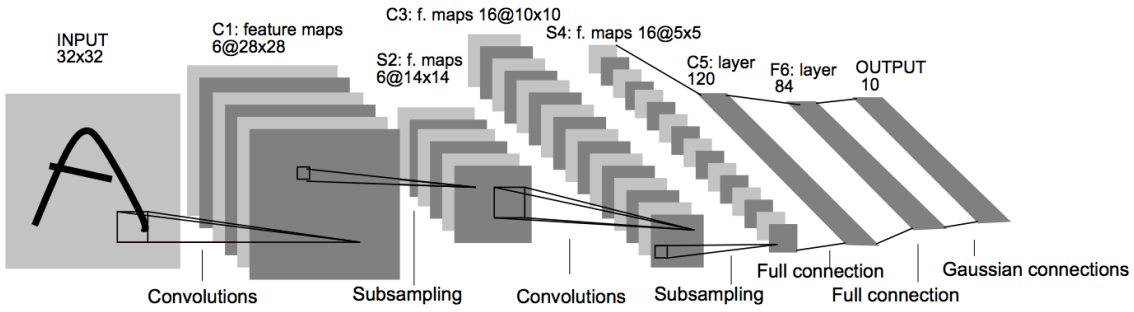
In deep learning there are two distinct ways to process input; sequentially or simultaneously. Sequential processing is associating with sequential data such as voice and text data while

[2]https://www.tensorflow.org/versions/r1.2/get_started/embedding_viz (Accessed: 2018-10-31)
[3]https://www.tensorflow.org/api_docs/python/tf/InteractiveSession (Accessed: 2018-10-31)
[4]https://www.tensorflow.org/api_docs/python/tf/summary/FileWriter (Accessed: 2018-10-31)

simultaneous processing is more associated with image processing. LSTMs are the typical neural networks architectures for processing sequential data whilst CNNs are more adept to process simultaneous data. Quasi-Recurrent Neural Networks (QRNN), proposed by Bradbury et al. [9] is an architecture which is capable to perform simultaneous processing on sequential data. In simple terms, processing text data as if it was an image. Since most of the processing happens simultaneously in parallel, Bradbury et al. state that QRNN is up to 16 times faster than conventional RNN whilst still achieving state-of-the-art results.

CNNs are one of the most popular deep learning architectures which are mostly applied to tasks which involve image processing, although they were also successfully utilized for sequence processing in particular Zhang et al. [68]. A CNN is constructed similar to a traditional neural network with an input, an output and a number of hidden layers. These hidden layers typically consist of convolution, pooling, fully-connected and normalisation layers. The convolution and pooling layers are used for feature extractions whilst the fully-connected and normalisation layers are used for classification. Convolution layers are the most important unit in a CNN. The architecture of the original CNN, as introduced by LeCun and Bengio [33], alternates between convolutional layers and subsampling layers as illustrated in Figure 3.1. The feature maps of the final subsampling layer are then fed into the actual classifier consisting of a number of fully connected layers. The output layer usually uses softmax activation functions.

Figure 3.1 Architecture of a traditional Convolutional Neural Network [33].



A convolution is a mathematical operation between two functions which results in a third function. In CNNs, the convolution will create two sets of data which are the input and convolution filter (or kernel) to produce a feature map. The pooling layers merge the outputs of neuron clusters at one layer into a single neuron in the next layer. For example, max pooling uses the maximum value from each cluster of neurons at the prior layer, whilst the average pooling uses the average value from each cluster of neurons. Since the fully connected layers accept one dimensional vectors whilst convolution and pooling layers operate with three dimensional vectors, the last pooling layer has to flatten the output to feed the fully connected layers.

Bradbury et al. [9] describe QRNN as a merge of the two architectures which like CNNs allow parallel processing and thus permitting high throughput and scaling for long sequences and similar to RNNs, QRNNs are able to output depending on the order of the sequence. The authors constructed a number of QRNN versions customised to perform several natural language tasks such as document-level sentiment classification, language modelling, and character-level machine translation. This work on dependency parsing has overlap on language modelling and the experiments by Bradbury et al. provided insights and encouraging results to consider QRNN as an alternative to the bi-LSTM architecture. Bidirectional LSTM is the defacto standard architecture for dependency parsing.

Figure 3.2 QRNN architecture as compared to LSTM and CNN architectures [9]

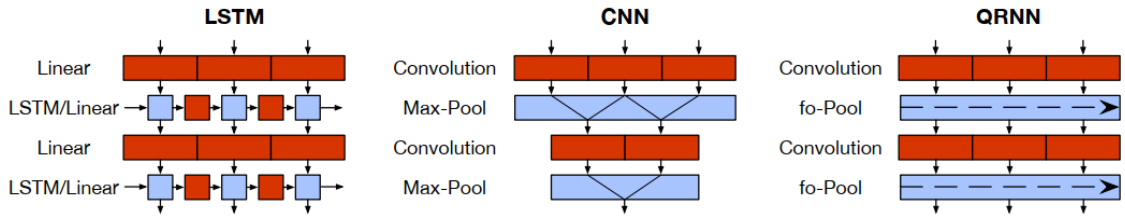


Figure 3.2 illustrates the computation structure of the QRNN when compared with the traditional LSTM and CNN architectures. Red denotes convolutions or matrix operations whilst a continuous block indicates that these computations are parallel. The blue denotes functions that work simultaneously along the channel.

A QRNN layer consists of two types of components which correspond to the convolution and pooling layers in CNNs. The convolutional component behaves similarly to the convolutional layer in CNNs which allows full parallel computation of the sequence dimension. Also like CNNs, the pooling component, does not have any trainable parameters and also allows full parallel computation.

In a QRNN, for an input sequence of n-dimensional vectors, the convolutional component performs convolutions with a set of filters producing a new sequence of m-dimensional vectors at a particular timestep. For the prediction of the upcoming token, the filter set allows computation to be performed on information from future timesteps. This process is known as *masked convolution* which was proposed by Oord et al. [46].

The functions of the pooling component is based on the traditional LSTM unit however the function of the gates is based on *dynamic average pooling* which was defined by Balduzzi and Ghifary [3]. This type of pooling was called 'fo-pooling' by Bradbury et al. [9].

Hence a single QRNN layer can perform input-dependent pooling with subsequent gated convolutional features. Similarly to CNNs, two or more QRNN layers must be stacked to build models which are capable to perform more complex functions.

For the language modelling experiments, Bradbury et al. performed the same experiments by Zaremba et al. [63] and Gal and Ghahramani [24] using the Penn Treebank [34]. In the main result, QRNN obtained a validation of 85.7% whilst [63] experienced a validation of 86.2% with a difference of just a 0.5%. However, QRNN outperformed the runtime performance of the implementation of Zaremba et al. by a maximum of 16 times. These results provide a valid reason to consider QRNN as an alternative to bi-LSTM architecture.

3.3 The Parser

In this section, the methods utilized to construct the parser are described in detail. The feature function is analysed and hence the architecture and actual parsing processes involved are outlined.

This work is based on the research by Kiperwasser and Goldberg [31] which was also used by Dozat et al. [18], placing first in CoNLL 2017 shared task of dependency parsing. Kiperwasser and Goldberg define two parsers one graph-based and the other a transition-based. The parser of this work is specifically based on the graph-based parser and the reference code[5] by Kiperwasser. The contributions which make this parser different is the utilisation of a Quasi-Recurrent Neural Network instead of bi-LSTM and bootstrapped multi-source treebanks.

Kiperwasser and Goldberg state that one of the most critical phases in the design process of the parser is the choice of the feature function. The feature function is a major challenge and is composed of mainly two tasks; which components to consider and which combinations of such components should be included in the function. Typically, state-of-the-art parsers depend on models rather than fully hand-crafted feature functions to focus on core features and the models perform the necessary combinations. In the many works reviewed by Kiperwasser and Goldberg, the feature function was highly complex. For example, the work of Chen and Manning [11] uses 18 different elements as feature function. Hand-crafted feature functions are highly prone to errors and time consuming to create. The feature functions describe the context of a word in an input sentence and the context of that whole input sentence. Such feature functions are based on on templates which when initialised would result in features of the form of:

- word on top of stack is X
- leftmost child if Y
- distance between head and modifier is Z

Typically, a published feature set which demonstrates its efficiency is adopted by other authors and modified to increase performance. An example of such feature set suggested by Kiperwasser and Goldberg is proposed by McDonald et al. [36] for graph-based parsing which consists of 18 templates whilst the actual implementation of the feature function consisted approximately of 100 templates [31].

[5]https://github.com/elikip/bist-parser (Accessed: 2018-10-31)

Kiperwasser and Goldberg propose a simple approach to this problem which is based on bi-LSTMs. As stated before, bi-LSTMs are highly capable of encoding sequences together with their respective contexts. Each word of an input sentence is encoded by its respective bi-LSTM. A small set of these encodings, are concatenated and hence used as a feature function to be passed through a non-linear scoring function.

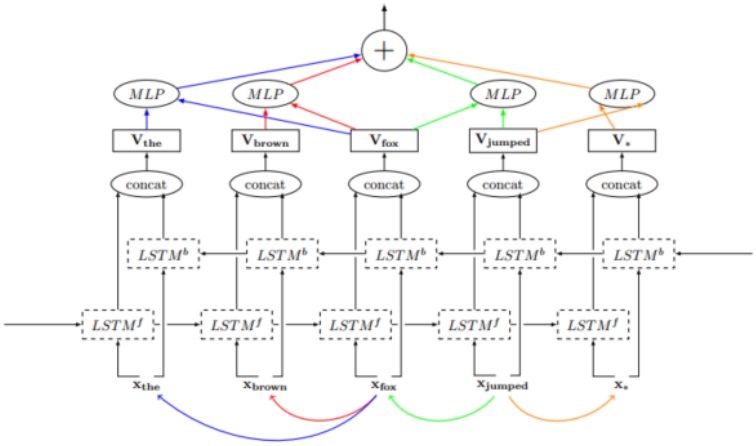
For an input sentence s composed of n-words with a sequence of $w_1, ..., w_n$ and corresponding POS tags $t_1, ..., t_n$, two embedding vectors $e(w_i)$ and $e(t_i)$ for each word and POS tag are constructed. Each of the vectors $e(w_i)$ and $e(t_i)$ are concatenated to form x_i. For all n-words in sentence s, $x_{1...n}$ is the sequence of concatenated vectors. These embeddings are trained with the model and the context is introduced at a later stage. The context is the bi-LSTM vector of the whole $x_{1...n}$ with respect to the word. The feature function is a bi-LSTM encoding of the head word and the modifier word of which the resulting vectors are then scored using a Multi-Layer Perceptron (MLP). Compared to other functions, this proposal by [31] is much more simpler.

The graph-based parser follows the model as proposed by McDonald et al. [36]. For an input sentence s with corresponding sequence of vectors $x_{1...n}$, the highest scoring parse tree y is searched from a space $Y(s)$ of dependency trees. In cite McDonald et al. [36], the parser is arc-factored which breaks down a dependency tree to the sum of the score of its head-modifier arc. Using the break-down of scores, the highest scoring tree can be fetched using Eisner's MST algorithm as proposed by Eisner [22]. Labelled parsing uses this same procedure but after the arc is predicted, the label is predicted. The resulting vectors from the label prediction are fed into a different MLP.

Figure 3.3 illustrates the neural model architecture for the graph-parser during the calculation of the score for the sentence *"the brown fox jumped."*. Each dependency relation is scored using a MLP which is fed the bi-LSTM encoding. The colours of the dependencies match to the colours of the MLP inputs.

These methods were implemented using the reference code for both the bi-LSTM and QRNN architectures. This will give the possibility to generate results for the two architectures

Figure 3.3 Graph-based parser architecture as proposed by Kiperwasser and Goldberg [31]



and compare them for effectiveness and efficiency. The methods and architecture of QRNN were described during the previous section.

3.4 The Bootstrapped Multi-source Treebank

In this section the methodology employed to build the bootstrapped multi-source treebank for the Maltese language is described in detail.

As detailed in Section 2.7, the Maltese annotated treebank, MUDTv1, was privately provided by Čéplö for this work. MUDTv1 currently is composed of just one text file whilst the CoNLL evaluation requires three files. Hence, the first required step was to split MUDTv1 as follows:

1. the training dataset which will contain the annotated sentences to be used for the training of the parser's model. This dataset should contain 60% of all sentences in MUDTv1. The filename should be 'mt-ud-train.conllu'.

2. the development dataset which will contain 20% of the annotated sentences of MUDTv1. These sentences should not be present in the training set. This dataset should be used for performing evaluation of the model. The filename should be 'mt-ud-dev.conllu'.

3. the test dataset which will contain the annotated sentences to perform the prediction based on the model. This file should contain the remaining 20% of MUDTv1. Hence, also for the test dataset, no sentences should be contained in the training dataset. The filename should be 'mt-ud-test.conllu'.

MUDTv1 contains approximately 2000 sentences, with the split training dataset now containing 1200 sentences and the rest split between the development and training datasets.

To perform the split another tool in C# had to be developed. Using a ready-made tools and text editors would risk splitting an annotated sentence and the split would have to be performed sequentially without having any randomization of the annotated sentences. The newly developed tool loads MUDTv1, or any CoNLL format text file and segments each annotated sentence into a list. The list is then shuffled and split into 60%, 20% and the rest 20%, representing the training, development and testing datasets respectively. The shuffling is required to make sure that the sentence originating from a specific source would not be concentrated in one of the datasets. Each list was finally streamed to disk to create the three required files. From this process, the standard Maltese treebank was acquired.

As discussed in Section 2.3, Shi et al. [54] used parallel dataset and projected unlexicalised data to improve performance of low-resourced languages. To execute the same process by Shi et al. would require much more computational resources, time and additional datasets. Eisner [22] states that dependency parsing is primarily a search problem. The aim is to correctly predict the tree of a sentence from the unseen testing dataset using the pre-trained model. The model itself is composed of a variety of trees which were fed into the neural network during the training phase. When a sentence is not parsed correctly, the only non-technical reason is because that tree was not present in the model. From this conclusion it can be stated that increasing the variety of trees in the model should result in better prediction.

Since the whole Maltese treebank was used, the other varied annotated trees must be sourced from other languages. Using the work of Tiedemann and van der Plas [58], the languages which performed best for parsing Maltese can be derived. These languages are listed in Table 2.1 with their respective achieved metrics.

To generate the multi-source treebank, the tool developed to split MUDTv1 was enhanced. The updated tool is capable of receiving a list of CoNLL formatted files. Each file is iterated and parsed into an individual list which is hence shuffled. This is the same procedure as it was performed to split MUTV1. From each list, 1200 sentences are chosen and finally all sentences are merged with the Maltese training dataset. The value of 1200 sentences was chosen to match the Maltese training set and keep the treebank balanced amongst all languages. The final result is a training dataset of Maltese, English, Spanish and Italian languages, each with 1200 sentences. The development and test datasets were not modified.

This whole process can be easily performed on other low-resourced languages. The difficult part is to determine those languages which have syntactic overlap with the target language. This process could be performed for the Maltese language because of the previous work by Tiedemann and van der Plas [58].

3.5 CoNLL 2017 Evaluation standard

The evaluation of the parser will be performed according the CoNLL standards. To perform the evaluation, the CoNLL 2017 evaluation script[6] was used. This ensures that this work follows a defined standard by an institution and the results can be compared with those achieved during CoNLL 2017.

For an evaluation to take place, the parser must output a valid CoNLL-U format file which is then compared to the gold standard. The gold standard for the Maltese for this work is the test dataset (mt-ud-test.conllu). During the training phase, at the end of each epoch an evaluation is performed using the development dataset (mt-ud-dev.conllu). The evaluation is important because the progress of the model can be monitored. The metrics should steadily increase over each epoch. For the prediction phase, the parser's output is compared to the test dataset (mt-ud-test.conllu). During the shared task, the test datasets for all languages will be released only when the models are completed. The whole process is controlled by a

[6]http://universaldependencies.org/conll17/eval.zip (Accessed: 2018-10-31)

system called TIRA[7] which ensures that all participants have a level playing field and none of the participant gain access to the gold standards before the test phase.

A valid output CoNLL-U file must:

1. contain no cycles - a cycle is a cyclic graph where a node (word) is reachable from itself.

2. have one root per sentence - every sentence must have one single root. If just one sentence has multiple roots, the file will be considered as invalid.

3. have the correct number of columns - the output must follow the file format as detailed in Section 2.6

4. not have the wrong indexing of the nodes - wrong indexing occurs when the order of the words of the sentence is modified.

If during evaluation, one of these rules is not adhered to, the file will be considered invalid, the processing halts and metrics are set to zero.

The first process to be performed by the script is to align the output from the parser to the gold standard. The alignment is performed in a series of steps. A multi-word token is a word which is composed by two or more than word tokens and hence it occupies more than one word line. If there are no multi-words in the parser's output, the token sequences should share the same underlying text without considering spaces. The token sequences are the offset boundaries of each token. This process is illustrated in Table 3.2.

Table 3.2 Sample sentence from UD Maltese treebank with offset boundaries.

Qaltlu	li	mhux	vera	li	x-	xmajjar
0-5	7-8	10-13	15-18	20-21	23-24	26-32
kollha	għandhom	memorji	koroh	.		
34-39	41-48	50-56	58-62	63		

[7]http://www.tira.io/tasks/conll/#universal-dependency-learning (Accessed: 2018-10-31)

If multi-words are present, the multi-words from the parser's output must be aligned to the individual words of the gold standard. In this case, the alignment is performed using the Longest Common Subsequence (LCS) algorithm. In LCS, the longest subsequence common to all sequences must be found. This is different from the longest common substring problem. Subsequences are not required to occupy consecutive positions with the original sequences like the substrings.

After the alignment is complete and successful, the parser's output and the gold standards are *attached* and hence the script can compute the metrics. The three metrics used are:

1. Labelled Attachment Score is the standard evaluation metric for dependency parsing. LAS represents the percentage of words that are assigned both the correct head and dependency label. To calculate LAS, only the Universal Dependency Labels are considered, which should be located in column 4 as per CoNLL-U format and illustrated in Table 2.4. Language specific dependency labels are ignored.

2. Unlabelled Attachment Score is the percentage of words that are assigned only the correct head.

3. Weighted Labelled Attachment Score is similar to LAS but the dependency labels are assigned a weight. The weights file is included with the CoNLLU 2017 evaluation script. This metric should always be lower than LAS.

3.6 Conclusion

This chapter provided an in-depth analysis of the processes and algorithms which were employed in developing the parser, associated datasets and tools. Following the development and implementation of the parser, the next step is to perform a number of experiments to gather results and evaluate the performance of the parser, which will be described in the following chapter.

Chapter 4

Evaluation and Results

In this chapter we describe in detail the full evaluation process conducted with the aim to benchmark the parser. All of the experiments devised are listed and documented to sustain the necessity of each set of experiments. In total, there are twenty five experiments categorized into five distinctive sets, with each set targeting a specific component of the parser. The results are presented after each set of experiments conducted. The experiments were constructed in a way that there is a natural progression of parser's design decisions by varying just one component.

4.1 Evaluation Procedure

The experiments were firstly constructed, ordered and split into five sets. The first set of experiments concern the neural network Optimizer which is one of the basic components of the neural network. The second set of experiments determine if there are any benefits of using an external word embedding. The third, establish the contribution of bootstrapped multi-source treebanks in the evaluation metrics, whilst the fourth set of experiments are based on a different neural network architecture with the aim to achieve a ground truth for the preferred architecture. The last set of experiments is aimed to provide metrics to compare with parsers which participated in CoNLL 2017.

The hardware used for the evaluation process was provided by the University of Malta and is detailed in Table 4.1. A virtual environment was provided on the server configured as in Table 4.2. The experiments were performed consecutively accord to their identification number. For each epoch performed, the model and validation results were stored. At the end of the training phase for each experiment, the prediction process was performed to produce the metrics as detailed in Section 3.3.

Table 4.1 GPU Server

CPU	2 x Intel Xeon E5-2640 with 8 cores (16 cores with hyperthreading)
RAM	64 GB DDR3 1600MHz
	2 GB RAMdisk
GPU	2 x NVIDIA Tesla K20m
Storage	1 TB Harddisk
OS	Ubuntu 14.04.5 LTS

Table 4.2 Software and frameworks

Python environment	Python 3.5
GPU support	nvidia CUDA 8.0
	nvidia drivers for CUDA 8.0
	libcupti-dev
Deeplearning framework	PyTorch 0.3.1 with CUDA 8.0 support
Word embedding implementation	fasttext
	GloVe

It is important to note that the Optimizers provided by PyTorch require a number of hyperparameters. These parameters and their relative default values are different for each Optimizer and are specifically configured for the specific Optimizer. Some of these hyperparameter values were tweaked by performing a small number of experiments to try to minimise the loss without overfitting the neural network. These experiments were not documented because we believe that the task of locating the best hyperparameters is too time consuming and would be out of scope for this work. For this reason, it was hence decided that for all

experiments, the default hyperparameter values provided for the Optimizers by PyTorch were used.

4.2 Evaluation metrics

The metrics used to evaluate the performance of this work are defined by CoNLL. These metrics are calculated by the CoNLL 2017 evaluation script as detailed in Section3.5. The use of CoNLL standard metrics enables comparison to the published CoNLL 2017 submissions. The metrics used for evaluation are:

1. LABELLED ATTACHMENT SCORE (LAS) which is the de-facto standard evaluation metric for dependency parsing. LAS is calculated as the percentage of words that are assigned both the correct head and dependency label.

2. UNLABELLED ATTACHMENT SCORE (UAS) is determined as the percentage of words that are assigned only the correct head.

3. WEIGHTED LABELLED ATTACHMENT SCORE (WEIGHTED LAS) is calculated like LAS but each of the dependency labels are assigned a weight. The weights file is included with the CoNLLU 2017 evaluation script. This metric should always score lower than LAS.

4.3 Experiments

Table 4.3 details all the experiments conducted during this work. The fields outlining the experiments are described as follows:

- ID is a consecutive integer value given to each experiment to be used as identification.
- TREEBANK is the annotated treebank used for the experiment.
- NEURAL ARCHITECTURE is the neural network architecture of the parser for the specific experiment. Valid values are QRNN and bi-LSTM.

- EXTERNAL EMBEDDING is the algorithm of the external word embedding used. When no word embedding was used, this field was populated with 'none'.
- OPTIMIZER is the neural network optimizer used for the specific experiment.

The experiments consist of a training phase of thirty epochs with the aim to acquire a model. At the end of each epoch, a model is created and evaluated using the evaluation treebank. From this evaluation, the three metrics; UAS, LAS and Weighted LAS are determined. Therefore, for each epoch in each experiment, a model is created together with the evaluation metrics. The prediction is always performed on the last model of the experiment against the test treebank. At the end of the experiments, 750 models and 750 evaluation metric sets were acquired resulting in 25 prediction metric sets.

Table 4.3 Experiments

ID	Treebank	Neural Architecture	External Embedding	Optimizer
1	Maltese	QRNN	fasttext	AdaDelta
2	Maltese	QRNN	fasttext	AdaGrad
3	Maltese	QRNN	fasttext	Adam
4	Maltese	QRNN	fasttext	SparseAdam
5	Maltese	QRNN	fasttext	Adamax
6	Maltese	QRNN	fasttext	ASGD
7	Maltese	QRNN	fasttext	SGD
8	Maltese	QRNN	fasttext	Rprop
9	Maltese	QRNN	fasttext	Adam
10	Maltese	QRNN	GloVe	Adam
11	Maltese	QRNN	None	Adam
12	Maltese & Romance	QRNN	None	Adam
13	Maltese, Romance & Arabic	QRNN	None	Adam
14	Maltese, Romance & Hebrew	QRNN	None	Adam
15	Maltese & Romance	QRNN	fasttext	Adam
16	Maltese, Romance & Arabic	QRNN	fasttext	Adam
17	Maltese, Romance & Hebrew	QRNN	fasttext	Adam
18	Maltese	bi-LSTM	fasttext	Adam
19	Maltese	bi-LSTM	None	Adam
20	Maltese & Romance	bi-LSTM	fasttext	Adam
21	Maltese & Romance	bi-LSTM	None	Adam
22	English	QRNN	fasttext	Adam
23	Spanish	QRNN	fasttext	Adam
24	Uyghur	QRNN	fasttext	Adam
25	Kazakh	QRNN	fasttext	Adam

4.4 Neural Network Optimization algorithms evaluation

The experiments detailed in Table 4.4 are intended to determine the best neural network optimizer for the implemented parser. All components of the parser were kept constant except for the optimizer. The best optimizer should give a loss which is closest to zero and the highest evaluation metrics. Tables 4.6 and 4.7 detail the loss of each optimizer during each epoch. The loss of each epoch is hence outlined in Figures 4.6 and 4.7. Table 4.5 illustrates the predicted metrics for each experiment using the specific optimizer.

Table 4.4 Neural Network Optimization algorithms experiments

ID	Treebank	Neural Architecture	External Embedding	Optimizer
1	Maltese	QRNN	None	AdaDelta
2	Maltese	QRNN	None	AdaGrad
3	Maltese	QRNN	None	Adam
4	Maltese	QRNN	None	SparseAdam
5	Maltese	QRNN	None	Adamax
6	Maltese	QRNN	None	ASGD
7	Maltese	QRNN	None	SGD
8	Maltese	QRNN	None	Rprop

Table 4.5 Prediction metrics using different optimizers

ID	Optimizer	Prediction Metrics		
		UAS	LAS	Weighted LAS
1	AdaDelta	78.09	73.03	67.05
2	AdaGrad	73.77	63.01	51.92
3	Adam	79.02	73.90	68.08
4	SparseAdam	74.39	69.46	63.49
5	**Adamax**	**79.14**	**74.16**	**68.49**
6	ASGD	77.98	73.05	67.15
7	SGD	12.49	01.84	01.85
8	Rprop	77.92	72.68	67.10

Table 4.6 Loss of Optimization algorithms during training

Epoch	1. AdaDelta	2. AdaGrad	3. Adam	4. SparseAdam
1	22.14	22.12	19.41	22.14
2	22.13	21.50	10.54	22.14
3	20.32	18.62	7.78	22.14
4	16.34	15.73	6.08	22.14
5	13.63	13.24	4.73	22.14
6	11.68	11.51	3.72	22.14
7	10.41	10.28	2.93	22.14
8	9.33	8.93	2.27	22.14
9	8.46	7.90	1.88	22.13
10	7.77	6.94	1.58	22.11
11	7.07	6.20	1.39	22.11
12	6.53	5.55	1.24	21.88
13	5.95	4.91	0.93	21.66
14	5.51	4.39	0.86	21.26
15	5.07	4.00	0.85	18.31
16	4.71	3.69	0.68	14.82
17	4.31	3.39	0.67	12.20
18	3.95	3.07	0.64	10.46
19	3.60	2.78	0.61	9.01
20	3.32	2.60	0.57	7.91
21	2.98	2.45	0.55	6.85
22	2.76	2.29	0.55	6.00
23	2.52	2.13	0.43	5.29
24	2.33	2.01	0.46	4.72
25	2.12	1.89	0.48	4.14
26	1.97	1.85	0.46	3.56
27	1.79	1.62	0.39	3.04
28	1.61	1.51	0.39	2.67
29	1.51	1.35	0.36	2.35
30	1.33	1.20	0.40	2.12

Table 4.7 Loss of Optimization algorithms during training

Epoch	5. Adamax	6. ASGD	7. SGD	8. RMSProp
1	22.13	22.14	22.11	22.14
2	17.62	22.14	22.14	22.14
3	12.00	22.14	22.13	22.14
4	9.56	22.14	22.13	22.14
5	8.03	22.14	22.14	21.95
6	6.80	21.43	22.13	16.27
7	5.85	15.27	22.13	11.87
8	5.02	11.47	22.13	9.83
9	4.19	9.33	22.14	8.28
10	3.58	7.94	22.14	7.04
11	3.05	6.91	22.14	6.36
12	2.57	5.92	22.13	5.69
13	2.12	5.20	22.14	5.12
14	1.70	4.43	22.14	4.40
15	1.51	4.10	22.14	3.93
16	1.36	3.68	22.13	3.45
17	1.11	3.10	22.14	3.09
18	1.04	2.90	22.14	2.91
19	0.81	2.69	22.13	2.57
20	0.81	2.35	22.14	2.21
21	0.72	2.05	22.14	2.24
22	0.66	2.08	22.14	1.97
23	0.53	1.97	22.14	1.82
24	0.61	1.95	22.14	1.67
25	0.52	1.70	22.14	1.66
26	0.46	1.54	22.14	1.58
27	0.46	1.56	22.14	1.53
28	0.40	1.46	22.14	1.53
29	0.38	1.39	22.14	1.37
30	0.35	1.35	22.14	1.29

4.5 External Word Embeddings evaluation

As documented in Section 2.4.2, there are three main word embedding algorithms; word2vec, fasttext and GloVe. This section reports the experiments using fasttext, GloVe and no external embeddings. Word2vec was not included because of time considerations and most probably this algorithm would not outperform the others because it is the legacy algorithm on which fasttext and GloVe were built upon. The aim of these experiments is to find which word embedding algorithm gives the highest contribution to the evaluation metrics.

Table 4.9 illustrates the metrics which resulted from the three experiments. Figures 4.1, 4.2 and 4.3 show the progression of these experiments at each epoch for the prediction metrics.

Table 4.8 External Word Embeddings experiments

ID	Treebank	Neural Architecture	External Embedding	Optimizer
9	Maltese	QRNN	fasttext	Adam
10	Maltese	QRNN	GloVe	Adam
11	Maltese	QRNN	None	Adam

Table 4.9 Prediction metrics with different external Word Embeddings

ID	External Word Embeddings	Prediction Metrics		
		UAS	LAS	Weighted LAS
9	fasttext	**80.68**	**76.27**	**71.02**
10	GloVe	80.41	75.73	70.45
11	None	80.65	74.92	68.74

Figure 4.1 Evaluation using fasttext external word embeddings ID: 9

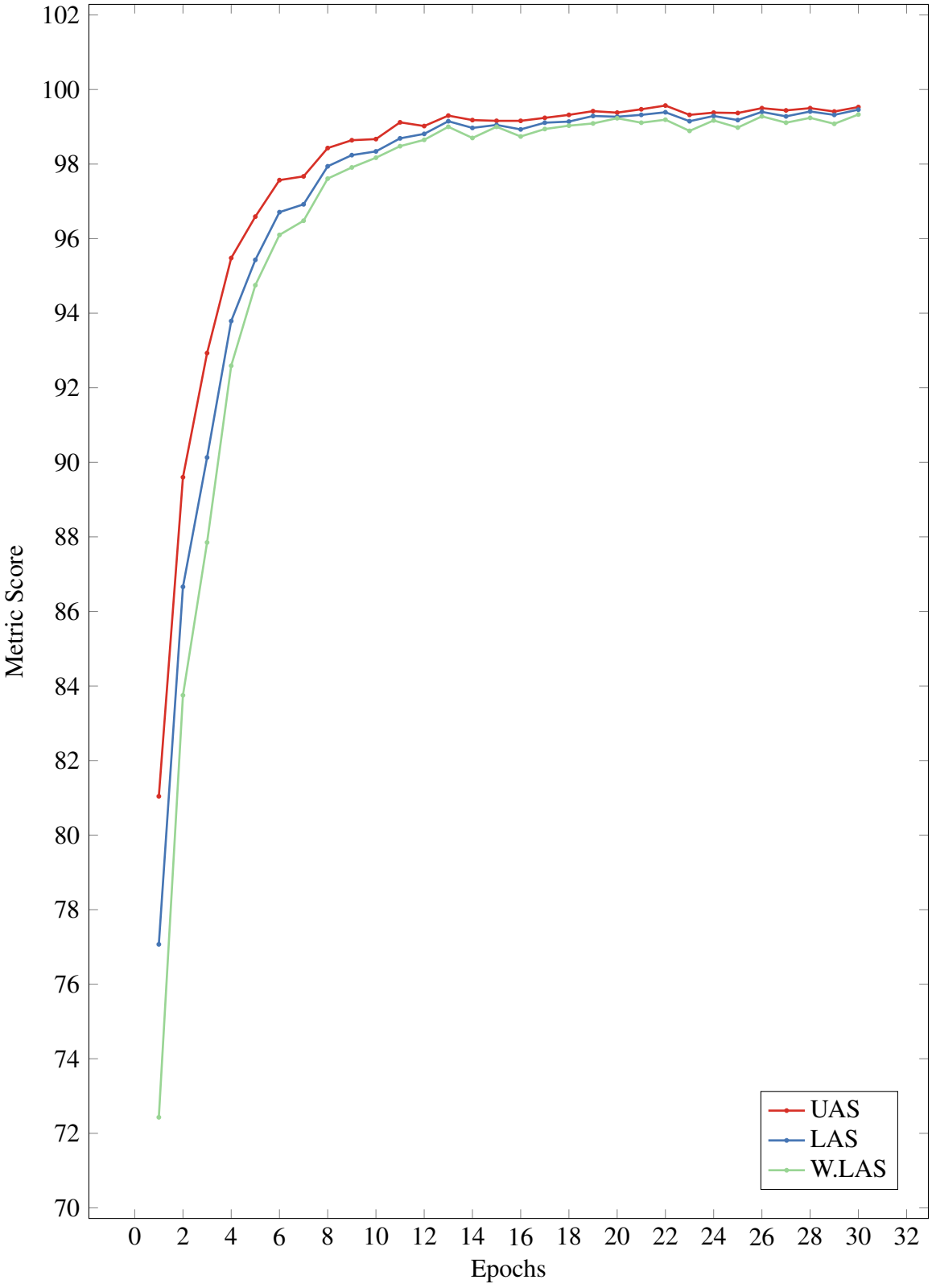


Figure 4.2 Evaluation using GloVe external word embeddings ID: 10

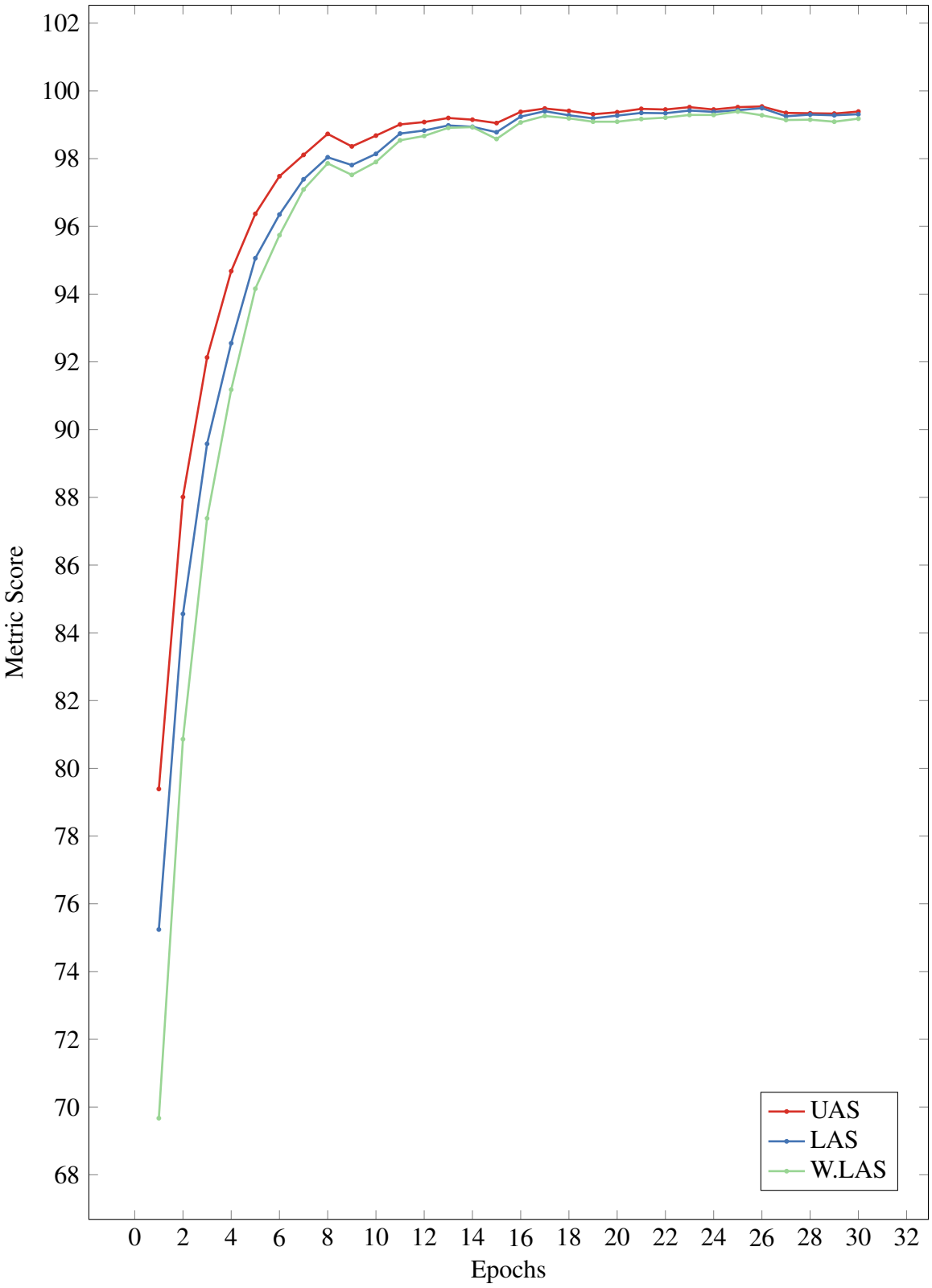
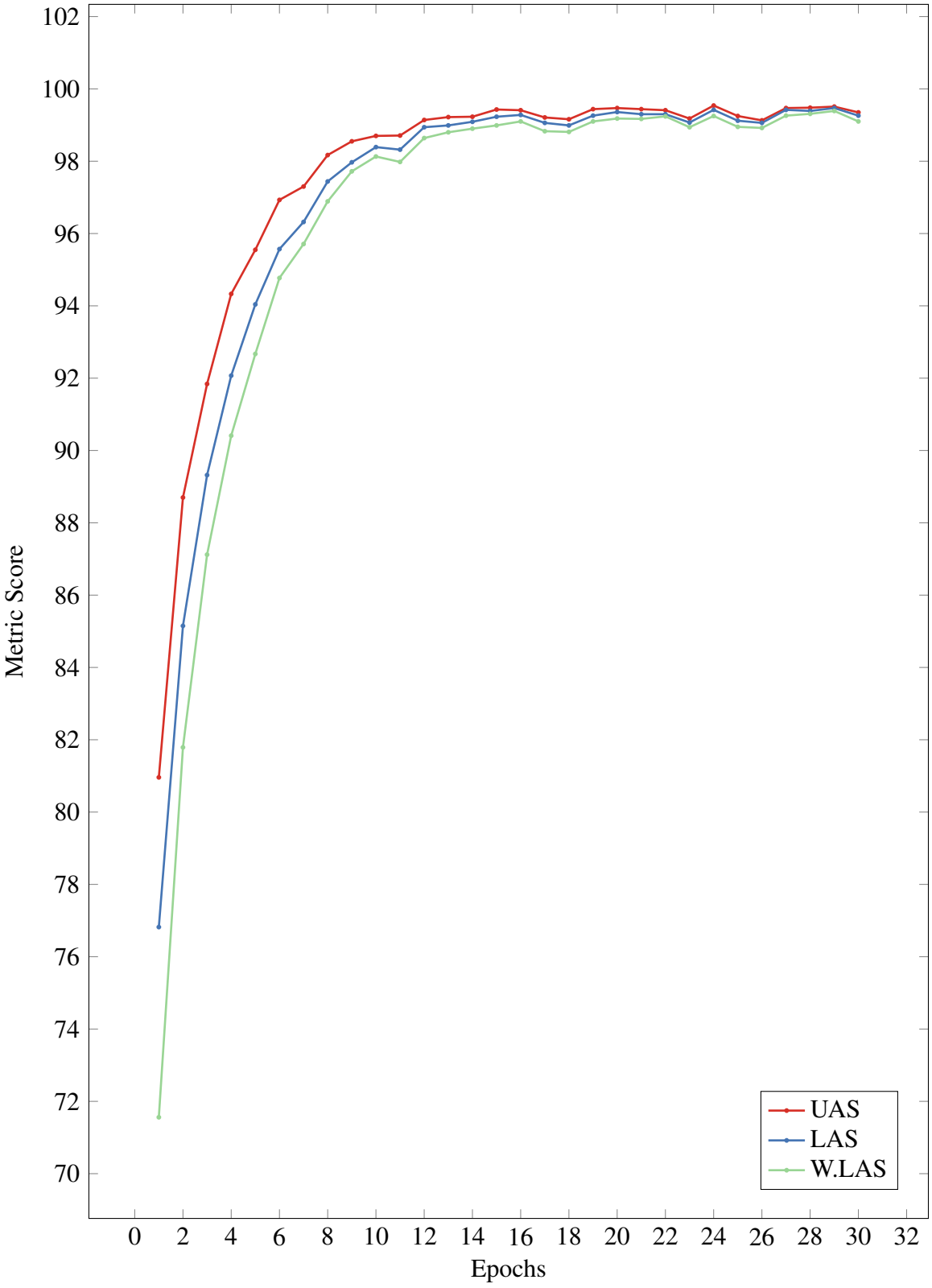


Figure 4.3 Evaluation using no external word embeddings ID: 11



4.6 Bootstrapped Multi-source Treebank evaluation

This section details the experiments performed using the different bootstrapped multi-sourced treebanks. The aim of this phase of experiments is to determine whether bootstrapped multi-sourced treebanks perform better than the single source treebank. Furthermore, some of the experiments in this set were performed without an external word embedding in order to extended the previous set and determine the actual contribution of the word embedding on the evaluation metrics.

Table 4.10 lists these experiments whilst Table 4.11 shows their respective results. Table 4.12 shows the average runtime performance per epoch of each experiment. Figures 4.4, 4.5 and 4.6 show the progression of these experiments at each epoch for each evaluation metric.

Table 4.10 Bootstrapped Multi-source Treebanks experiments

ID	Treebank	Neural Architecture	External Embedding	Optimizer
12	Maltese & Romance	QRNN	None	Adam
13	Maltese, Romance & Arabic	QRNN	None	Adam
14	Maltese, Romance & Hebrew	QRNN	None	Adam
15	Maltese & Romance	QRNN	fasttext	Adam
16	Maltese, Romance & Arabic	QRNN	fasttext	Adam
17	Maltese, Romance & Hebrew	QRNN	fasttext	Adam

Table 4.11 Evaluation metrics using Bootstrapped Multi-source Treebanks

ID	Treebank	External Embedding	Evaluation Metrics		
			UAS	LAS	Weighted LAS
12	Maltese & Romance	None	88.49	85.07	81.85
13	Maltese, Romance & Arabic	None	87.97	84.40	81.07
14	Maltese, Romance & Hebrew	None	87.94	84.21	80.70
15	**Maltese & Romance**	fasttext	**89.77**	**86.33**	**83.17**
16	Maltese, Romance & Arabic	fasttext	88.86	85.45	81.81
17	Maltese, Romance & Hebrew	fasttext	88.61	85.21	82.10

Figure 4.4 UAS during training of models for experiments ID: 12 to 17

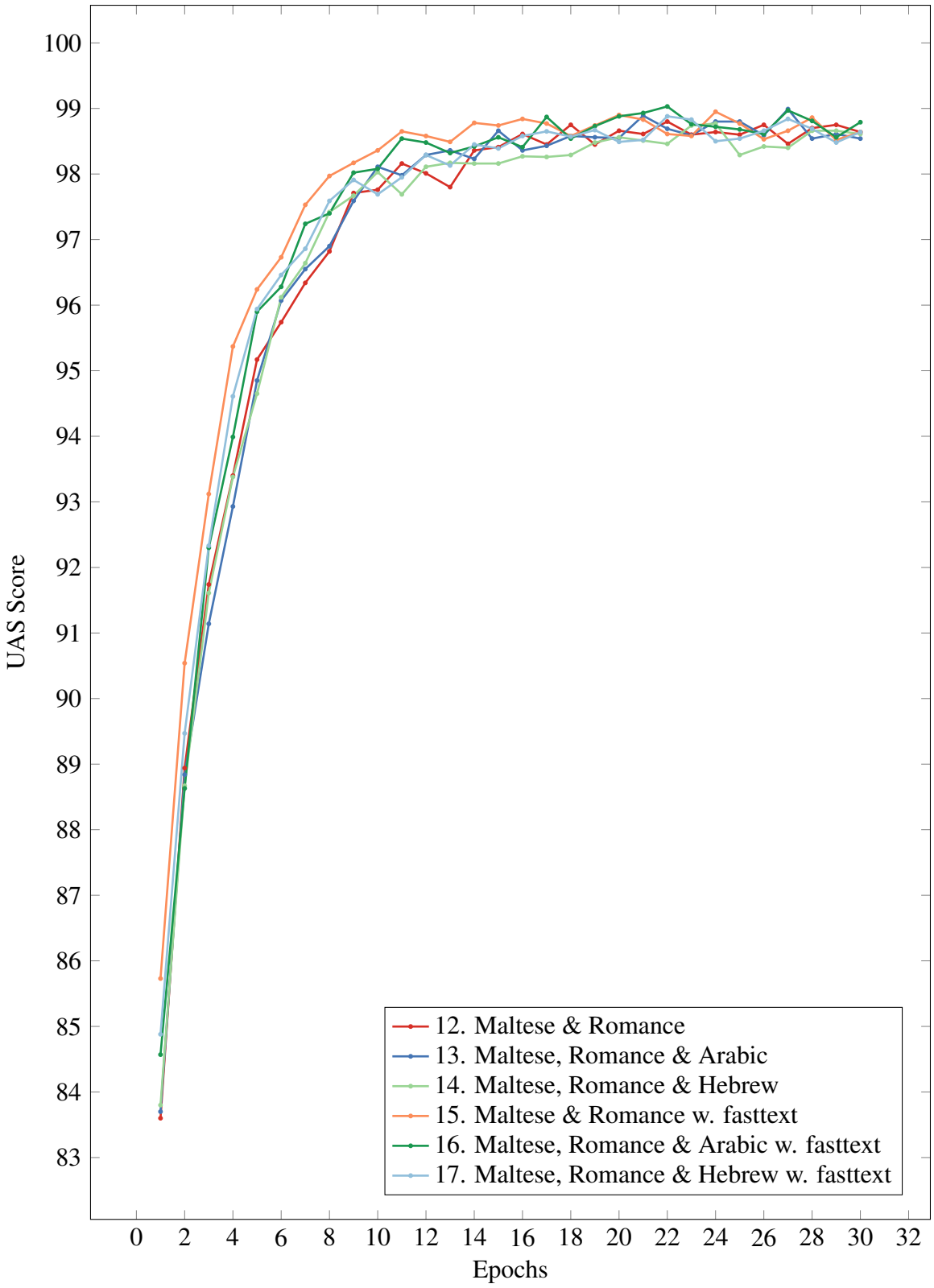


Figure 4.5 LAS during training of models for experiments ID: 12 to 17

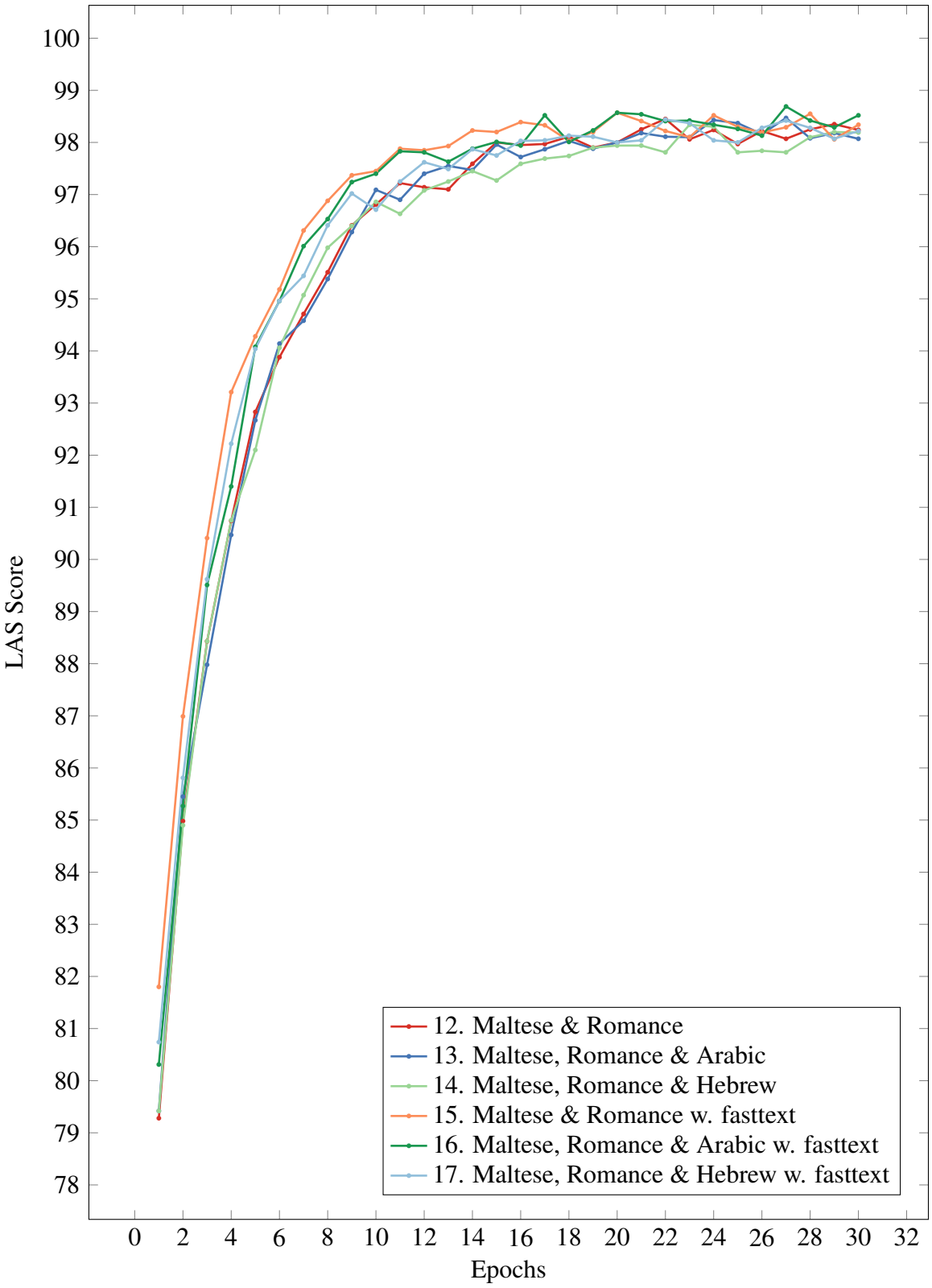


Figure 4.6 Weighted LAS during training of models for experiments ID: 12 to 17

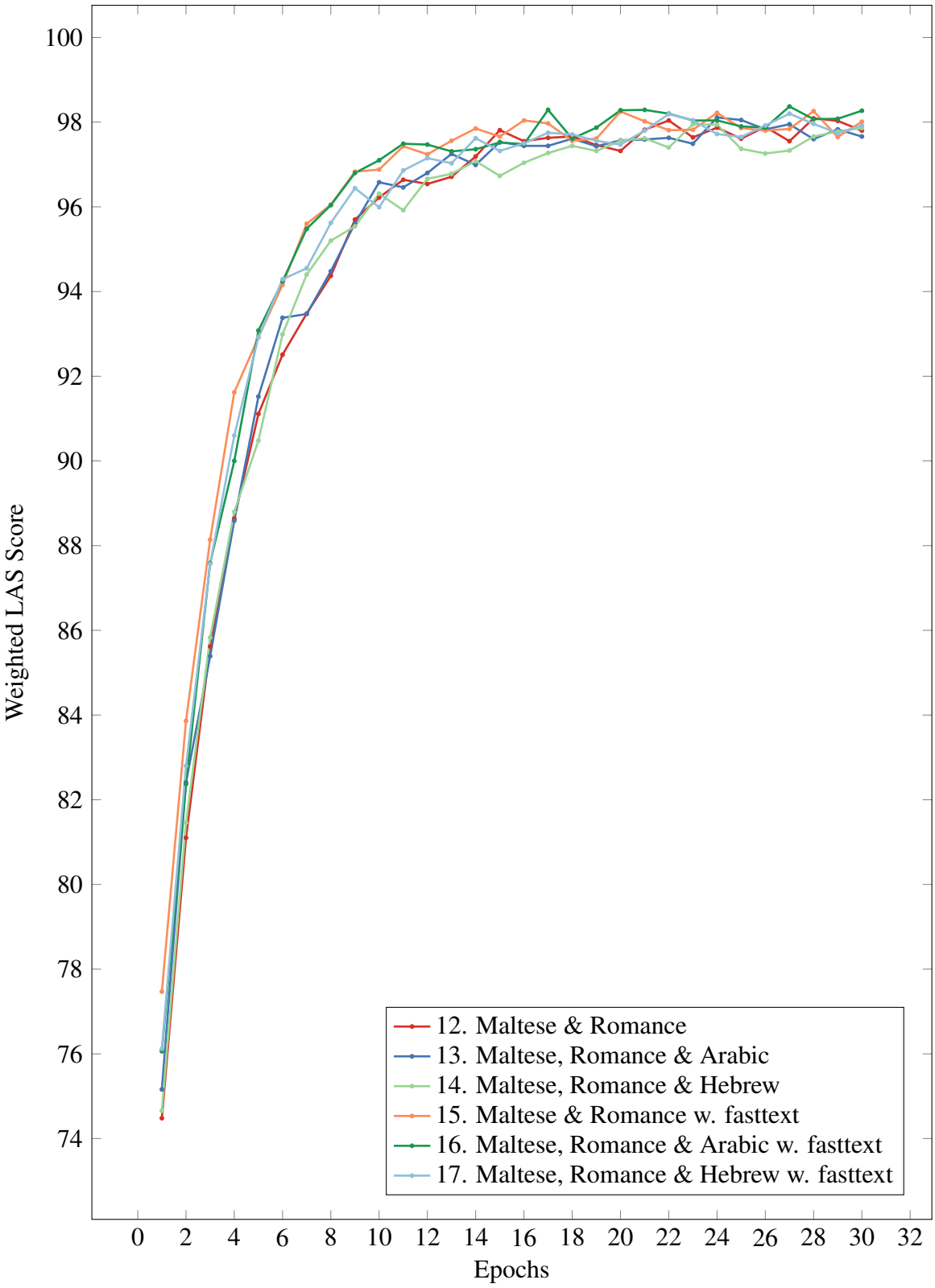


Table 4.12 Performance using Bootstrapped Multi-source Treebank

ID	Treebank	Neural Architecture	Elapsed time per Epoch in minutes
12	Maltese & Romance	QRNN	35
13	Maltese, Romance & Arabic	QRNN	37
14	Maltese, Romance & Hebrew	QRNN	39
15	Maltese & Romance	QRNN	40
16	Maltese, Romance & Arabic	QRNN	38
17	Maltese, Romance & Hebrew	QRNN	40

4.7 Neural Network Architecture evaluation

This section details the experiments performed using bi-LSTM as neural network architecture of the parser. This set of experiments was required in order to compare the performance of QRNN with bi-LSTM. Table 4.13 lists these experiments which are the same as the previous set but using bi-LSTM as neural network. Table 4.14 shows the results obtained from these experiments whilst Table 4.15 shows the average runtime performance per epoch of each experiment.

Table 4.13 Neural Network Architecture experiments

ID	Treebank	Neural Architecture	External Embedding	Optimizer
18	Maltese	bi-LSTM	fasttext	Adam
19	Maltese	bi-LSTM	None	Adam
20	Maltese & Romance	bi-LSTM	fasttext	Adam
21	Maltese & Romance	bi-LSTM	None	Adam

Figure 4.7 Training of model for experiment ID: 20

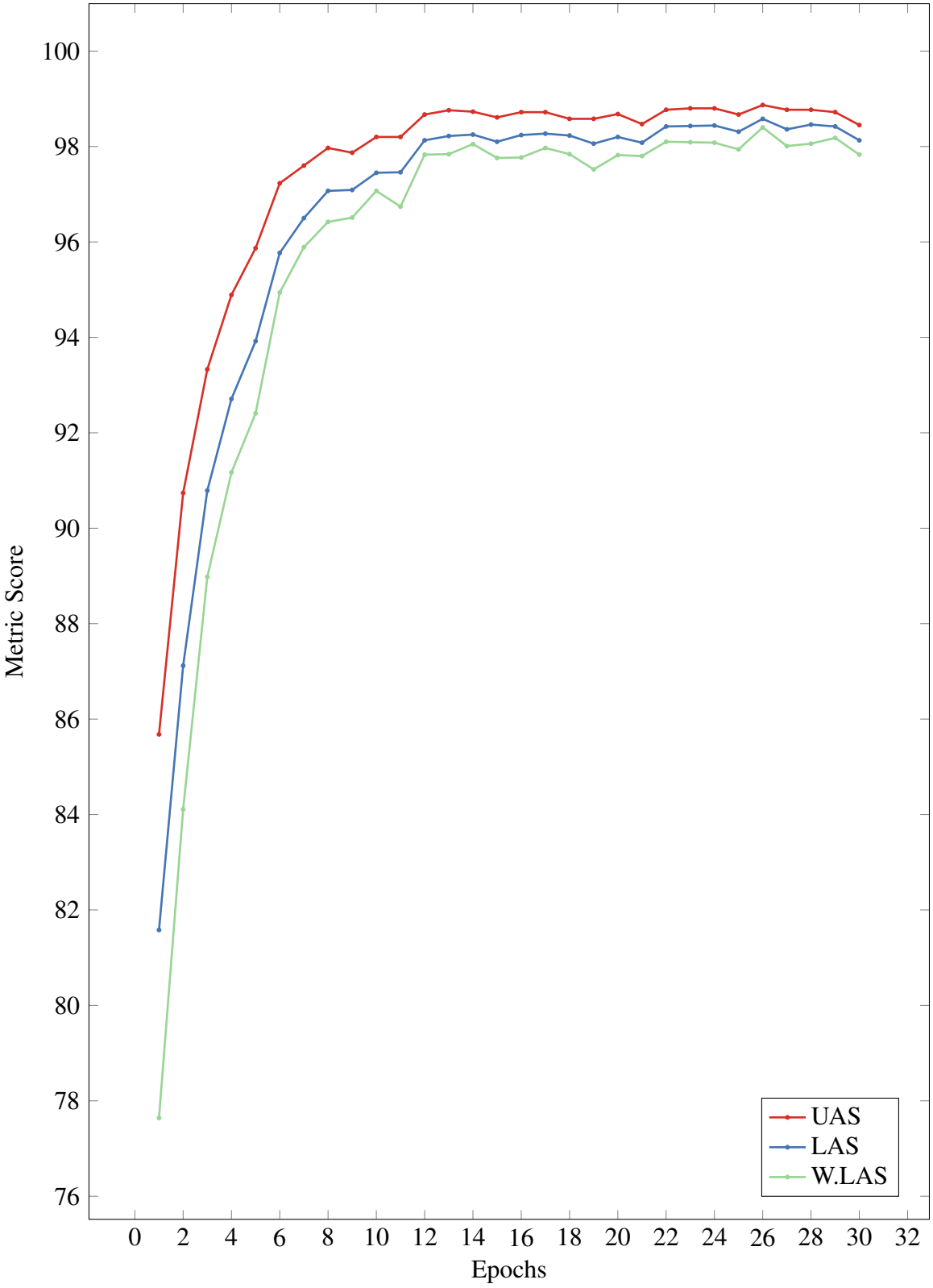


Table 4.14 Evaluation metrics using different Neural Architecture

ID	Treebank	Neural Architecture	Evaluation Metrics		
			UAS	LAS	Weighted LAS
18	Maltese	bi-LSTM	81.05	76.72	71.41
19	Maltese	bi-LSTM	80.98	75.28	68.96
20	**Maltese & Romance**	**bi-LSTM**	**89.89**	**86.51**	**83.38**
21	Maltese & Romance	bi-LSTM	88.70	85.31	82.12

Table 4.15 Runtime performance using bi-LSTM Neural Architecture

ID	Treebank	Neural Architecture	Elapsed time per Epoch in minutes
18	Maltese	bi-LSTM	120
19	Maltese	bi-LSTM	120
20	Maltese & Romance	bi-LSTM	122
21	Maltese & Romance	bi-LSTM	121

4.8 Alternate Languages evaluation

This set of experiments gauges the performance of this parser with other parsing systems which participated in CoNLL 2017. Since the Maltese treebank was not used in the shared task, two well-resourced languages; English and Spanish and two low resourced languages; Uyghur and Kazakh, were used for the experiments.

Table 4.16 Alternate Languages experiments

ID	Treebank	Neural Architecture	External Embedding	Optimizer
22	English	QRNN	fasttext	Adam
23	Spanish	QRNN	fasttext	Adam
24	Uyghur	QRNN	fasttext	Adam
25	Kazakh	QRNN	fasttext	Adam

Table 4.17 Evaluation metrics for Alternate Languages

ID	Alternate Languages	Evaluation Metrics		
		UAS	LAS	Weighted LAS
22	English	87.12	84.58	81.63
23	Spanish	91.99	89.23	84.28
24	Uyghur	73.10	56.13	47.25
25	Kazakh	53.95	34.52	27.22

4.9 Summary of Experiments and Results

This section details the overall experiments and results obtained in this work for easier reference and lookup. Table 4.18 summarises all the experiments categorised in the different sections together with the respective results. The best performing result in each category is highlighted. Shortened descriptions of the treebanks are used in order have a better visual presentation of the table. Detailed descriptions of the treebanks are provided in Table 4.19.

The first set of experiments determined which neural network optimizer performed best for this task. SparseAdam achieved the best results, however, Adam was used for all other experiments because convergence to zero error was achieved at a much earlier stage and the difference in the results when compared to SparseAdam is negligible. The second set of experiments showed that fasttext offers the best word embeddings algorithm for our parser. The aim of the third set of experiments was to determine if multi-sourced treebanks offer better performance over the single-sourced Maltese treebank. The Maltese and Romance languages multi-sourced treebank achieved the best metrics amongst all experiments. The forth set of experiments was performed using a bi-LSTM neural architecture in order to contrast the results achieved by the novel QRNN architecture. The bi-LSTM architecture achieved the better results when compared to the same experiment using QRNN, albeit with a slight difference of 0.12% in performance. The last set of experiments was to determine the performance of our parser when compared to the published CoNLL 2017 submissions.

Table 4.18 Experiments and Results

ID	Treebank	Neural Arch.	External Embedding	Optimizer	Prediction Metrics		
					UAS	LAS	W. LAS
1	Maltese	QRNN	fasttext	AdaDelta	78.09	73.03	67.05
2	Maltese	QRNN	fasttext	AdaGrad	73.77	63.01	51.92
3	Maltese	QRNN	fasttext	Adam	79.02	73.90	68.08
4	Maltese	QRNN	fasttext	SparseAdam	74.39	69.46	63.49
5	**Maltese**	**QRNN**	**fasttext**	**Adamax**	**79.14**	**74.16**	**68.49**
6	Maltese	QRNN	fasttext	ASGD	77.98	73.05	67.15
7	Maltese	QRNN	fasttext	SGD	12.49	01.84	01.85
8	Maltese	QRNN	fasttext	Rprop	77.92	72.68	67.10
9	**Maltese**	**QRNN**	**fasttext**	**Adam**	**80.68**	**76.27**	**71.02**
10	Maltese	QRNN	GloVe	Adam	80.41	75.73	70.45
11	Maltese	QRNN	None	Adam	80.65	74.92	68.74
12	Maltese, R	QRNN	None	Adam	88.49	85.07	81.85
13	Maltese, R, A	QRNN	None	Adam	87.97	84.40	81.07
14	Maltese, R, H	QRNN	None	Adam	87.94	84.21	80.70
15	**Maltese, R**	**QRNN**	**fasttext**	**Adam**	**89.77**	**86.33**	**83.17**
16	Maltese, R, A	QRNN	fasttext	Adam	88.86	85.45	81.81
17	Maltese, R, H	QRNN	fasttext	Adam	88.61	85.21	82.10
18	Maltese	bi-LSTM	fasttext	Adam	81.05	76.72	71.41
19	Maltese	bi-LSTM	None	Adam	80.98	75.28	68.96
20	**Maltese, R**	**bi-LSTM**	**fasttext**	**Adam**	**89.89**	**86.51**	**83.38**
21	Maltese, R	bi-LSTM	None	Adam	88.70	85.31	82.12
22	English	QRNN	fasttext	Adam	87.12	84.58	81.63
23	Spanish	QRNN	fasttext	Adam	91.99	89.23	84.28
24	Uyghur	QRNN	fasttext	Adam	73.10	56.13	47.25
25	Kazakh	QRNN	fasttext	Adam	53.95	34.52	27.22

Table 4.19 Treebank reference

Reference	Treebank
Maltese, R	Maltese & Romance
Maltese, R, A	Maltese, Romance & Arabic
Maltese, R, H	Maltese, Romance & Hebrew

4.10 Conclusion

In this chapter we presented the evaluation procedure which was utilised to evaluate the parser. We reviewed and described all the experiments which were conducted and the results of each experiment were duly reported. In the next chapter the results will be compared and contrasted and a discussion will take placed with the aim to critically appraise this work.

Chapter 5

Discussion

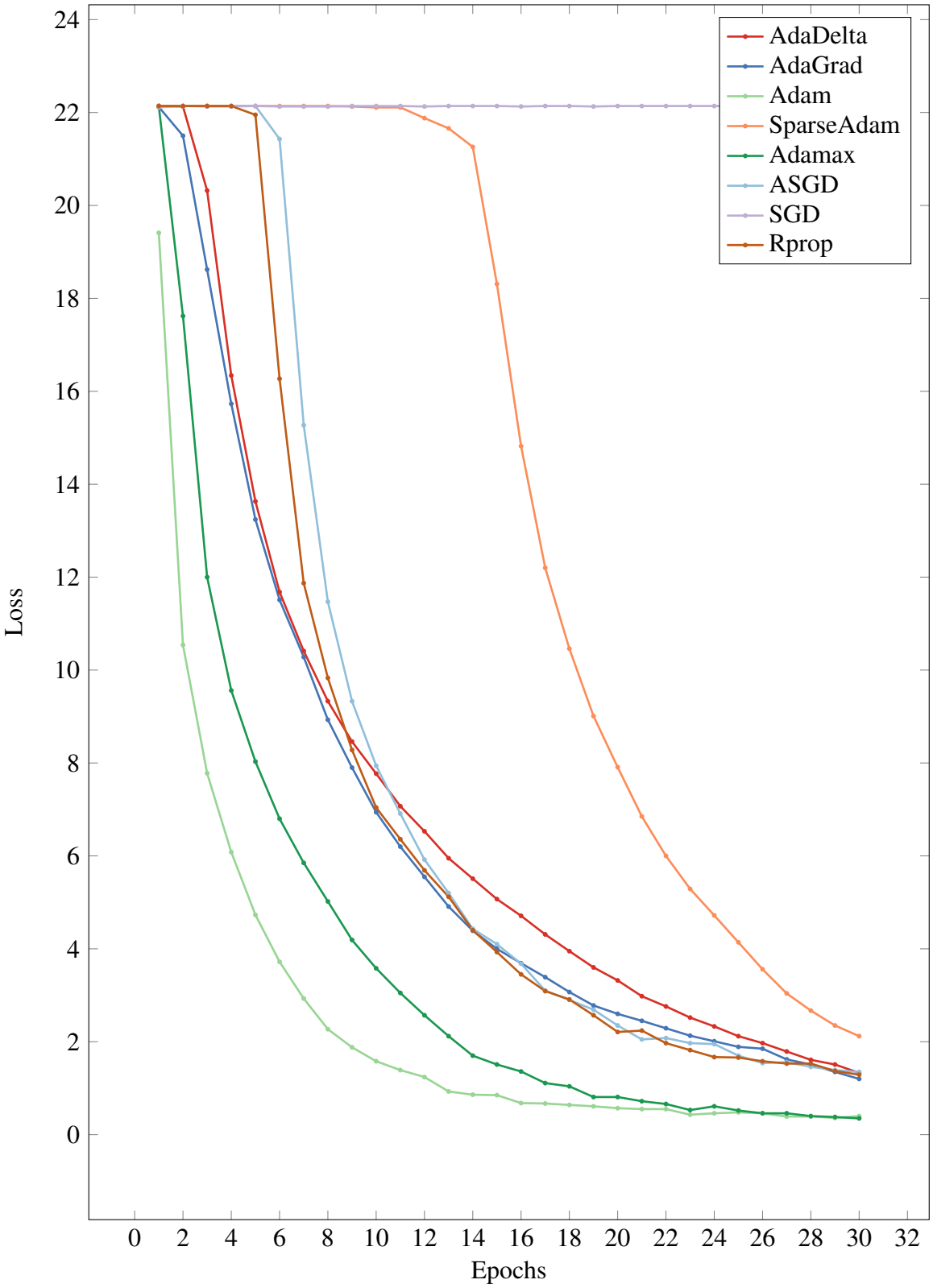
Following the experiments and acquired results, in this chapter we compare and contrast these results and critically discuss their relevance and contribution of this work. We will determine if the aims and objectives of this dissertation were achieved and highlight any limitations of the implemented parser.

5.1 Neural Network Optimization algorithms

All of the optimizers used for this work form part of the PyTorch framework. The hyper-parameters of all optimization algorithms were kept with their default configured values as supplied by PyTorch. As discussed in Section 2.4.1, the aim of a neural network optimization algorithm is to minimise the error function effectively and efficiently.

As demonstrated in Figure 5.1, all of the algorithms converge except for SGD. At epoch thirty, the most effective optimizer is Adamax with a loss of 0.40% and the least effective is SparseAdam with 2.12%. The difference between Adamax and Adam, the second most effective optimizer, is of 0.05%. Since, the difference is very low, the efficiency of the optimizer had to be considered.

Figure 5.1 Optimizer performance through training phase for thirty epochs



A threshold of 1.00% was considered where the graphs will start converging to a nearly straight line approaching zero. The Adam optimizer hits this threshold at epoch 13 whilst Adamax at epoch 19. This is a considerable difference when the computing and runtime resources are taken into account. It can be determined that Adam is the most efficient and ideal optimizer for this work. The fact that Adam converges more efficiently will be discussed in future work as detailed in Section 6.2.

Figure 5.2 compares the predicted metrics from the use of different optimizers. These results do not exclude the possibility that the other optimizers perform well with other treebanks. Adam also performed well during the last batch of experiments where the English, Spanish, Uyghur and Kazakh treebanks were used. Due to the limited time and resources, experiments for the other optimizers were not performed using these treebanks.

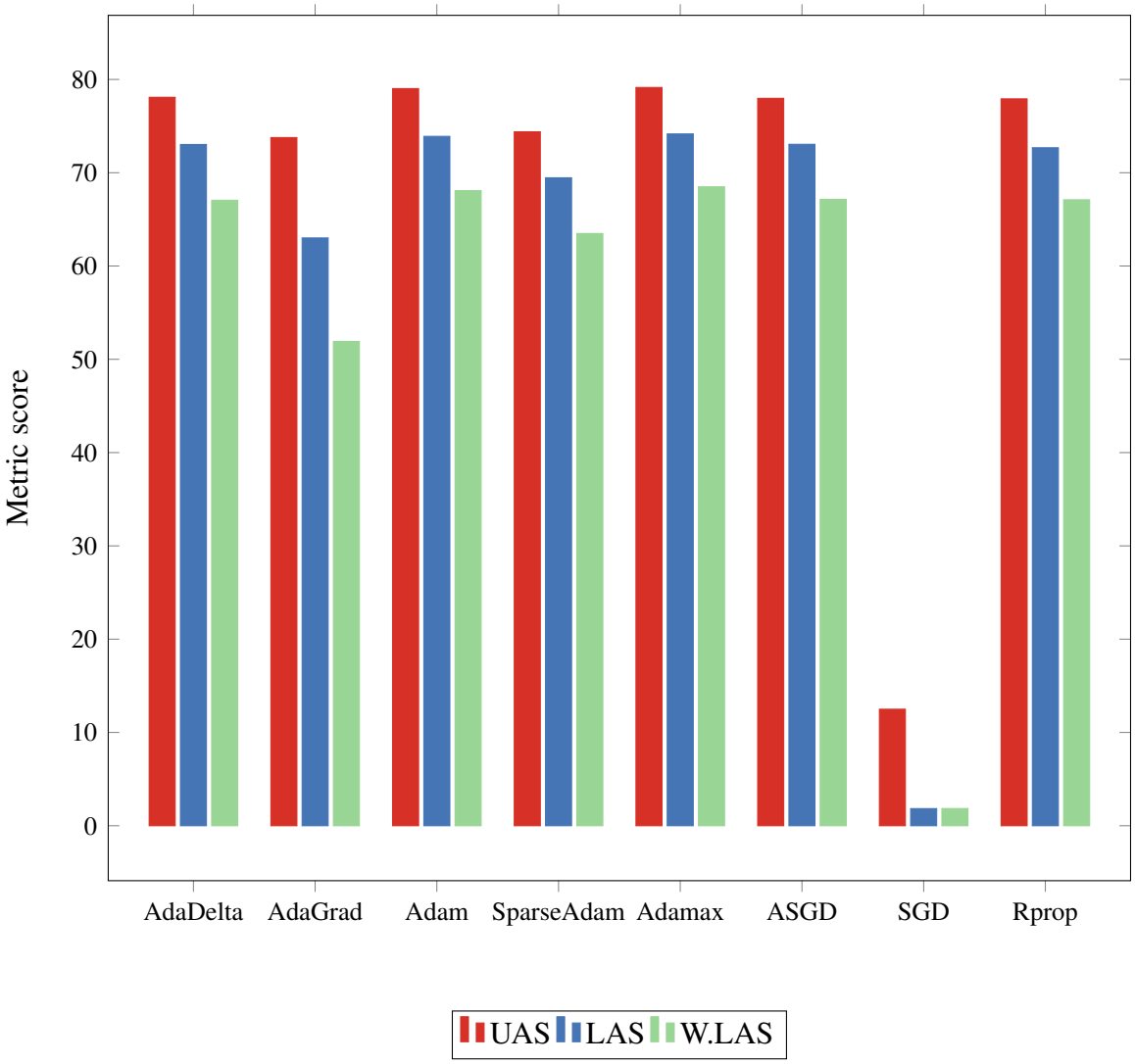
We can conclude that Adam performed particularly well for the implemented parser using a number of different treebanks. We cannot exclude that other optimizers outperform Adam when using treebanks other than the Maltese treebank.

5.2 External Word Embeddings

The word embeddings algorithms used for this work are fasttext and GloVe. The implementations used for both algorithms are from their respective authors in order to make sure that the implementations follow their published research. In both cases, embeddings were created in 100 dimensions based on the same MLRS source text.

Figure 5.3 gives a clear overview of the predicted metrics from the use of the two external word embeddings and without any use. In LAS, the most important metric, fasttext outperforms GloVe by only 0.54%. In UAS, the two algorithms are even closer with fasttext surpassing GloVe by only 0.27%. If these values are considered in isolation, the performance of the algorithms is approximately the same. The results do not give any practical situation when to use a specific word embedding over the other. Such minimal differences can be beneficial during academic research or a shared task like CoNLL's Multilingual Parsing from

Figure 5.2 Prediction metrics using different optimizers

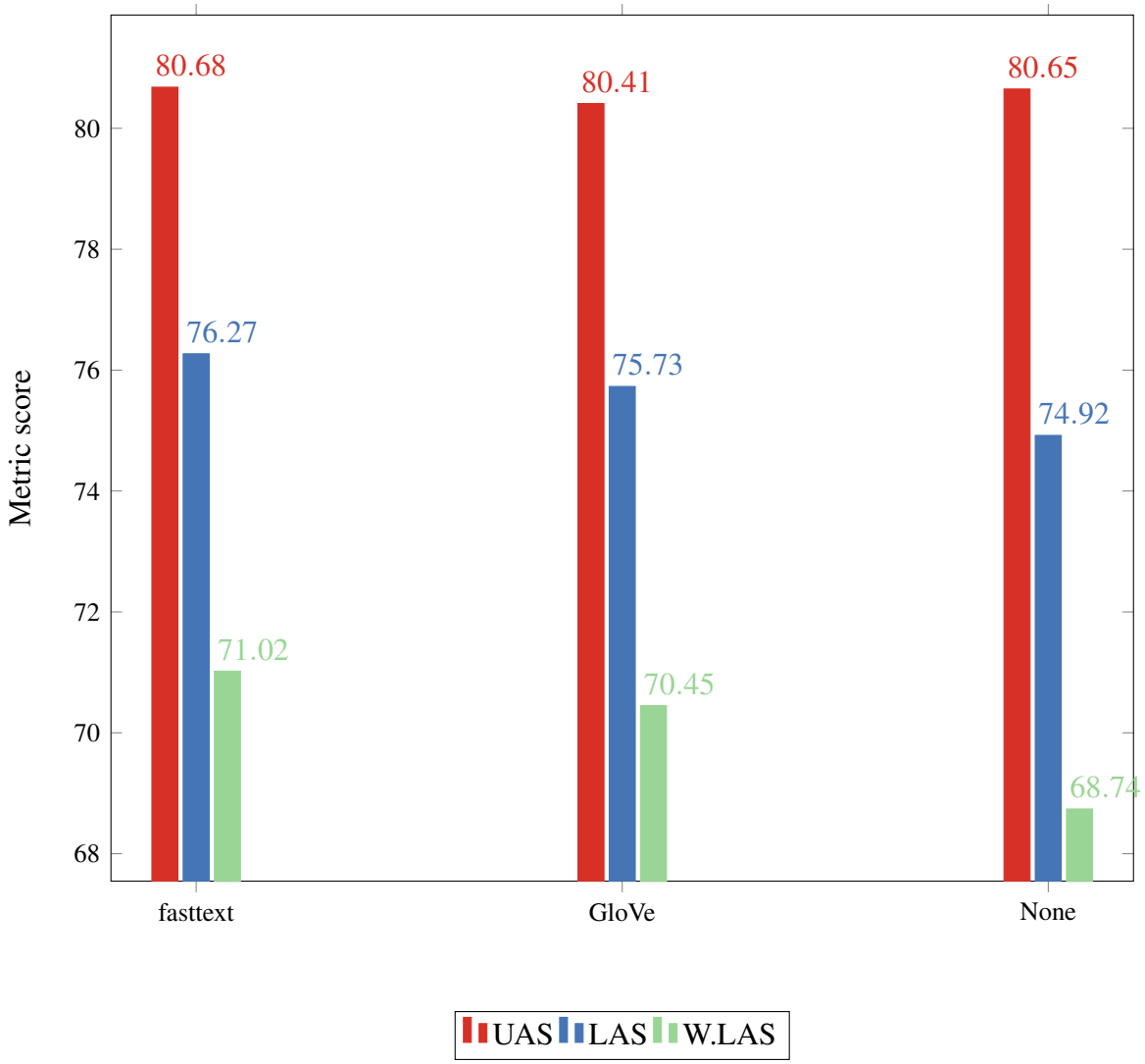


Raw Text to Universal Dependencies, where such small variances can affect the placing of a participant.

The most interesting outcome of these tests is the performance of the parser when no external embeddings were used. In UAS, without using external word embeddings, better scores are achieved rather than using GloVe. Furthermore, when using fasttext word embeddings, it will only result in 0.03% better performance. From these results, it can be stated that for the UAS metric, when using a GloVe embedding, performance actually

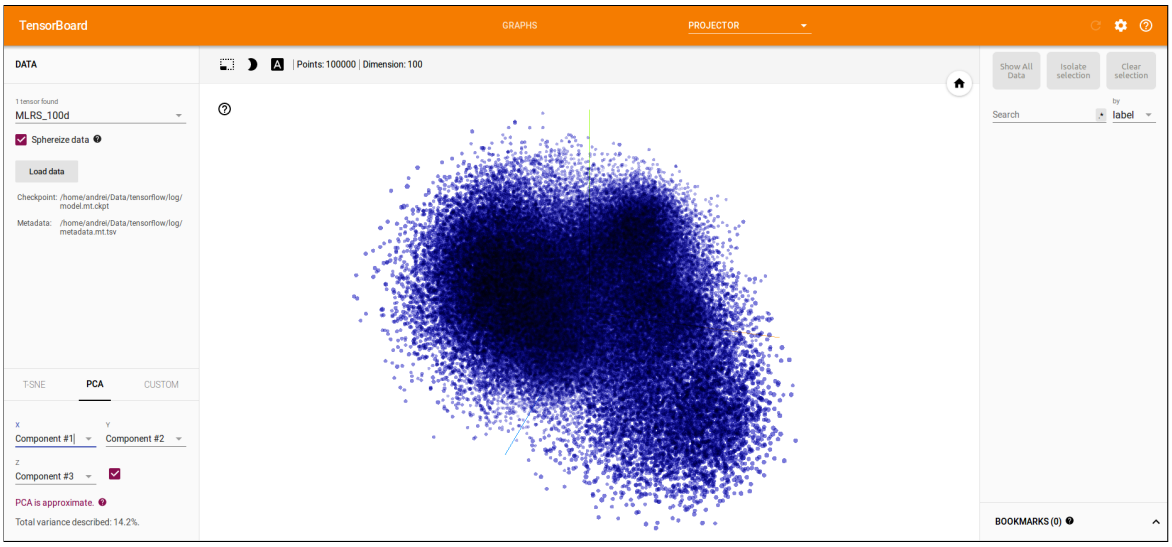
degrades. This phenomenon was observed by Kiperwasser and Goldberg [31], but no reason was given for this occurrence. The authors state, "Interestingly, when adding external word embeddings the accuracy of the graph-based parser degrades. We are not sure why this happens, and leave the exploration of effective semi-supervised parsing with the graph-based model for future work" [31]. This was the only instance when the authors refer to this issue. It also important to note that Dozat and Manning [17] do not discuss this observation and we do not know if it was encountered or not. Dozat and Manning [17] based their work on the graph-parser of Kiperwasser and Goldberg [31]. This phenomenon cannot be further analysed in this work because it requires more expertise and specific research as proposed by Kiperwasser and Goldberg.

Figure 5.3 Prediction metrics using different external word embeddings



As a result of the generated word embeddings, the MLRS source dataset was plotted on three dimensional scatter-plot using Tensorboard, the interactive dashboard of Tensorflow [1]. This result can be observed in Figure 5.4 in a cloud with dense and less dense points indicating the proximity of the Maltese words to each other.

Figure 5.4 Word embeddings plotted on three dimensional scatter-plot



5.3 Bootstrapped Multi-source Treebank

Using bootstrapped multi-source treebanks, better performance was obtained in all experiments over the single-source, Maltese language treebank. One of the main resources of this study is the work of Tiedemann and van der Plas [58] which revealed the languages which have the greatest influence on the Maltese language for the dependency parsing task. Private communication with Čéplö suggests that Arabic and Hebrew should also favourably impact the metrics.

Figure 5.5 demonstrates that all the prediction metrics attained from the bootstrapped multi-source treebanks outperform the single-source Maltese treebank. The most important metrics of this work are those of experiment with ID 15 where the Maltese and Romance treebank was used together with fasttext external word embeddings. Comparing these metrics with experiment ID 9, the multi-source treebanks results increase performance of 9.09% for LAS, 10.06% for UAS and 12.15% from Weighted LAS. This comparison is demonstrated in Figure 5.2.

These results were also compared to those obtained by Tiedemann and van der Plas [58] during two different experiments in order to compare the best predicted metrics. This parser

significantly surpasses the results obtained by Tiedemann and van der Plas as demonstrated in Table 5.3.

Another significant result is that the use of Arabic or Hebrew did not improve performance but actually lowered the metrics, albeit by a small percentage. This is very interesting since according to Čéplö the metrics should have actually improve over using the Maltese and Romance treebank. In the work by Tiedemann and van der Plas [58], this fact was also observed. In their last batch of experiments, the experiment which made use of all European languages performed less than that which made use only of the Romance languages. Tiedemann and van der Plas state that "This suggests that adding lexical information without contextual disambiguation provides only little help but coverage issues may also be good reason for the failure of this approach" [58]. Čéplö and van der Plas were provided with these results for feedback. As a reason Čéplö suggested that typology is to blame for degraded results. The Arabic of Arabic UD treebank is Modern Standard, so basically the same grammar as that of Sibawayh from circa 750 AD and thus it is a completely different language typologically. Hebrew may share a few similarities with Maltese, but it has been argued that in its syntax, it's more Slavic and Germanic than Semitic. Sibawayh is a famous grammarian of the Arabic language and is credited with writing the first grammar of the language. Tiedemann and van der Plas [58] confirmed that during their work they did encounter problems with Arabic and they had to leave the language out of further experiments. At the time of their work, van der Plas also contacted other linguists who confirmed that the word order of the Arabic UD treebank is very different for modern standard Arabic and Maltese.

It also important to keep in context how the model performed during training. As already stated, for every experiment, the model attained during the last epoch was used for prediction. Referring to Figure 4.5, the best performing model at the end of the last epoch is actually achieved during experiment ID 16, when the parser is used with the Maltese, Romance and Arabic treebank. Interestingly, this model did not outperform the model achieved when using Maltese and Romance treebank for the prediction metrics. We are nearly certain that there is a correlation with the previous finding. The most plausible reason is that the addition of

the Arabic treebank to the Maltese and Romance multi-source treebank did not affect the performance of the parser since the evaluation and predicted metrics of the two experiments are very close.

We can determine that multi-source treebanks performed exceptionally better when comparing to the work of Tiedemann and van der Plas [58]. Considering only the most important metric, LAS, an increase of 10.06% is a significant improvement over the single-source Maltese only treebank. There are a number of questions which this set of experiments pose in relation with the use of Arabic and the performance of certain models. Analysing the possibilities which can offer valid answers is out of scope of this project.

Table 5.1 Experiments reference for Figure 5.5

ID	Treebank	Neural Architecture	External Embedding
9	Maltese	QRNN	fasttext
12	Maltese & Romance	QRNN	None
13	Maltese, Romance & Arabic	QRNN	None
14	Maltese, Romance & Hebrew	QRNN	None
15	**Maltese & Romance**	**QRNN**	**fasttext**
16	Maltese, Romance & Arabic	QRNN	fasttext
17	Maltese, Romance & Hebrew	QRNN	fasttext

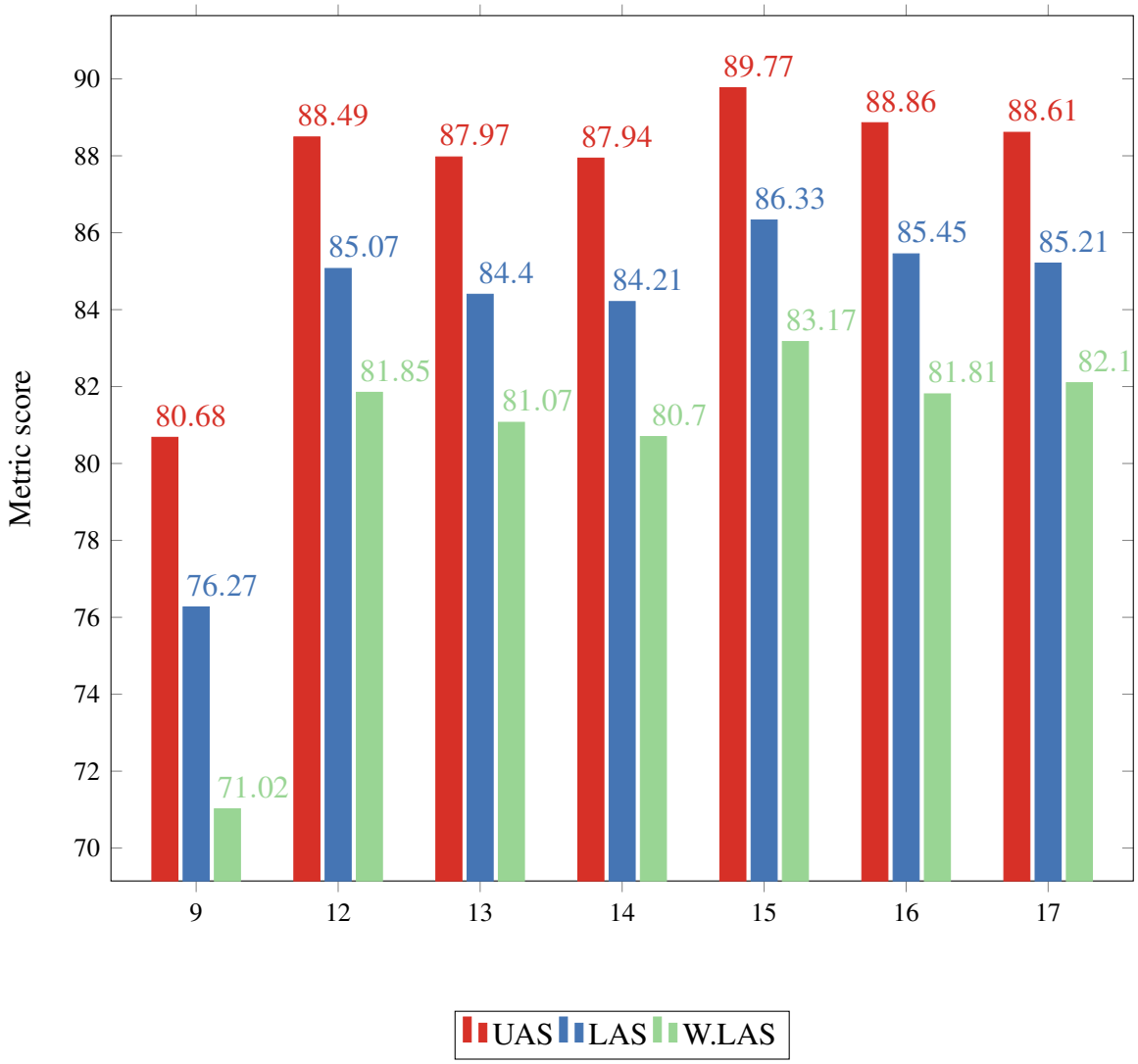
Table 5.2 Comparing results from single-source and multi-source treebanks

ID	Treebank	Neural Architecture	External Embedding	Prediction Metrics		
				UAS	LAS	Weighted LAS
9	Maltese	QRNN	fasttext	80.68	76.27	71.02
15	Maltese & Romance	QRNN	fasttext	89.77	86.33	83.17
15	Performance			+09.09	+10.06	+12.15

Table 5.3 Comparing results from multi-source treebanks to Tiedemann and van der Plas [58]

ID	Author	Prediction Metrics		
		UAS	LAS	Weighted LAS
15	Zammit	89.77	86.33	83.17
	Tiedemann and van der Plas	71.80	63.03	
15	Performance	+17.97	+23.30	

Figure 5.5 Prediction metrics using single-source and multi-source treebanks



5.4 Neural Network Architecture

One of the most important aspects of the work is the use of QRNN as the neural network architecture of the parser. QRNN is a novel architecture which was not ever used for the task of dependency parsing. On the other hand, bi-LSTM is the traditional neural architecture for all tasks involving NLP and is the natural choice for dependency parsing. Several published works confirm that bi-LSTM performs consistently and reliably.

Table 5.4 compares the two best performing experiments for QRNN and bi-LSTM. In all metrics, bi-LSTM performed better with a margin of 0.18% for LAS. Although in these experiments, bi-LSTM is superior by a very small degree, it is important to note that bi-LSTM is bi-directional and hence more context is given to the input sentence. Currently, the only available implementation for QRNN available[1] is unidirectional. It is not possible to predict the increase of performance of the parser with the use of a bi-directional QRNN. However, given that currently the margin difference is minimal, a bi-directional QRNN should outperform bi-LSTM.

According to Bradbury et al. [9], one of the most important contributions of QRNN is the runtime performance. Comparing the runtime performance results as demonstrated in Figure 5.5, QRNN executes three times faster than bi-LSTM.

We can determine that QRNN is a viable alternative to bi-LSTM. The runtime performance of QRNN greatly outperforms bi-LSTM. As regarding to prediction metrics, QRNN is at par to bi-LSTM and the release of a bi-directional QRNN as promised by Bradbury et al. should outperform bi-LSTM.

[1] https://github.com/salesforce/pytorch-qrnn (Accessed: 2018-10-31)

Table 5.4 Prediction metrics using QRNN and bi-LSTM neural architectures

ID	Treebank	Neural Architecture	External Embedding	Prediction Metrics		
				UAS	LAS	Weighted LAS
20	Maltese & Romance	bi-LSTM	fasttext	89.89	86.51	83.38
15	Maltese & Romance	QRNN	fasttext	89.77	86.33	83.17
	Performance			+00.12	+00.18	+00.21

Table 5.5 Runtime performance of QRNN and bi-LSTM neural architectures

ID	Treebank	Neural Architecture	External Embedding	Elapsed time per Epoch in minutes
20	Maltese & Romance	bi-LSTM	fasttext	122
15	Maltese & Romance	QRNN	fasttext	40

5.5 Alternate Languages

The objective of this set of experiments is to compare the predicted metrics from this parser to metrics from the proceedings of CoNLL 2017 [67] and two participants; Dozat et al. [18] and Shi et al. [54]. Two well-resourced languages; English and Spanish and two low-resourced Uyghur and Kazakh were used for the experiments. This exercise should gauge how well the parser performs when compared to other parsers. Dozat et al. [18] placed first overall and achieved the best scores for the well-resourced languages. Shi et al. [54] placed second overall and achieved the best scores for the low-resourced languages. In the proceedings, the Weighted LAS was not reported and hence during comparison this metric was omitted.

Tables 5.6 to 5.9 compares the predicted metrics for the four languages. For the English language, this parser surpasses Dozat et al. [18] by approximately 2.35% on each metric. The results for the Spanish language are similarly positive with approximately 2.25% additional performance.

The performance of the parser on the two low-resourced treebanks is superior. For the Uyghur language, this work registered approximately 12.50% performance improvement

across all three metrics when compared to Shi et al. [54]. For the Kazakh, the improvement was nearly of 9.00% for each metric when comparing again to Shi et al. [54]. Kazakh is the language with the smallest UD treebank [67]. These observations are illustrated in Figure 5.6.

These comparatives establish that the parser performs substantially well when compared to other parsers which participated in CoNLL 2017. We attribute this performance to the use of PyTorch as the deep learning framework with its Autograd and Dynamic Networks features and the choice of Adam as the neural network optimizer. From our knowledge, PyTorch was not used by any of the top participants whose choice was Tensorflow [1] or Dynet[2]. It is important note that there were a number of participants who were affected by an issue in Dynet which caused to produce suboptimal metrics when the training and prediction machines are different [67]. However, neither Dozat et al. [18] or Shi et al. [54] report this issue. This set of experiments demonstrate that the choices we performed through this work we the most appropriate and offered the best performance.

Table 5.6 Predicted metrics for English Language

ID	Author	Prediction Metrics		
		UAS	LAS	Weighted LAS
18	Zammit	87.12	84.58	81.63
	Dozat et al.	84.74	82.23	78.99
	Performance	+02.38	+02.35	+02.64

Table 5.7 Predicted metrics for Spanish Language

ID	Author	Prediction Metrics		
		UAS	LAS	Weighted LAS
19	Zammit	91.99	89.23	84.28
	Dozat et al.	90.01	87.29	82.08
	Performance	+01.98	+01.94	+02.20

[2]https://github.com/clab/dynet (Accessed: 2018-10-31)

Table 5.8 Predicted metrics for Uyghur Language

ID	Author	Prediction Metrics		
		UAS	LAS	Weighted LAS
20	Zammit	73.10	56.13	47.25
	Shi et al.	60.57	43.51	
	Performance	+12.53	+12.62	

Table 5.9 Predicted metrics for Kazakh Language

ID	Author	Prediction Metrics		
		UAS	LAS	Weighted LAS
21	Zammit	53.95	34.52	27.22
	Shi et al.	44.25	25.29	
	Performance	+09.70	+09.23	

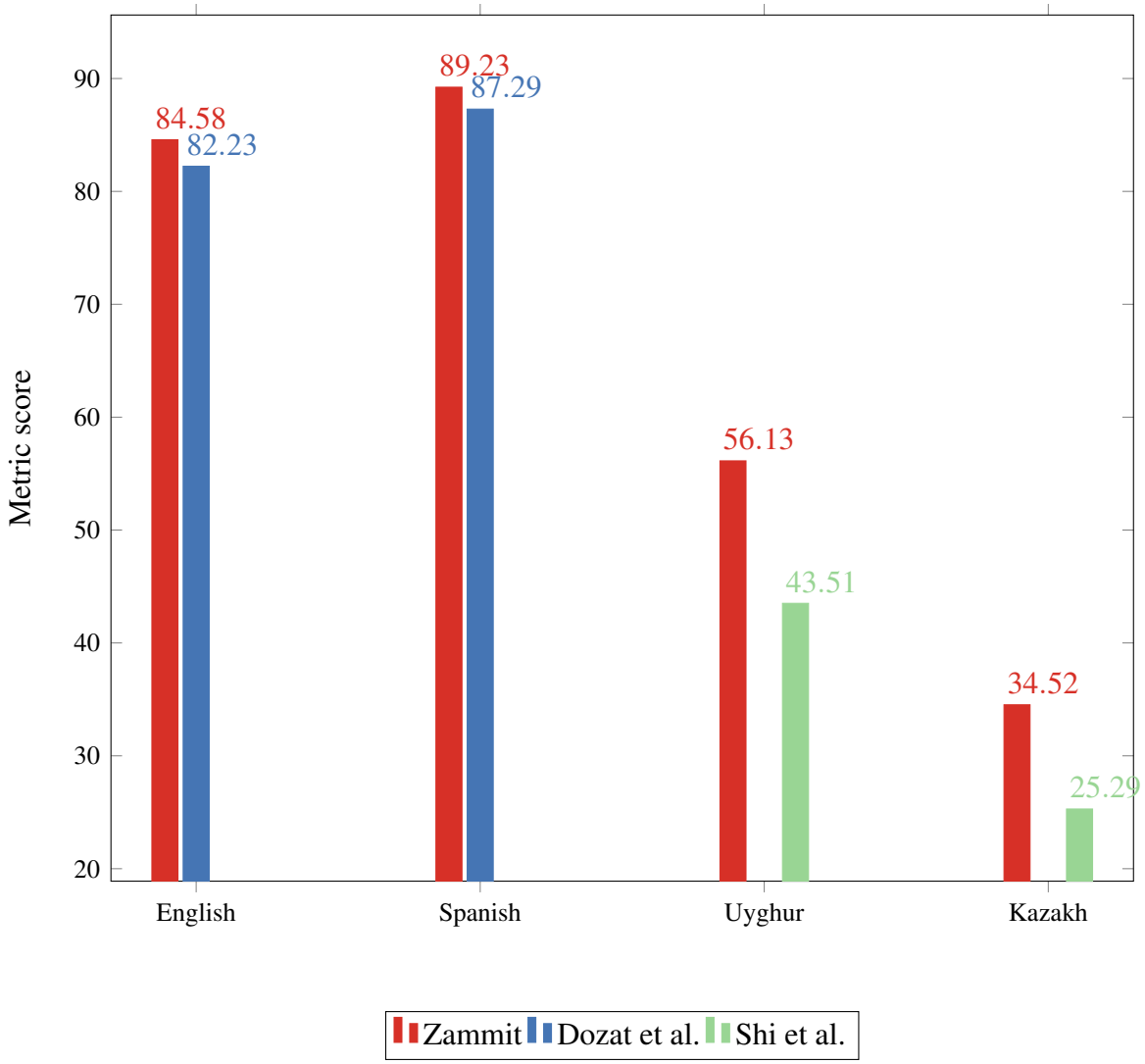
5.6 Contributions

This work investigates the computational parsing of the Maltese language using novel machine learning techniques and the latest Deep Learning technologies. The current state-of-the-art methodologies and architectures were researched and reviewed and proposed a novel approach to dependency parsing with the aim to contribute to the Maltese computational resources and NLP.

This work proposed, designed and implemented the first dependency parser for the Maltese language. To our knowledge, there are no dependency parsers for Maltese and this is one of the main contributions of this work to Maltese, which is computationally low-resourced.

Furthermore, the Maltese language has also attained a corpus of embedded words based on MLRS, which can be used for further research. The word embeddings were mapped and visualised on a three-dimensional scatter-plot using Tensorboard. Further analysis and research can also be performed directly from Tensorboard on the Maltese word embeddings.

Figure 5.6 Prediction for LAS metric using alternate languages



This work also explored the possibility of using bootstrapped multi-source treebank to enhance the performance of the parser. The solution proposed by Shi et al. [54] is a long complicated process which requires several iterations of pre-processing to achieve the necessary datasets. Our approach of merging different UD treebanks is simple to accomplish and uses already available datasets. This technique increased performance by 10% on the LAS metric compared to the single-source treebank.

As confirmed by the results detailed in Table 4.17, the implemented dependency parser is capable to perform on any language based on a trained model. Our parsing algorithm is executed using a Quasi-Recurrent Neural Network, a novel deep neural network architecture. To our knowledge, this is the first time and the only published work that QRNN is used for the task of dependency parsing. The comprehensive results in Tables 5.4 and 5.5 confirm that QRNN is a viable alternative to the traditional bi-LSTM, offering superior runtime performance and at par predictive metrics.

The composition of these techniques and technologies gave our parser a performance which surpassed other parsers which participated in CoNLL 2017 as demonstrated in Tables 5.6 to 5.9. Comparing the results for the Maltese language to that of Tiedemann and van der Plas [58], our parser surpassed their best LAS metric by approximately 23%.

Currently, this work can be applied and used for upcoming CoNLL shared tasks. Furthermore, this work proved that there exist alternatives to the standard neural network architectures. In our case, we proved that QRNN is a possibility. The parser can also be used as a template to explore the possibility of using more simplified data sources, especially for low-resourced languages, which offer a higher metrics performance. The results from the use of bootstrapped multi-source treebanks confirmed the validity of such process.

5.7 Limitations

From a technical perspective, one of the main limitations of this work is that currently there is no bidirectional implementation of QRNN. A bidirectional QRNN would receive more context from the input sentence, and therefore, theoretically, should achieve better results. Of course, it would be essential to run experiments with similar settings to measure and compare the actual performance. According to Bradbury et al. [9] there four modifications required to the QRNN implementation to achieve the bidirectional feature; one related to the CUDA[3] kernel, one to PyTorch and two related to the QRNN implementation. The modification of the CUDA kernel is the most difficult task out of the four modifications[4].

[3] https://docs.nvidia.com/cuda (Accessed: 2018-10-31)
[4] https://github.com/salesforce/pytorch-qrnn (Accessed: 2018-10-31)

Another limitation directly related to the parser is the training phase to construct the model. In the current form, the parser receives a parameter on how many epochs must be performed for the training phase. For each epoch, a model and an evaluation are performed. This process has two main disadvantages; the first is that we cannot know which model, during all epochs, was evaluated with the best metrics if we do not check manually the results of evaluations performed. The second disadvantage is closely related to the first; the training phase will continue until all epochs terminate. Currently, we cannot deduct if the best model was achieved and hence concluded the training phase successfully.

For all of the experiments, we performed thirty epochs and used the model created during the last epoch. As it can be observed in Figure 4.5 this choice was not always the best option. Furthermore, several epochs were performed when there was no need, wasting time and resources.

The final limitation of this work is the discovery of the languages which should compose the bootstrapped multi-source treebank. For the Maltese multi-source treebank, the work of Tiedemann and van der Plas [58] was used by observing which languages achieved the best metrics. If this process is to be used on another language other than Maltese, there should be a similar study beforehand.

5.8 Conclusion

In this chapter we compared, contrasted and discussed the results acquired from the experiments. The findings and limitations were determined with the relevant reasons given. The contributions of this work to the computational resources of the Maltese language and NLP were emphasised, based the critical discussion of the experiments' results. In the next chapter we summarily review the contributions, conclude our work and propose future improvements.

Chapter 6

Conclusion

6.1 Achieved Aims and Objectives

In conclusion, it can be stated that all of the aims and objectives set for this dissertation were met within the scope of this project. The literature was reviewed with an emphasis on CoNLL 2017 and Deep Learning technologies in order to comprehend the latest techniques. The aim was to construct an intelligible set of experiments to decide which technologies and process should compose the parser. The results from the experiments were evaluated and critically discussed. Furthermore, these results were compared to published work and hence the conclusions were stated.

One of the major objectives is to increase the computational resources for the Maltese language. The MLRS was rebuilt as text source from which word embeddings were generated using fasttext. These word embeddings were hence mapped and plotted on a three-dimensional scatter plot using Tensorboard. Tensorboard will enable further analysis of the Maltese vocabulary and additional research can be performed on the word embedding corpus. From the knowledge gained during the background work and literature review, a dependency parser was built. According to our knowledge, this is the first parser for the Maltese language.

Another objective is to contribute to NLP and we achieved this by demonstrating that QRNN is a viable alternative to the standard bi-LSTM. The parsing architecture is based

on QRNN and we attained constant at par prediction metrics when compared to bi-LSTM and three times better runtime performance. We also proved that bootstrapped multi-source treebanks enhance metrics performance over single-source treebanks.

Comparing results for the English language to Dozat et al. [18], the first placed participant of the shared task at CoNLL 2017, our parser accomplished superior metrics with an UAS of 87% and a LAS of 84%. For the Maltese language, our parser outperformed the work of Tiedemann and van der Plas [58] by approximately 23% for an UAS of 90% and a LAS of 86%.

6.2 Future work

There are various opportunities for future work to overcome the limitations and extend this work. This section describes some of these opportunities which vary in degree of difficulty.

A bidirectional QRNN will surely offer better performance and should surpass the metrics results of the bi-LSTM architecture. This parser can switch between the various neural networks' architectures available from PyTorch such as bi-LSTM and third parties which are compatible with the framework. Bradbury et al. [9] adhered to the PyTorch standards when implementing QRNN. We believe that the upcoming release of QRNN, with the bidirectional feature, should remain compatible and hence it should be considerably easy to implement in our parser. This will also give the opportunity to perform comparative experiments between unidirectional and bidirectional QRNN, thus giving clear indications of the effectiveness of the bidirectional feature.

One of the discussed limitations is the fact that the training phase concludes when all epochs finish. The current implementation does not have any indication in which epoch the optimal model has been achieved. One potential solution is to keep a history of the evaluation metrics through all epochs and a pointer to the epoch which resulted with the best evaluation metrics. The history earlier than the highest scoring epoch can be discarded. A threshold is required to determine how many consecutive epochs have to result in lower evaluation metrics in order to stop the training process. For example; if epoch ten has the

best evaluation metrics and hence the best model, models from epochs one to nine can be discarded. Assuming that the threshold is set to three, if epochs eleven to thirteen result in lower evaluation metrics, the testing phase can be successfully concluded. The model from epoch ten will be used for prediction. This solution should overcome the some of the limitations discussed in the previous chapter. Although such solution does require some work, the implementation should not be arduous.

One of the main contributions of this work is the use of bootstrapped multi-source treebanks. We could use the correct language UD treebanks because Tiedemann and van der Plas performed a study from which we could acquire those languages which performed best for Maltese dependency parsing [58]. There are a considerable number of other low-resourced languages such as Uyghur and Kazakh for which the same process can be applied. The most difficult task is discovering those languages which should be used for the multi-source treebank. This work can be extended to enable the parser to detect which UD trees of the target language are close in terms of LAS to other UD trees in other languages. This technique is known as Projection and was used by Tiedemann and van der Plas [58]. Implementing such solution would require time and effort which are beyond our possibilities and this technique does not guarantee that the system would attain the desired results as Tiedemann and van der Plas [58] experienced in their work.

6.3 Final remarks

Dependency parsing is a thriving research area within NLP, with a yearly shared task dedicated to its research and development. It is a challenging task and many academics are consistently working in this area both for research and industrial purposes. Its broad spectrum offers a wide range of innovation and novelty in processes and system architectures.

Maltese is the only Semitic language written in Latin script, which is also influenced by Romance languages and is still a computationally low-resourced language but it is positively experiencing a surge in interest.

This work contributes to the Maltese language by provisioning a dependency parser and a corpus of embedded Maltese words. The contribution to dependency parsing is the use of Quasi-Recurrent Neural Networks as the basis of the parsing architecture and the use of bootstrapped multi-source treebanks. This work achieved state-of-the-art results for the Maltese language with Unlabelled Attachment Score (UAS) of 90% and Labelled Attachment Score (LAS) of 86% whilst for the English language an UAS of 87% and a LAS of 84%.

The importance of a dependency parser for any language can only be highlighted as more technological advanced in Human-Computer/Machine communication is achieved. For instance, in speech recognition, it is possible to communicate using a highly-resourced language such as English. However, it cannot be considered as a blanket solution that will work for everyone. It could be that people are not fluent enough to speak such a main language or it could also be that the accent spoken is not understood by the system. The continuous development of computational processing tools for Maltese is essential to ensure that computer systems have the possibilities and facilities to communicate in different languages seamlessly.

Bibliography

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org.

[2] Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., Petrov, S., and Collins, M. (2016). Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452. Association for Computational Linguistics.

[3] Balduzzi, D. and Ghifary, M. (2016). Strongly-typed Recurrent Neural Networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1292–1300. JMLR.org.

[4] Björkelund, A., Falenska, A., Yu, X., and Kuhn, J. (2017). IMS at the CoNLL 2017 UD Shared Task: CRFs and Perceptrons Meet Neural Networks. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 40–51, Vancouver, Canada. Association for Computational Linguistics.

[5] Bohnet, B. (2010). Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd international conference on computational linguistics*, pages 89–97. Association for Computational Linguistics.

[6] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

[7] Borg, C. (2015). *Morphology in the Maltese language: A computational perspective*. PhD thesis, University of Malta.

[8] Borg, C. and Gatt, A. (2014). Crowd-sourcing Evaluation of Automatically Acquired, Morphologically Related Word Groupings. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*.

[9] Bradbury, J., Merity, S., Xiong, C., and Socher, R. (2017). Quasi-Recurrent Neural Networks. In *International Conference on Learning Representations (ICLR 2017)*.

[10] Buchholz, S. and Marsi, E. (2006). CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 149–164. Association for Computational Linguistics.

[11] Chen, D. and Manning, C. (2014). A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750.

[12] Chu, Y.-J. (1965). On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

[13] Collins, M. (2003). Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637.

[14] Das, D. and Petrov, S. (2011). Unsupervised part-of-speech tagging with bilingual graph-based projections. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 600–609. Association for Computational Linguistics.

[15] de Marneffe, M.-C., Dozat, T., Silveira, N., Haverinen, K., Ginter, F., Nivre, J., and Manning, C. D. (2014). Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland.

[16] De Marneffe, M.-C., MacCartney, B., Manning, C. D., et al. (2006). Generating typed dependency parses from phrase structure parses. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'06)*, volume 6, pages 449–454. Genoa Italy.

[17] Dozat, T. and Manning, C. D. (2017). Deep biaffine attention for neural dependency parsing. In *International Conference on Learning Representations (ICLR 2017)*.

[18] Dozat, T., Qi, P., and Manning, C. D. (2017). Stanford's Graph-based Neural Dependency Parser at the CoNLL 2017 Shared Task. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, Vancouver, Canada, August 3-4, 2017*, pages 20–30.

[19] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

[20] Earley, J. (1970). An Efficient Context-Free Parsing Algorithm. *Commun. ACM*, 13:94–102.

[21] Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71(4):233–240.

[22] Eisner, J. M. (1996). Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345. Association for Computational Linguistics.

[23] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

[24] Gal, Y. and Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027.

[25] Gatt, A. and Céplö, S. (2013). Digital corpora and other electronic resources for Maltese. In *Proceedings of Corpus Linguistics 2013*, Lancaster, UK.

[26] Gers, F. A., Schmidhuber, J., and Cummins, F. (1999). Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12:2451–2471.

[27] Hochreiter, S. and Schmidhuber, J. (1997). Long Short-term Memory. *Neural Comput.*, 9(9):1735–1780.

[28] Hwa, R., Resnik, P., Weinberg, A., Cabezas, C., and Kolak, O. (2005). Bootstrapping parsers via syntactic projection across parallel texts. *Natural language engineering*, 11(3):311–325.

[29] Jurafsky, D. (2000). Speech and language processing: An introduction to natural language processing. *Computational linguistics, and speech recognition*.

[30] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

[31] Kiperwasser, E. and Goldberg, Y. (2016). Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

[32] Kong, L., Alberti, C., Andor, D., Bogatyy, I., and Weiss, D. (2017). DRAGNN: A Transition-based Framework for Dynamically Connected Neural Networks. *CoRR*, abs/1703.04474.

[33] LeCun, Y. and Bengio, Y. (1995). Convolutional Networks for Images, Speech, and Time-Series. In Arbib, M. A., editor, *The Handbook of Brain Theory and Neural Networks*, pages 255–257. MIT Press.

[34] Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):313–330.

[35] Maruyama, H. (1990). Structural disambiguation with constraint propagation. In *Proceedings of the 28th annual meeting on Association for Computational Linguistics*, pages 31–38. Association for Computational Linguistics.

[36] McDonald, R., Crammer, K., and Pereira, F. (2005). Online large-margin training of dependency parsers. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 91–98. Association for Computational Linguistics.

[37] McDonald, R. and Nivre, J. (2007). Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

[38] Menzel, W. and Schröder, I. (1998). Decision Procedures for Dependency Parsing Using Graded Constraints. In *Proceedings of ACL'90*, pages 78–87.

[39] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient Estimation of Word Representations in Vector Space. *International Conference on Learning Representations (ICLR) Workshop*.

[40] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

[41] Nivre, J. (2003). An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*.

[42] Nivre, J. (2006). Inductive Dependency Parsing. In *Text, speech and language technology*. Springer.

[43] Nivre, J., de Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajic, J., Manning, C. D., McDonald, R. T., Petrov, S., Pyysalo, S., Silveira, N., et al. (2016). Universal Dependencies v1: A Multilingual Treebank Collection. In *LREC*.

[44] Nivre, J., Hall, J., Kübler, S., McDonald, R., Nilsson, J., Riedel, S., and Yuret, D. (2007a). The conll 2007 shared task on dependency parsing. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

[45] Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S., and Marsi, E. (2007b). MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13:95–135.

[46] Oord, A. V., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel Recurrent Neural Networks. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1747–1756, New York, New York, USA. PMLR.

[47] Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

[48] Čéplö, S. (2018). *Constituent order in Maltese: A quantitative analysis*. PhD thesis, Charles University.

[49] Polyak, B. T. and Juditsky, A. B. (1992). Acceleration of Stochastic Approximation by Averaging. *SIAM J. Control Optim.*, 30(4):838–855.

[50] Rosner, M. and Joachimsen, J. (2012). *Il-Lingwa Maltija Fl-Era Diġitali – The Maltese Language in the Digital Age*. META-NET White Paper Series. Georg Rehm and Hans Uszkoreit (Series Editors). Springer. Available online at http://www.meta-net.eu/whitepapers.

[51] Sagae, K. and Lavie, A. (2006). Parser Combination by Reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, NAACL-Short '06, pages 129–132, Stroudsburg, PA, USA. Association for Computational Linguistics.

[52] Schröder, I., Pop, H. F., Menzel, W., and Foth, K. A. (2001). Learning Grammar Weights Using Genetic Algorithms. In *Recent advances in Natural Language Processing, RANLP-2001*, pages 235–239.

[53] Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

[54] Shi, T., Wu, F. G., Chen, X., and Cheng, Y. (2017). Combining Global Models for Parsing Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 31–39, Vancouver, Canada. Association for Computational Linguistics.

[55] Straka, M., Hajic, J., and Straková, J. (2016). UDPipe: Trainable Pipeline for Processing CoNLL-U Files Performing Tokenization, Morphological Analysis, POS Tagging and Parsing. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France. European Language Resources Association (ELRA).

[56] Tesniére, L. (2015). *Elements of Structural Syntax (English Translation of Tesniere 1966)*. John Benjamins Publishing Company.

[57] Tiedemann, J., Agić, Ž., and Nivre, J. (2014). Treebank translation for cross-lingual parser induction. In *Eighteenth Conference on Computational Natural Language Learning (CoNLL 2014)*.

[58] Tiedemann, J. and van der Plas, L. (2016). Bootstrapping a dependency parser for Maltese - a real-world test case. In *From Semantics to Dialectometry : Festschrift in honor of John Nerbonne*, pages 355–365, Milton Keynes, England. College Publications.

[59] Waegel, D. (2013). A Survey of Bootstrapping Techniques in Natural Language Processing. *Department of Computer Science, University of Delaware, Literature Survey Reports*.

[60] Weiss, D., Alberti, C., Collins, M., and Petrov, S. (2015). Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 323–333. Association for Computational Linguistics.

[61] Yamada, H. and Matsumoto, Y. (2003). Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, volume 3, pages 195–206. Nancy, France.

[62] Younger, D. H. (1967). Recognition and Parsing of Context-Free Languages in Time n^3. *Information and Control*, 10:189–208.

[63] Zaremba, W., Sutskever, I., and Vinyals, O. (2015). Recurrent Neural Network regularization. In *International Conference on Learning Representations (ICLR 2015)*.

[64] Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. *CoRR*, abs/1212.5701.

[65] Zeman, D. (2008). Reusable Tagset Conversion Using Tagset Drivers. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'08)*, volume 2008, pages 28–30.

[66] Zeman, D., Marecek, D., Popel, M., Ramasamy, L., Stepánek, J., Zabokrtskỳ, Z., and Hajic, J. (2012). HamleDT: To parse or not to parse? In *LREC*, pages 2735–2741.

[67] Zeman, D., Popel, M., Straka, M., Hajic, J., Nivre, J., Ginter, F., Luotolahti, J., Pyysalo, S., Petrov, S., Potthast, M., Tyers, F., Badmaeva, E., Gokirmak, M., Nedoluzhko, A., Cinkova, S., Hajic jr., J., Hlavacova, J., Kettnerová, V., Uresova, Z., Kanerva, J., Ojala, S., Missilä, A., Manning, C. D., Schuster, S., Reddy, S., Taji, D., Habash, N., Leung, H., de Marneffe, M.-C., Sanguinetti, M., Simi, M., Kanayama, H., dePaiva, V., Droganova, K., Martínez Alonso, H., Çöltekin, c., Sulubacak, U., Uszkoreit, H., Macketanz, V., Burchardt, A., Harris, K., Marheinecke, K., Rehm, G., Kayadelen, T., Attia, M., Elkahky, A., Yu, Z., Pitler, E., Lertpradit, S., Mandl, M., Kirchner, J., Alcalde, H. F., Strnadová, J., Banerjee, E., Manurung, R., Stella, A., Shimada, A., Kwak, S., Mendonca, G., Lando, T., Nitisaroj, R., and Li, J. (2017). CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19, Vancouver, Canada. Association for Computational Linguistics.

[68] Zhang, X., Zhao, J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657.

Appendix A

Universal Dependencies specifications

Universal Features

The list of Universal features with the corresponding descriptions from the Universal Dependencies online documentation[1].

Abbr	abbreviation
AbsErgDatNumber	number agreement with absolutive/ergative/dative argument
AbsErgDatPerson	person agreement with absolutive/ergative/dative argument
AbsErgDatPolite	politeness agreement with absolutive/ergative/dative argument
AdpType	adposition type
AdvType	adverb type
Animacy	animacy
Aspect	aspect
Case	case
Clusivity	clusivity
ConjType	conjunction type
Definite	definiteness or state
Degree	degree of comparison
Echo	is this an echo word or a reduplicative?

[1] http://universaldependencies.org/u/feat/index.html Last accessed: 2018-10-31

ErgDatGender	gender agreement with ergative/dative argument
Evident	evidentiality
Foreign	is this a foreign word?
Gender	gender
Hyph	hyphenated compound or part of it
Mood	mood
NameType	type of named entity
NounType	noun type
NumForm	numeral form
NumType	numeral type
NumValue	numeric value
Number	number
PartType	particle type
Person	person
Polarity	polarity
Polite	politeness
Poss	possessive
PossGender	possessor's gender
PossNumber	possessor's number
PossPerson	possessor's person
PossedNumber	possessed object's number
Prefix	Word functions as a prefix in a compund construction
PrepCase	case form sensitive to prepositions
PronType	pronominal type
PunctSide	which side of paired punctuation is this?
PunctType	punctuation type
Reflex	reflexive
Style	style or sublanguage to which this word form belongs
Subcat	subcategorization

Tense	tense
Typo	is this a misspelled word?
VerbForm	form of verb or deverbative
VerbType	verb type
Voice	voice

Universal Dependency Relations

The list of Universal Dependency Relations with the corresponding descriptions from the Universal Dependencies online documentation[2].

acl	clausal modifier of noun (adjectival clause)
advcl	adverbial clause modifier
advmod	adverbial modifier
amod	adjectival modifier
appos	appositional modifier
aux	auxiliary
case	case marking
cc	coordinating conjunction
ccomp	clausal complement
clf	classifier
compound	compound
conj	conjunct
cop	copula
csubj	clausal subject
dep	unspecified dependency
det	determiner
discourse	discourse element
dislocated	dislocated elements
expl	expletive
fixed	fixed multiword expression
flat	flat multiword expression
goeswith	goes with
iobj	indirect object
list	list

[2]http://universaldependencies.org/u/dep/index.html Last accessed: 2018-10-31

mark	marker
nmod	nominal modifier
nsubj	nominal subject
nummod	numeric modifier
obj	object
obl	oblique nominal
orphan	orphan
parataxis	parataxis
punct	punctuation
reparandum	overridden disfluency
root	root
vocative	vocative
xcomp	open clausal complement