A Dependency Parser for the Maltese Language using Deep Neural Networks

Andrei Zammit
Department of Artificial Intelligence
University of Malta
andrei.zammit.16@um.edu.mt

Claudia Borġ
Department of Artificial Intelligence
University of Malta
claudia.borg@um.edu.mt

ABSTRACT

Tasks such as information retrieval and sentiment analysis require natural language processing tools at their foundation. Sentence parsing is one of the tasks performed in NLP to analyse the grammatical 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 language and currently there is no parsers for Maltese. This work investigates computational parsing of Maltese by using novel Deep Learning and bootstrapping techniques from multilingual sources, with the aim of contributing to the increase in computational resources for Maltese and also to dependency parsing.

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. To our knowledge, this is the first time that QRNN is applied to the task of dependency parsing. Thus we report on the applicability of this technique to the task of dependency parsing in general.

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. 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 further relations between words.

The computational treatment of Maltese has lagged behind when compared to the development of other major European languages [26]. 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 [18] and further research at the development of a morphological analyser [3, 4]. More recently, the development of a manually annotated set of sentences with their respective dependency parse trees (Maltese UD Treebank, [25]) means that it is now possible to experiment with machine learning techniques and create a dependency parser based on this annotated data. This research is the first of its kind, looking specifically at computational dependency parsing of Maltese by using this newly developed treebank. It will also be the first time that a particular deep learning architecture will be applied to dependency parsing, with the aim to contribute not just to Maltese dependency parsing, but also to the broader field of dependency parsing in general.

2 MOTIVATION

Modern Dependency Grammar theory is attributed to the work of the French linguist Tesniére [30]. 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*, and the other word on which it depends, called the *head* or *modifier*. A sentence will always have a *root* 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 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.

The dependency relations are represented with arrows originating from the HEAD pointing to the DEPENDENT. Each of these arrows is labeled, denoting the type of dependency relation between two words. For example, the noun *Ġanni* is dependent on the verb *tefa'* (threw), 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.

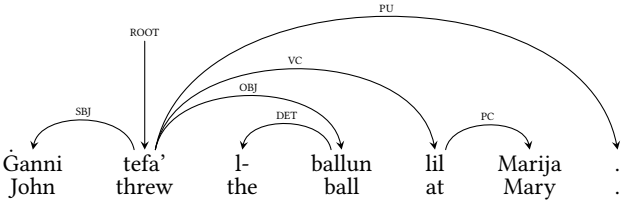

Figure 1: Dependency structure for Maltese and English sentence

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 [25] 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 Universal Dependency project[1]. The Maltese treebank consists of just 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 also explore 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 languages related 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.

3 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 one that could 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 dependency parsing in general. The stated aims were 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] http://universaldependencies.org/introduction.html

4 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 de facto standard framework for dependency parsing — the Universal Dependencies framework and the Maltese treebank which has been released under this framework.

4.1 Traditional methodologies

Traditionally, there are four approaches to dependency parsing: dynamic programming, constraint satisfaction, transition-based and graph-based.

4.1.1 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 set of rules, and then a number of algorithms, such as Cocke-Younger-Kasami (CKY) [35] and Earley's algorithm [14], 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 words. However, parsing algorithms used to process a CFG can also be adapted for dependency parsing. Eisner's dynamic programming algorithm [16] 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. 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, whilst chart parsing uses dynamic programming.

4.1.2 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 [22]. This work showed that a Constraint Dependency Grammar (CDG) parsing can be formalised as

a constraint satisfaction problem. To lower 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 [22].

Menzel and Schröder [24] 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. [28], who used a machine learning approach based on genetic algorithms to assign weights.

4.1.3 Transition-based approaches. A transition-based system is an abstract machine composed of states and transitions between the states. 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.

The transition actions are defined by the arc standard transition system by Collins [9]. 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 [34] who achieved state-of-the-art results using the English language. The authors used the annotated Penn treebank [21] for the experiments.

4.1.4 Graph-based approaches. Transition-based approaches use 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 [8, 15]. The first extensive work on graph-based dependency parsing is attributed to McDonald et al. [23].

4.2 Neural Networks

Chen and Manning [7] 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.

Weiss et al. [33] at Google adopted this paradigm to improve the state-of-the-art parsing with the release of SyntaxNet [2] which was considered as the world's most accurate parser with over 94% accuracy on well-formed English text. Placing this result into perspective, the annotator agreement between trained linguists on this task is between 96 to 97%. This indicates that parsers are approaching human performance [33].

4.3 CoNLL 2017 and Deep Learning techniques

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' [37], was dedicated to dependency parsers for an extensive number of languages that can operate in a real world setting. This task was possible because in the previous years, de Marneffe et al. [10] started 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 the CoNLL 2017 shared task, amongst the released 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 were not previously released in UD and were only available one week prior to the evaluation stage [37].

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 [13]. 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 [29]. This system ranked first on the surprise languages and

[2] https://research.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html

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 [2]. An important conclusion from CoNLL 2017 was that the surprise languages and those languages which had small treebanks were the most difficult to parse with the best accuracy under 50% [37].

The process employed by Shi et al. [29], 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 [27]. The graph-based parser was based on Eisner's algorithm [16] and used MST [8, 15] 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 [12], which was the same scoring algorithm used by Stanford's University's winning entry [13].

For the arc-labeling phase, a labeler proposed by Kiperwasser and Goldberg [20] 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[13] were attributed to the feature extraction phase [37].

4.4 A bootstrapping approach for Maltese

The work by Tiedemann and van der Plas [32], to our knowledge, is the only publish work which evaluated methods of parsing for the Maltese language. The authors focused their work on three widely used bootstrapping techniques: annotation projection, model transfer and translated treebanks.

Early bootstrapping techniques concentrated on annotation projection which is also known as data transfer [19]. 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. [19] that make it possible to project the annotations from one language to another through word alignment.

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 used 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. [31] and evaluation results proved that this approach performed better than the annotation projection [32]. The two main problems of this approach is the lack of quality translation and insufficient training data to create the models.

The Maltese treebank used by Tiedemann and van der Plas was the initial private version of the research conducted by Čéplö [25] to release the first Maltese annotated treebank. 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 from the European Union's Acquis Communautaire (AC).

For the evaluation, Tiedemann and van der Plas used a subset of 19 languages, including Maltese which had sufficient parallel translations. 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%. 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.

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. One of the experiments was conducted with the inclusion of inflection information from the *Korpus Malti*.

At the conclusion, 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 concluded that cross-lingual parsing was still lagging behind fully supervised models.

4.5 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 [11]. The annotation scheme used in UD is the evolution of previous efforts by the University of Stanford, Google and Interset [25].

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 [6]. The format of the UD treebanks was later updated and named CoNLL-U with the initial guidelines published in October 2014.

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 as *word line*, *empty line* and *hash tag*.

4.6 The Maltese Universal Dependencies

For this work, a private first version of the Maltese UD Treebank (MUDTv1) was provided by Čéplö [25].

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) [3] is the latest available and was one of the corpus used by Čéplö. This corpus was tagged with the Maltese Tagset v3.0 (MTSv3) and has an accuracy of approximately 97%. The other corpus is called bulbulistan maltiV3 (BCv3) which was developed by Čéplö, the same author of MUDTv1. Although these two corpora were developed independently, both works were published under one study by Gatt and Céplö [18] themselves.

5 METHODOLOGY

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

5.1 Maltese Word Embeddings

One of the first tasks required 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. From the word embeddings, a model is created and finally the model is mapped to the scatter plot.

[3] http://mlrs.research.um.edu.mt/index.php?page=corpora

The first step in generating word embeddings for the Maltese language is to acquire a source text in Maltese. The MLRS corpus (also known as *Korpus Malti*, Gatt and Céplö [18]) 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. 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. In a word embedding corpus, similar words will result in closely related vectors and hence, the lemma should be located close to the centre of a vector space which should also enhance context by adding more neighbours.

5.2 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 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. [5] 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.

Bradbury et al. [5] 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 de facto standard architecture for dependency parsing.

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.

For the language modelling experiments, Bradbury et al. performed the same experiments by Zaremba et al. [36] and Gal and Ghahramani [17] using the Penn Treebank [21]. In the main result, QRNN obtained a validation of 85.7% whilst [36] 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.

5.3 The Parser

This work is based on the research by Kiperwasser and Goldberg [20] which was also used by Dozat et al. [13], 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 [4] 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. 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.

These methods were implemented using the reference code for both the bi-LSTM and QRNN architectures. This gives the possibility to generate results for the two architectures and compare them for effectiveness and efficiency.

5.4 The Bootstrapped Multi-source Treebank

As detailed in Section 4.6, the Maltese annotated treebank, MUDTv1, was privately provided by Čéplö for this work. MUDTv1 was composed of just one text file whilst the CoNLL evaluation requires three files. Hence, the first required step was to split MUDTv1 by 60% as the training dataset, a development dataset which will contain 20% and the remaining 20% as the test dataset.

MUDTv1 contains approximately 2000 sentences, with the split training dataset now containing 1200 sentences and the rest split between the development and training datasets. As discussed in Section 4.3, Shi et al. [29] used parallel datasets 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 [16] 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.

Basing upon the work of Tiedemann and van der Plas [32], the languages which performed best for parsing Maltese were English, Spanish and Italian. To generate the multi-source treebank, a tool was developed to merge various treebanks with the number of annotated sentences constant across the merged treebanks. This whole process can be easily performed on other low-resourced languages in future.

5.5 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 [5] was used. This ensures that this work follows a defined standard 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 was performed using the development dataset (mt-ud-dev.conllu). This 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).

The three metrics used for evaluation during CoNLL 2017 are:

(1) Labeled 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. Language specific dependency labels are ignored.
(2) Unlabeled Attachment Score is the percentage of words that are assigned only the correct head.
(3) Weighted Labeled 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.

6 EXPERIMENTS, RESULTS AND DISCUSSION

In this section we describe in detail the full evaluation process conducted with the aim to benchmark the parser. In total, there are twenty five experiments categorized into five distinctive sets, with each set targeting a specific component of the parser.

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 is benefit to using an external word embedding. The third establishes the contribution of bootstrapped multi-source treebanks in the evaluation metrics, whilst the forth set of experiments examines a different neural network architecture with the aim to achieve a ground truth for the preferred architecture. The last set of experiments aim to

[4] https://github.com/elikip/bist-parser

[5] http://universaldependencies.org/conll17/eval.zip

provide metrics to compare our system with parsers which participated in CoNLL 2017. The experiments consists 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, thus providing figures for the UAS, LAS and Weighted LAS metrics.

6.0.1 Neural Network Optimization algorithms. This set of experiments 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.

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, Adam converges much earlier than Adamax, it was concluded that Adam is a more suitable optimizer for this task.

6.0.2 External Word Embeddings. The word embeddings algorithms used for this work are fastext 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.

Comparing 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 the 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.

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 observed result is a cloud with dense and less dense points indicating the proximity of the Maltese words to each other.

6.0.3 Bootstrapped Multi-source Treebank. 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.

Using bootstrapped multi-source treebanks, a better performance was obtained in all experiments over the single-sourced Maltese language treebank. One of the most important results of our work is the use of the Maltese and Romance multi-source treebank with fasttext external word embeddings. Comparing these metrics with the Maltese only treebank, the multi-source treebank results in better performance of 9.09% for LAS, 10.06% for UAS and 12.15% from Weighted LAS.

We can determine that multi-source treebanks performed exceptionally better when comparing to the work of Tiedemann and van der Plas [32]. Considering only 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.

6.0.4 Neural Network Architecture. One of the most important aspect of the work is the use of QRNN as the neural network architecture of the parser. 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.

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 [6] is unidirectional. According to Bradbury et al. [5], one of the most important contributions of QRNN is the runtime performance. Comparing the runtime performance results, 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 continue to outperform bi-LSTM even further.

6.0.5 Evaluating the approach. The objective of this set of experiments is to compare the predicted metrics from this parser to metrics from the proceedings of CoNLL 2017 [37] and two participants; Dozat et al. [13] and Shi et al. [29]. Two well-resourced languages; English and Spanish and two low-resourced Uyghur and Kazakh were used for these experiments. This exercise should gauge how well the parser performs when compared to other parsers.Dozat et al. [13] placed first overall and achieved the best scores for the well-resourced languages. Shi et al. [29] 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.

For the English language, our parser surpasses Dozat et al. [13] 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 also superior. For the Uyghur language, this work registered approximately 12.50% performance improvement across all three metrics when compared to Shi et al. [29]. For the Kazakh, the improvement was nearly of 9.00% for each metric when comparing again to Shi et al. [29]. Kazakh is the language with the smallest UD treebank [37].

6.1 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

[6] https://github.com/salesforce/pytorch-qrnn

results. Of course, it would be essential to run experiments with similar settings to measure and compare the actual performance.

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 is 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 determine if the best model was achieved and hence conclude the training phase successfully.

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 [32] 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.

7 CONCLUSION

In conclusion, it can be stated that all of the aims and objectives set for this work 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 a intelligible set of experiments to decide which technologies and process should be undertaken. 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 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 dependency 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. [13], 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 [32] by approximately 23% for an UAS of 90% and a LAS of 86%.

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 the language thrives in the digital era.

7.1 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.

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.

One of the main contributions of this work is the use of bootrapped 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 [32]. 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 [32]. 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 [32] experienced in their work.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Anders Björkelund, Agnieszka Falenska, Xiang Yu, and Jonas Kuhn. 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*. Association for Computational Linguistics, Vancouver, Canada, 40–51.

[3] Claudia Borg. 2015. *Morphology in the Maltese language: A computational perspective*. Ph.D. Dissertation. University of Malta.

[4] Claudia Borg and Albert Gatt. 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)* (26–31).

[5] James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2016. Quasi-Recurrent Neural Networks. *CoRR* abs/1611.01576 (2016).

[6] Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, 149–164.

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

[8] Yoeng-Jin Chu. 1965. On the shortest arborescence of a directed graph. *Science Sinica* 14 (1965), 1396–1400.

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

[10] Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. 2014. Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)* (26-31). Reykjavik, Iceland.

[11] Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, Vol. 6. Genoa Italy, 449–454.

[12] Timothy Dozat and Christopher D Manning. 2016. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734* (2016).

[13] Timothy Dozat, Peng Qi, and Christopher D. Manning. 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*. 20–30.

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

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

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

[17] Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*. 1019–1027.

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

[19] Rebecca Hwa, Philip Resnik, Amy Weinberg, Clara Cabezas, and Okan Kolak. 2005. Bootstrapping parsers via syntactic projection across parallel texts. *Natural language engineering* 11, 3 (2005), 311–325.

[20] Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *CoRR* abs/1603.04351 (2016).

[21] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics* 19, 2 (1993), 313–330.

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

[23] Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd annual meeting on association for computational linguistics*. Association for Computational Linguistics, 91–98.

[24] Wolfgang Menzel and Ingo Schröder. 1998. Decision Procedures for Dependency Parsing Using Graded Constraints. In *in proceedings of ACL'90*. 78–87. http://www.aclweb.org/anthology/W/W98/W98-0509.pdf

[25] Slavomír Čéplö. 2018. *Constituent order in Maltese: A quantitative analysis*. Ph.D. Dissertation. Charles University.

[26] Mike Rosner and Jan Joachimsen. 2012. *Il-Lingwa Maltija Fl-Era Diġitali – The Maltese Language in the Digital Age*. Springer. Available online at http://www.meta-net.eu/whitepapers.

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

[28] Ingo Schröder, Horia F. Pop, Wolfgang Menzel, and Kilian A. Foth. 2001. Learning Grammar Weights Using Genetic Algorithms. In *IN PROCEEDINGS RECENT ADVANCES IN NATURAL LANGUAGE PROCESSING, RANLP-2001*. 235–239.

[29] Tianze Shi, Felix G. Wu, Xilun Chen, and Yao Cheng. 2017. Combining Global Models for Parsing Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Association for Computational Linguistics, Vancouver, Canada, 31–39.

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

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

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

[33] David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. *arXiv preprint arXiv:1506.06158* (2015).

[34] Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, Vol. 3. Nancy, France, 195–206.

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

[36] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329* (2014).

[37] Daniel Zeman, Martin Popel, Milan Straka, Jan Hajic, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gokirmak, Anna Nedoluzhko, Silvie Cinkova, Jan Hajic jr., Jaroslava Hlavacova, Václava Kettnerová, Zdenka Uresova, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher D. Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria dePaiva, Kira Droganova, Héctor Martínez Alonso, Çağrı Çöltekin, Umut Sulubacak, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadová, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonca, Tatiana Lando, Rattima Nitisaroj, and Josie Li. 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*. Association for Computational Linguistics, Vancouver, Canada, 1–19.