The Open University of Israel
Department of Mathematics and Computer Science

Latest Developments in Morphological Disambiguation Strategies of Modern Hebrew

Thesis submitted as partial fulfillment of the requirements
towards an M.Sc. degree in Computer Science
The Open University of Israel
Department of Mathematics and Computer Science

By
Amit Seker

Prepared under the supervision of **Professor Reut Tsarfaty**

July 2021

Abstract

In morphologically rich languages words are ambiguous, complex and consist of sub-token units referred to as morphemes.

Such cases often requires Morphological Disambiguation (MD), i.e., the prediction of the correct morphological decomposition of tokens into morphemes. Morphemes are used as the actual input to downstream tasks, and consequently any disambiguation errors made are hard to recover from and might have adverse impact on the performance of the entire pipeline.

There are two possible strategies for achieving MD. The first is to structure this task as a two-step process known as Morphological Analysis and Disambiguation (MA&D). In this case words are morphologically analyzed first, transforming every word into a set of morphological analyses representing different out-of-context syntactic interpretation that follow the rules of the language and the disambiguation component is designed to choose the most likely analysis in context. The alternative is to perform MD as an End-to-End (E2E) process that directly operates on the words (i.e. without going through an intermediate representation). On the one hand, E2E models directly predict morphological information from the raw words but do not enjoy explicit access to morphemes. On the other hand, MA&D frameworks depend on the quality of the morphological analyses generated for each word – the analyzer must cover all possible interpretations in order to give the disambiguator a chance to choose the correct one.

In this work we apply 3 different morphological disambiguation models, 2 of them designed for MA&D while the third works E2E. By covering the different design choices of each model we track the impact of deep learning breakthroughs in NLP as they have contributed to Hebrew MD over the last years. The case studies presented in this work focus on Modern Hebrew, but all three disambiguation models can be applied to any lan-

guage.

We start by applying a pre-neural joint morpho-syntactic framework, AKA YAP, on all 82 languages of the CoNLL 2018 UD Shared Task and provide a deep analysis of the results in Modern Hebrew. We then propose a novel deep learning approach for disambiguation based on the Pointer Network topology - applied on both Turkish and Hebrew - achieving significant accuracy improvement over the YAP standalone Hebrew MD component. Both of these efforts highlight the contribution of broad coverage lexical resources for substantially limiting morphological errors, leading to high accuracy in downstream tasks. Finally we acknowledge the downside of these MA&D frameworks involved with constructing a language-specific broad-coverage MA component. Driven by this recognition we investigate an alternative solution that benefits from a different type of broad coverage linguistic resource which does not require linguistic expertise or manual annotation effort. We offer AlephBERT, a Pre-trained Language Model (PLM) trained in unsupervised learning settings over a large corpus composed of nearly 100M sentences in Modern Hebrew. We use this PLM to compose an E2E MD for Hebrew that achieves outstanding state-of-the-art results without the need for labor-intensive manual generation of lexical resources or feature engineering.

Acknowledgements

First and foremost, I would like to express my deep and sincere gratitude to my research supervisor, Reut Tsarfaty, Professor at Bar Ilan University and head of the ONLP Lab, for giving me the opportunity to do research and providing invaluable guidance throughout. Her vision, sincerity and motivation have deeply inspired me. She has taught me the methodology to carry out the research and to present my work as clearly and effectively as possible. It was a great privilege and honor to work and learn under her guidance.

I am extremely grateful to my family for their love and for their continuously and unequivocally supporting me in completing this research work.

Thank you.

Contents

List of Figures

List of Tables

1 Introduction

Natural language processing (NLP) is a sub-branch of Computer Science concerned with processing and automatically extracting *linguistic structure* from free - i.e. unstructured - texts expressed in natural human language such as English or Hebrew. Once extracted, these structures serve downstream processing tasks for Information Extraction, Text Analytics, and diverse Data Science applications. As such, NLP stands at the foundation of Artificial Intelligence.

Deep learning is driving major breakthroughs in a wide range of NLP tasks ranging from basic low level tasks such as Part-of-Speech (POS) Tagging, Named Entity Recognition (NER) and Dependency Parsing all the way to high level Machine Translation and Question Answering applications. Recently, large Pre-trained Language Models (PLMs) have become ubiquitous in the development of language understanding technology and lie at the heart of many artificial intelligence advances. Notably, advances have been demonstrated mostly on English and other highly researched languages, while reported advances in Hebrew are few and far between.

The problem is twofold. First, Hebrew resources for training large language models are not at the same order of magnitude as their English counterparts. Secondly, Hebrew is a Morphologically Rich Language (MRL) in which significant information is expressed morphologically, via intra word-level variation, leading to diverse and ambiguous structures, accompanied with huge lexica, which in turn make MRLs notoriously hard to parse (Nivre et al., 2007; Tsarfaty, 2013). English, on the other hand, is a language with simple morphology in which syntactic information is expressed via inter word relationships.

English has a broad support of annotation pipelines successfully serving Artificial Intelligence (AI) and Data Science projects in academia and the industry. But when applied to MRLs, these pipelines show sub-optimal

performance that limits their applicability for text analysis. The reason being that MRLs often required Morphological Disambiguation (MD), i.e., prediction of correct morphological decomposition of tokens into morphemes, early in the pipeline and the sub-optimal performance is mainly due to errors in early morphological disambiguation decisions, which cannot be recovered later in the pipeline, yielding incoherent annotations on the whole.

As we will see, in order to improve processing performance in MRLs in general and Hebrew in particular we need to address 3 strategical challenges as defined by Tsarfaty, Bareket, Klein, and Seker [70]:

> (i) The Architectural Challenge: What input units are adequate for processing MRLs?
>
> (ii) The Modeling Challenge: What modeling assumptions are adequate for MRLs?
>
> (iii) The Lexical Challenge: How can we cope with extreme data sparseness in MRLs lexica?

1.1 Morphologically Rich Languages

The term *Morphologically Rich Languages* (MRLs) [69] refers to languages such as Hebrew, Arabic and Turkish in which significant grammatical information and syntactic relations are expressed at word-level as opposed to via word-order and inter-word constructions as in English. Words in MRLs tend to consist of a sequence of *sub-word segments* each assigned its own Part-of-Speech (POS) tag and other grammatical features such as gender, number, tense, etc. These sub-token units are referred to as *morphemes*.[1]

A morpheme is defined as a tuple containing several types of syntactic information. The full morphological signature defines the syntactic role of morphological segments and contains the following:

(1) Form - orthographic representation of the surface segment.

(2) Lemma - canonical representation of the form.

[1] In Universal Dependencies terms, these are called *syntactic words*, to be distinguished from *raw input words*.

(3) Part-of-Speech Tag

(4) Features - various grammatical features such as: gender, number, person, tense, etc.

1.1.1 Syntactic Differences Between English and Hebrew

English adheres to a certain morpho-syntactic set of rules - it obeys word order and uses auxiliary verbs, adpositions and other inter-word relationships to express the grammar that governs the language. Space delimited words are (mostly) self-contained, i.e. with no internal structure, represent a single syntactic as well as semantic interpretation and therefore serve as the basic processing units in NLP. This in turn affects the design and implementation of downstream NLP tasks - the input sequence is well defined and known in advance and the output is some word-based structure depending on the task. So for example when performing Part-of-Speech (POS) Tagging each word by itself may have more than a single possible tag and it is up to the tagger to choose the most likely sequence of POS tags, one per word, in the context of the sentence. Likewise a Named Entity Recognition (NER) task generates a list of NER labels one per word and a Dependency Parser infers syntactic relations between words and constructs the most likely parse tree associated with the sentence.

In Hebrew, words are morphologically ambiguous, complex, carry many possible interpretations and therefor should be broken down before further processing. This gets even more complicated by the lack of diacritics in standardized texts, meaning that most vowels are not present, and thus out of context no reading is a-priory more likely than the others. Only in context the correct interpretation and segmentation into morphemes become apparent, and it is in fact the morphological segments that serve as the basic processing units in downstream applications. So in this case, before choosing the most likely sequence of POS tags or NER labels in context it is required to first identify the morphological segments and only then assign each segment with a label. Similarly a dependency parser generates a parse tree on segmented forms.

Let's look at the following example:

Hebrew Phrase: "הלכנו לבית הלבן"

Transliterated: *"hlknv lbit hlbn"*[2]

[2]Using the transliteration of Sima'any et al. [60]

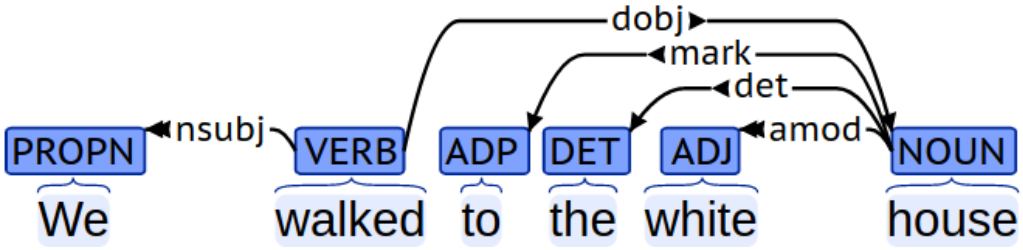

Figure 1.1: A parse tree for English - POS tags and dependency relations are word-level.

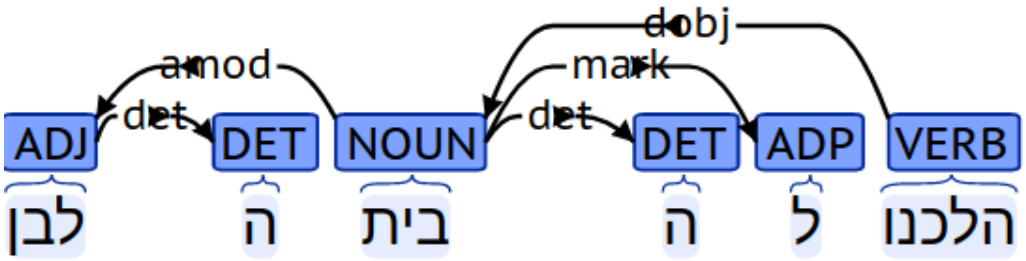

Figure 1.2: A parse tree for Hebrew - POS tags and dependency relations are morpheme-level.

Translated: "We walked to the white house"

The parse tree for the English sentence is depicted Figure 1.1 in which words are used as tree nodes with edges between them defining the parsed syntax. Compare this with Figure 1.2 depicting the same phrase in Hebrew, expressed with 3 words but broken into 6 sub-word units depicting the dependency tree associated with the morpheme sequence.

The overall ambiguity level in Hebrew, and MRLs in general, is compounded - words are ambiguous with regard to their segmentation and segments are ambiguous by themselves in various aspects such as POS tags, dependency relations, etc.

To make things worse, some of these sub-word morphological units might be implicit in the orthographic word form as seen in the Hebrew word "*lbit*". In this case the word might break into 3 morphological segments, "*l*", "*h*" and "*bit*" corresponding to the English counterparts "*to*" "*the*" "*house*" where the determiner "*h*" segment is not visible in the word form.

The high level of ambiguity built into words in MRLs present a major obstacle to the entire NLP pipeline. As shown by Goldberg and Elhadad [20], any errors made during morphological segmentation early on are critical

to the overall performance of the entire Hebrew NLP pipeline, because they are hard to recover from in the downstream tasks.

The fact that in English it is space-delimited tokens that are employed as the basic processing units, has significant impact on the advancements made in numerous NLP tasks in English. Many MRLs, and Hebrew in particular, do not benefit from these advancements due to the fundamental differences at the input level which pose several challenges as described next.

1.2 Hebrew Resources

Labeled and unlabeled textual resources stand at the core of NLP. Labeled datasets serve as the backbone for various supervised statistical parsers. Supervised methods rely on labeled data to learn from and fit model parameters (i.e. train), find the best training and model meta-parameters (e.g. learning rate, model size, number of training epochs, etc.) as well as provide accuracy performance measure (i.e. test). Unlabeled datasets are heavily used in unsupervised tasks, such as word embedding and language modeling, that are driving the successful application of deep learning in NLP.

In English, creating word-level labeled resources do not require further pre-processing before annotation, and unlabeled space-separated texts are (relatively) straight forward to extract. Consequently, English is a resource-rich language with a large number of available datasets.

Hebrew, besides being morphologically rich, is also known to be resource poor. Not only is the amount of available texts available for Hebrew is smaller, using these texts in NLP tasks that require morphological information require a pre-step to segment words into morphological units.

Even today after decades of NLP research, the number of available annotated resources which can benefit Hebrew NLP is minimal. The first morphologically-aware resource for Hebrew have been developed during the early 2000s, by the MILA knowledge center. Namely they are the Hebrew treebank [60] and the Hebrew Lexicon [26]. Over the years these resources have been expanded and improved upon and have served as the building blocks for statistical taggers [1, 6] and parsers [20, 43] in Hebrew. The Hebrew treebank was included in the SPMRL 2013 SHARED TASK [53] and SPMRL 2014 SHARED TASK [54] that introduced to the NLP community the unique processing challenges associated with MRLs.

Most recently More and Tsarfaty [39] has automatically transformed the Hebrew treebank into the UD scheme and added it to the UD dataset which is part of the CoNLL 2018 UD Shared Task . Contrary to the SPMRL tasks, the latest Universal Dependencies (UD) initiative aims to provide cross-linguistic morpho-syntactic data sets under a unified, harmonized annotation scheme [46].

Data Sparseness Hebrew datasets exhibit word-level sparseness that follows from the highly complex and compositional structure of words. Consequently many words are not seen during training even though their constituent morphological segments might have been present in the data.

In addition, we point out that the Hebrew Treebank is a small-sized dataset - the training partition is about 4000 sentences which is considered to be relatively modest for training syntactic models such as POS tagging, NER and dependency parsing. Even though the treebank does contain morpheme-level information, due to the small training partition size many morphological forms are not seen during training.

To build NLP models that can effectively handle texts in Hebrew we must account for the fact that many words and many sub-word segments are not seen during training and therefore considered out-of-vocabulary (OOV). To overcome data spareness we set out to test 3 mechanisms that introduce additional information beyond labeled data, augmenting systems with extensive linguistic knowledge.

First we demonstrate the effectiveness of injecting wide-coverage Morphological Analyzer (MA) component in parsing Hebrew texts. Specifically we take advantage of the CoNLL-U and CoNLL-UL datasets that constitutes the MA output complementing the train/dev/test partitions of the Hebrew SPMRL and UD Treebank respectively and use them as input into a morpho-syntactic parsing system.

Secondly, we experiment with pre-trained FastText Hebrew model that can generate embedded vectors for any input by using character ngrams, thus handling any out-of-vocabulary form [9].

Finally, we build a large BERT language model capturing syntactic and semantic information learned from massive amounts of Hebrew texts. We deploy this language model in a morphological disambiguation component and demonstrate its ability cover any Hebrew text thus overcoming the OOV problem.

1.3 Morphological Modeling

As previously mentioned, models that are applied successfully to English fail when applied as is on MRLs. These models implicitly assume the space-delimited input word sequence is known in advance and do not take into consideration the internal structure of words. In Hebrew, however, morphological segments serve as the basic processing units. Consequently when designing text processing solutions in MRLs, we are faced with a decision. If we choose to ignore the morphological issue and process words directly we might run into performance issues for various processing tasks [28]. On the other hand, dealing directly with morphological segments introduces further challenges in designing morphologically-aware models and evaluation pipelines.

It is therefore no surprise that MRLs have been known to be notoriously hard to parse ([45, 68]). The results in SPMRL 2014 SHARED TASK and CONLL 2018 UD SHARED TASK clearly indicate that the performance on many MRLs, and in particular Semitic languages, is not on par with the performance on English. For Hebrew this performance gap has long been a show-stopper for advancing Language Technology and Artificial Intelligence for the Hebrew-speaking community.

In this work we focus on the *Morphological Disambiguation* (MD) task which we identify as the most critical component in MRLs. We investigate 3 different MD approaches and compare the performance gained by each one on various tasks in Hebrew.

We start by presenting our contribution to the CONLL 2018 UD SHARED TASK on MULTILINGUAL PARSING FROM RAW TEXT TO UNIVERSAL DEPENDENCIES. Our contribution is based on a pre-neural transition-based parser called *yap: yet another parser* which includes a standalone morphological disambiguation model, a standalone dependency model, and a joint morpho-syntactic model. In the task we used *yap*'s standalone dependency parser to parse morphologically disambiguated input and obtained the official score of 58.35 LAS. In a follow up investigation we use *yap* to show how the incorporation of morphological and lexical resources may improve the performance of end-to-end raw-to-dependencies parsing in the case of a *morphologically-rich* and *low-resource* languages, such as Modern Hebrew. Our results on Hebrew underscore the importance of CoNLL-UL, a UD-compatible standard for accessing external lexical resources, for enhancing end-to-end UD parsing.

We then move on to a neural-MD architecture that combines the symbolic

knowledge of morphemes with the learning capacity of end-to-end deep-learning modeling. We propose a new, general and easy-to-implement Pointer Network disambiguation model where the input is a morphological lattice and the output is a sequence of indices pointing at a single disambiguated path of morphemes. We demonstrate the efficacy of the model on segmentation and tagging, for Hebrew and Turkish texts, based on their respective Universal Dependencies (UD) treebanks. Our experiments show that with complete lattices, our model outperforms all shared-task results on segmenting and tagging these languages. On the SPMRL treebank, our model outperforms all previously reported results for Hebrew MD in realistic scenarios.

Last but not least, we pursue the latest advancements in NLP spearheaded by *Pre-trained Language Models* (PLM). We present *AlephBERT*, a large pre-trained language model for Modern Hebrew, which is trained on larger vocabulary and a larger dataset than any Hebrew PLM before. Using *AlephBERT* we present new state-of-the-art results on multiple Hebrew tasks and benchmarks, including: Segmentation, Part-of-Speech Tagging, full Morphological Tagging, Named-Entity Recognition and Sentiment Analysis. Crucially, evaluating PLMs on morpheme-level tasks is non trivial. Current PLM implementations output word-level embedded vectors, while sub-word morpheme-level vectors which are required for morpheme based tasks are not readily available. To address this we introduce a neural MD component that operates on contextualized word vectors, extracts morphological segments and outputs vector representations that can be used to further label each morphological segment with various properties such as POS tags, NER labels and morphological features.

2 Background and Formal Preliminaries

Here we provide an overview of the main morphological processes involved in the formation of words in MRLs (2.1), we formally define the morphological analyzer function that maps words to a set of possible morphological analyses (2.2) and describe the Lattice data structure that concisely represents morphological ambiguity (2.3). We then go over two possible strategies for performing the morphological disambiguation task (2.4) and conclude by covering the differences between the 3 modeling choices surveyed in this work (2.5). Finally we describe embedded word vectors and their contribution to the successful adaption of deep learning in NLP (2.6).

2.1 Morphological Typology

Every language can be described in terms of the processes involved in forming words. On the one end of the spectrum are *isolative* or *analytic* languages in which most words tend to comprise of a single morpheme. *Analytic* languages, such as Chinese and English, have a lower morpheme-to-word ratio and higher use of auxiliary verbs and word order to convey syntactic relationships. As we move away toward the other end of the spectrum we find *synthetic* languages whose words tend to contain multiple morphemes. *Synthetic* languages can be further described according to their *agglutinative* (concatenative) and *inflectional* (fusional) properties.

Agglutination is a derivational process that concatenates two or more morphemes into a single word without modifying spelling or phonetics. *Inflection* on the other hand, adds functional affixes to a root morpheme which assigns grammatical properties to the root. In the former, the morphemes

tend to have clear boundaries between them whereas in the latter, the morphemes tend to be indistinguishable. The following Hebrew phrase illustrates the difference between agglutination and fusion:

Hebrew phrase: "בצלם הנעים"

Transliterated: "*bclm hneim*"

Disambiguated: *b/ADP,h/DET,cl/NOUN,fl/POS,hm/PROPN,h/DET,neim/ADJ*

Literally: "In-the-shadow-of-them the-pleasant"

Translated: "In their pleasant shadow"

The first word, "*bclm*", can break down into five morphemes where three of them - the second, fourth and fifth, corresponding to the segments: "*h*", "*fl*" and "*hm*" - are fused and have no visible boundaries in the surface word form. The second word, "*hneim*", can be segmented into two morphemes "*h* + *neim*", with clear boundaries between the two.

In this work we focus on Hebrew, a highly *synthetic* language where word form ambiguity is common, i.e. words can be segmented in different ways depending on context. As reflected by the above example, words can exhibit both agglutinative and fusional properties to various degrees and may contain covert segments making it more challenging to recover.

2.2 Morphological Analysis

Morphemes A *morpheme* is defined as a tuple holding lexical information, such as the form and lemma, as well as non-lexical categorical information including POS tag and other features that convey grammatical attributes such as gender, number, person, tense, etc. Formally we define a morpheme as a tuple

$$m = (f, l, t, g) \tag{2.1}$$

with f and l the form and lemma, t a tag, and g a set of (possibly empty) 'attribute=value' grammatical properties.

The form is the surface appearance of the morphological segment, and can replace the word as the orthographic unit used by downstream tasks such as NER or parsing. Lemma is the canonical representation of meaning of the form, it is based on the normalized version of the word as male, singular, third person, past tense (if it is a verb). For example the Hebrew

word which represents the phrase "I walked" is composed of a single morpheme:

$$m = \big(\textit{"hlkti"},\textit{"hlk"},VERB,gender{=}Male|number{=}Singular|person{=}First|Tense{=}Past\big)$$

In the above, the form is identical to the surface word and the lemma is obtained by inflecting the word as male, singular, third person, past tense.

An *analysis* is defined as a list of morphemes representing the entire decomposition of the word into its constituent components:

$$Analysis^i(w) = [m_1, m_2, ...] \tag{2.2}$$

E.g. the phrase "in the houses", represented by the transliterated word "*bbtim*" and its full breakdown into morphological components is:

$$Analysis(\textit{"bbtim"}) = [{}^{(\textit{"b"},\textit{"b"},ADP,\_),(\textit{"h"},\textit{"h"},DET,\_),}_{(\textit{"btim"},\textit{"bit"},NOUN,gender{=}Male|number{=}Plural)}]$$

A *Morphological Analyzer* (MA) converts each word to a set of morphological decompositions, licensed by the rules of the language, each equally likely out of context:

$$MA(w) = \{a^1, a^2, ...\}, a^i = Analysis^i(w) \tag{2.3}$$

For example, the word "hlbn" by itself (i.e. out-of-context) is ambiguous because there are several possible ways to interpret this word in the Hebrew language:

$$a_1(\textit{"hlbn"}) = [{}^{(\textit{"h"},\textit{"h"},DET,\_),}_{(\textit{"lbn"},\textit{"lbn"},ADJ,gender{=}Male|number{=}Singular)}]$$

$$a_2(\textit{"hlbn"}) = [{}^{(\textit{"h"},\textit{"h"},DET,\_),}_{(\textit{"lbn"},\textit{"lbn"},NOUN,gender{=}Male|number{=}Singular)}]$$

$$a_3(\textit{"hlbn"}) = [{}^{(\textit{"hlbn"},\textit{"hlbin"},VERB,}_{gender{=}Male|number{=}Singular|person{=}First|tense{=}Past)}]$$

Lexicon Implementing a broad-coverage MA involves a language specific, manually generated set of rules, stored in a designated linguistic resource, i.e. *lexicon*. A lexicon is a data structure storing lexical entries that are used to map every word to the set of all possible analyses governed by the rules of the language.

For example, table 2.1 lays out the lexical entries applied in order to cover all of the valid morphological analyses associated with the Hebrew word "הלבן" (transliterated *"hlbn"*) according to the rules of the Hebrew language.

Lemma	Tag	gender	number	person	tense	binyan	Suf_Tag	Suf_gender	Suf_number	Suf_person
h	DET	_	_	_	_	_	_	_	_	_
lbn	ADJ	Male	Single	_	_	_	_	_	_	_
lbn	NOUN	Male	Single	_	_	_	_	_	_	_
lbn	PROPN	Male	Single	_	_	_	_	_	_	_
libn	VERB	Male	Single	3rd	Past	Piel	_	_	_	_
lb	NOUN	Male	Single	_	_	_	PRON	Female	Plural	3rd

Table 2.1: Lexical entries relevant for the morphological analysis of the word "הלבן" (*"hlbn"*).
"Suf_*" stands for grammatical properties associated with the suffix units.

Some words might not be covered by the lexicon, in which case the MA need to generate a default list of possible analyses as a fall back strategy.

2.3 Morphological Ambiguity

We've established the fact that input tokens in MRLs are internally complex, and bear multiple units of meaning. *Morphological Analysis* (MA) is aimed to convert each of the tokens to a set of all possible morphological decompositions licensed by the rules of the language. Every decomposition represents one possible interpretation of the token being analyzed.

For every word in the language, MA outputs multiple, ambiguous, morphological analyses which can be stored in a *lattice* data structure. A lattice is *Directed Acyclic Graph* (DAG) often used to encode ambiguity in NLP. In a morphological lattice, every node represents a segment boundary, and every edge represents a morpheme. Every path through the lattice represents a single possible analysis of the entire sentence.

Formally, we define for each word x_i its *word-lattice*:

$$L_i = MA(x_i) \tag{2.4}$$

where each *lattice-edge* in L_i corresponds to a morpheme with internal representation as defined in equation 2.1.

The complete sentence lattice is obtained by concatenating the word lattices. L is now a data structure that encodes all possible analyses applicable to the sentence $x_1...x_k$:

$$L = MA(x) = MA(x_1) \circ ... \circ MA(x_k) \tag{2.5}$$

Consider again the Hebrew phrase "bbit hlbn". A partial lattice representation of the analyses is illustrated in Figure 2.1.

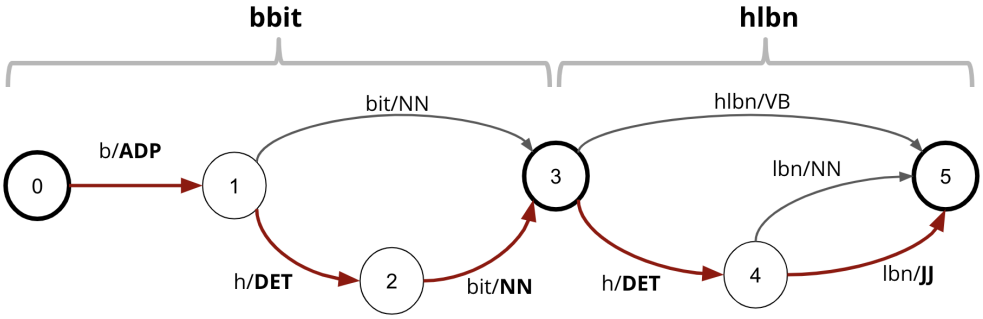

Figure 2.1: Lattice of the Hebrew tokens *"bbit hlbn"*. Edges are morphemes. Nodes are segmentation boundaries. Bold nodes are token boundaries. Every path through the lattice represents a single morphological analysis.

In chapter 4 and chapter 5 of this work we cover 2 modeling choices that directly represent morphological ambiguity in the form of a Lattice data structure which is consumed as input by the respective disambiguation components.

2.4 Morphological Disambiguation Strategies

Morphological Disambiguation (MD) is the task of selecting a single most-likely analysis for each token in the context of the sentence. The resulting morpheme sequence may then serve as the input processing units for downstream tasks (similarly to space-delimited words in English).

The *Morphological Disambiguation* (MD) task is defined to take a sequence of (space-delimited) words as input and output a sequence of *morphemes* where the length of the output sequence may differ from the length of the input sequence. In order to extract a sequence of morphemes from words we need to overcome the two types of ambiguities - morphological segmentation ambiguity and syntactic ambiguity. The former rises whenever words may be decomposed into different candidate sequences of segments while the latter is a consequence of multiple plausible syntactic roles that can be assigned to each of the segments.

Traditionally MD has been addressed by two different disambiguation methods. The first operates on the raw word sequence to directly predict morphological information. The second method involves a two-step process, transforming the word sequence into a morphological lattice (as

defined in the previous section) before proceeding to choose the single most likely path through the lattice, which represents the most likely morphological analysis of the sentence. The former method is referred to as End-to-end (E2E) while the latter is referred to as Morphological Analysis and Disambiguation (MA&D).

2.4.1 End-to-End Morphological Disambiguation

The most straightforward E2E approach views MD as a morphological segmentation task. In this case, the disambiguation model takes a sequence of words as input and predicts the most likely sequence of segmented morphological forms. Once predicted, the new sequence of segments can now be used as input to other downstream tasks such as POS tagging, NER and dependency parsing. As previously mentioned, each word may break down into a number of segments corresponding to the morphological composition of the word, and as a consequence the length of the resulting segmented sequence is unknown in advance and may vary depending on the context.

This segmentation-first approach is, in essence, an attempt to mimic NLP in English, replacing space-separated words with the predicted sequence of forms. The idea of this approach is that once the segmented forms are predicted, it is possible to benefit from applying the same models that work so well for English.

In actuality, however, such pipeline architectures are prone to error propagation which undermines the accuracy of almost any task down the NLP pipeline [7, 28, 70]. NLP models that are optimized for English do not expect and have no capacity to deal with errors in the input stream.

Formally, MD pipeline is realized as a composition of functions:

$$MD(x) = g(f(x)) \tag{2.6}$$

where $f(x) = y$ is a segmentation function taking a sequence of words and generating a sequence of segments. and $g(y) = z$ is a grammatical function taking a sequence of segments and enriching each segment to construct a sequence of morphemes.

A second conceivable E2E approach is implemented in terms of a *sequence-to-sequence* model that consumes a sequence of words and produces a sequence of morphological signatures. For example, it is possible to jointly

predict both segmented forms and POS tags in a multi-task setup facilitating the interaction between the orthographic and grammatical signals to benefit in both learning as well as decoding. Defined this way, the MD training objective is to optimize both segmentation and tagging together as opposed to the pipeline regime where each model is trained independently and without being affected by quality of the other model.

To clear up the difference between pipeline and multi-task E2E approaches let's consider again the example "bbit hlbn". In a pipeline, MD is applied first and it breaks down this phrase into segmented forms: "b h bit h lbn", then this five-segment sequence can be passed as input to the next task. Downstream models for POS tagging or NER will process this sequence of segments as if they are the original sequence of raw words:

$$\begin{aligned} MD_{pipeline}([bbit,hlbn]) &= g(f([bbit,hlbn])) \\ &= g([b,h,bit,h,lbn]) \\ &= ([b,h,bit,h,lbn],[ADP,DET,NOUN,DET,ADJ]) \end{aligned}$$

In multi-task settings, on the other hand, MD directly and simultaneously predicts both morphological segments as well as other grammatical properties associated with each segment:

$$MD_{multitask}([bbit,hlbn]) = [b/ADP,h/DET,bit/NOUN,h/DET,lbn/ADJ]$$

Both Pipeline and Multitask approaches are considered end-to-end because they operate directly on the input sequence of words.

The drawback of E2E MD lies in lack of access to morphological information in the raw token input stream. Tokens in MRLs are lexically and syntactically ambiguous, and carry many possible interpretations.

There is an alternative that combines symbolic knowledge of morphemes, encoded as a lattice, with the learning capacity of machine learning and deep learning models, as described next.

2.4.2 Morphological Analysis and Disambiguation

Statistical approaches for MA&D typically use weighted finite-state machines to unravel the possible morphological decompositions, and machine learning models to select the most likely decomposition. Every raw token in the input sequence first goes through *Morphological Analysis* (MA)

exposing all possible morphological decompositions as a lattice (see Figure 2.1). The ambiguous morphological lattice is then passed to a disambiguation component selecting a sequence of arcs which represents the most likely analysis in the context of the sentence being processed. Since every lattice arc contains rich information made available by the MA — namely, segmentation boundaries, lemma, POS tag and a set of morphological features — this MA&D framework jointly predicts rich morphological layers while avoiding the pipeline pitfall.

Enjoying rich grammatical information does come at a price incurred by relying on a language specific MA component. Under the MA&D regime, not only is it necessary to consume ambiguous lattices as input, the correct analysis must be represented in the lattice otherwise the MD would not be able to choose it. MA&D cannot be applied if either a language-specific MA is not available, or if the analyzer provides a narrow lexical coverage resulting in small and partial lattices from which the MD cannot learn and generalize.

Since the advent of Neural Network models in NLP [19], many tasks rely on pre-trained dense vector representations of words, aka word embedding. Word embedding encode multiple aspects of structure and meaning which shift the focus from modeling various linguistic complexities to fine-tuning models that extract task-specific signals from the embedded vectors. Similar to a lexically driven MA component, having embedded vectors at hand increases the capacity of both E2E and MA&D frameworks to overcome word spareness and better handle morphological segments that are OOV.

2.5 Neural MD and Word Embeddings

Classical NLP models rely heavily on carefully hand-crafted features (or feature-templates). In contrast, neural networks (a.k.a deep learning models) for NLP are designed to encode raw sequences of words free from any manual and time-consuming feature engineering. In particular, in such architectures, words are first transformed into dense vector representations, referred to as word embedding.

Embedded vectors are learned in an unsupervised fashion from large volumes of textual corpora without requiring manual annotation. They implicitly encode syntactic and semantic features exhibited by words based on their distributional properties found in the training data. The abil-

ity of supervised networks to generalized and decode input sequences is dramatically improved even when they include out-of-vocabulary (OOV) words - words that were not seen in the annotated training data but might be similar (i.e. close in embedding vector space) to other words that were observed in training.

Initially, word embedding layers were provided as a static mapping from words to their corresponding dense vector representations. The work of Mikolov, Chen, Corrado, and Dean [37] (Word2Vec) and Pennington, Socher, and Manning [48] (GloVe) are such examples. The latest language models, such as BERT [16], offer contextualized embedding by dynamically assigning dense vectors to occurrences of words (so different vectors could be assigned to occurrences of the same word in different contexts). While advances reported for English using such models are unprecedented, in Hebrew previously reported results using BERT-based models are far from impressive. Specifically, the BERT-based Hebrew section of multilingual-BERT [16] (henceforth, mBERT), did not provide a similar boost in performance to what is observed for the English section of mBERT. In fact, for several reported tasks, the mBERT model results are on a par with pre-neural models [28, 70].

2.5.1 Word Embedding in MRLs

Unsupervised pre-training techniques are applied to raw words rather than morphemes, and deliver word-embedding vectors agnostic to internal word structure. Word embedding was expected to provide a language-independent representation of the input which potentially should eliminate the need to model different languages differently. In reality though, applying neural models that work well in English led to sub-optimal performance when applied on MRLs. While some morphological structure may be implicitly encoded in these vectors, the morphemes themselves remain inaccessible [15, 71].

In this work we introduce 2 solutions that provide morphologically-aware embedding layers (static and dynamic) and demonstrate their effectiveness in capturing morphological level features that benefit downstream tasks.

3 Previous Work

3.1 E2E Morphological Disambiguation

Historically, *Morphological Disambiguation* (MD) has been viewed in several ways by researchers depending on task goals. Earlier work on MD concentrated on the specific task of *morphological segmentation* only. Morfessor [61] performs morphological segmentation across many languages by supporting a family of generative probabilistic machine learning methods for finding the morphological segmentation from raw text. Wang, Cao, Xia, and de Melo [73] tackled the segmentation challenge by taking an unsupervised approach for learning segment boundaries. Recently, Shao, Hardmeier, and Nivre [57] modeled the segmentation task as a character-level sequence labeling problem. While improving segmentation results across many languages, Arabic and Hebrew accuracy remained low. We, on the other hand, incorporate linguistic information in frameworks that can predict both segments as well as other morphological layers.

Another popular MD scenario is commonly known as *morphological tagging* where every word is assigned a single morphological signature. In other words, every word is assigned a single lemma, a single main POS tag, and all other information (including affixes and clitics) is represented as features. So, for example, the word "hbit", which represents the phrase "the house", would get assigned the single morpheme ("*hbit*", "*bit*", *NOUN*, $gender = Male|number = Singular|def = true$), where the definite article information is represented as an extra feature as opposed to a separate morpheme. Morphological tagging has been applied using generative probabilistic frameworks in Turkish [23] and Arabic [18]. Mueller, Schmid, and Schütze [44] used a discriminative feature-based high order CRF tagging models, in a coarse-to-fine paradigm and ran tagging experiments on Arabic, Czech, Spanish, German and Hungarian. Our work is concerned with morpheme level tagging as opposed to word level tagging.

In order to extract morpheme level tags, Straka and Straková [64] developed *UDPipe* which, as the name implies, is a processing pipeline compatible with the UD schema. They offered a trainable pipeline which performs, among other tasks, tokenization, lemmatization and POS tagging. By taking the pipeline approach, they suffered from error propagation as evident from the POS tagging results of MRLs in the CONLL 2018 UD SHARED TASK . Like UDPipe we identify both morphological segments as well as assign a single POS tag to each segment. In fact we consider the UDPipe results from the CONLL 2018 UD SHARED TASK as the segmentation and tagging baselines in some of our experiments.

3.2 MA&D Morphological Disambiguation

A Morphological Analyzer maps words into all possible out-of-context analyses according to word formation rules in a specific language. Consequently, MA&D frameworks are forced to focus either on a specific language or family of languages depending on the analyzer component.

Yildiz, Tirkaz, Sahin, Eren, and Sonmez [75] implemented MA&D framework for for Turkish and applied it on German and French as well. They first embed various morphological properties of each analysis, pass it to Convolutional (CNN) layer that encoded each candidate analysis along with a context window of correct embedded analyses. Finally, a binary classification layer was used to calculate the likelihood that the analysis is correct or not given the window of previously disambiguated analyses, repeated for every analysis provided by the MA. Decoding was done using the Viterbi algorithm to pick the most likely sequence of analyses. The morphological embedding layer proposed in chapter 5 of this work is designed similarly, however we use a different topology in the second layer, and our final softmax layer is designed to compute the likelihood of each analysis as opposed to a binary classification score.

Shen, Clothiaux, Tagtow, Littell, and Dyer [59] utilized a compatibility function between the representation of each candidate analysis and the representation of the context. They encoded each analysis by concatenating character level BiLSTM over the stem with a LSTM encoding of all the grammatical features. The compatibility function computed a dot product between a matrix representation of the analyses and the context of the word being disambiguated before passing it to a softmax layer for computing the likelihood of each analysis given the context. They also experi-

mented with using the dot product as the emission value to a Conditional Random Field (CRF) which can make better global predictions. In chapter 5 among the various neural components we also apply the Attention mechanism which acts very similarly to the compatibility function.

3.2.1 Hebrew MA&D

Initial work on MD in Hebrew viewed it as a special case of POS tagging. Bar-haim et al. [6] implemented *MorphTagger* – Hidden Markov Model (HMM) trained under supervision augmented with a unsupervised smoothing using a large unlabeled corpus. Adler and Elhadad [1] showed tagging models whose terminal symbols are morphological segments are advantageous over word-level models for Modern Hebrew. They trained a semi-supervised HMM model supported by a morphological analyzer to help in word segmentation and a large unlabeled corpus to compute the probability distribution of tags over segments. Both of these works applied generative disambiguation models taking the morphological analyses of Segal [55] as input and computing for each sequence of morphemes the probability distributions of lexical word emissions as well as tag transitions. While the performance was adequate for some applications, Goldberg and Elhadad [20] showed that consuming the predicted MD output of Adler and Elhadad [1] as input to *dependency parsing* significantly reduced parsing performance on Hebrew.

Recently More and Tsarfaty [39] presented a standalone transition-based MA&D which jointly solves morphological segmentation, tagging and feature assignment in a discriminative feature-based framework, presenting new state-of-the-art for Hebrew MD. To overcome error propagation inherent in the pipeline approach, More et al. [43] proposed joint morpho-syntactic framework which enable interaction between the morphological and syntactic layers. Our submission to the CoNLL 2018 UD Shared Task described in chapter 4 is based on this joint framework. While proving to be state-of-the-art for both MD and dependency parsing on Hebrew, this solution involved massive time-consuming hand-crafted feature engineering. Our proposal in chapter 5 follows the MA&D framework solving for segments and POS tags by applying a neural model fit for choosing the most likely morphological analysis provided by a MA component.

3.2.2 Arabic MA&D

Arabic is a Semitic language with rich morphology closely related to Hebrew. As such we survey here alternative approaches for MA&D that work on Arabic. Both Habash and Rambow [21], Roth, Rambow, Habash, Diab, and Rudin [50] used Arabic MA output and applied a set of classification and language models to make grammatical and lexical predictions. A ranking component then scored the analyses produced by the MA using a weighted sum of matched predicted features.

Zalmout and Habash [76] presented a neural version of the above system using LSTM networks in several configurations and embedding levels to model the various morphological features and use them to score and rank the MA analyses. In addition, they incorporated linguistic features based on the analyses from the MA into the MD component. By enriching the input word embedding with these additional morphological features they increased MD accuracy drastically. This ranking technique requires building several models - language models to predict form and lemma and sequence labeling models to predict non-lexical features such as POS, gender, number etc. Most recently Khalifa, Zalmout, and Habash [27] provided further validation on the hypothesis that in low settings, morphological analyzers help boost the performance of the full morphological disambiguation task.

In chapter 5 we present a solution involving ambiguous morphological analyses in both Hebrew on Turkish which are considered low resource languages. We then apply a single disambiguation model to score each analysis and choose the best one. Our neural MD component is language agnostic and doesn't depend on any language specific resource, and as a result can be easily applied to any language.

3.2.3 Lattice Disambiguation

As previously mentioned, lattice structures encode ambiguity and used in a number of NLP tasks where the input might be ambiguous. MA&D essentially turns the MD problem into a lattice disambiguation problem.

Recently there have a been a number of neural networks that were designed specifically for lattice disambiguation in voice-to-text tasks which might also be applicable to MD. Ladhak, Gandhe, Dreyer, Mathias, Rastrow, and Hoffmeister [31], Sperber, Neubig, Niehues, and Waibel [62] developed a lattice disambiguation architecture which encodes lattice struc-

tures by modifying LSTM cells to keep track of the history of multiple node children. More recently, Sperber, Neubig, Pham, and Waibel [63] applied self-attention layers coupled with reachability masks and positional embedding to efficiently handle lattice inputs. All of these lattice-aware networks were applied on speech recognition tasks, where the segmentation of the input stream refers only to *overt* elements, with no *covert* elements as in morphology. In chapter 5 we describe a lattice disambiguation model that copes with fusional (i.e. non-concatenative) morphological phenomena. Our system is easy to apply and simple to understand — instead of non-trivial modification to the internals of the neural model, we parse and encode the lattice as a sequence to be fed into an existing neural component.

3.3 Morphologically-Aware Datasets

Acknowledging the difference between English and MRLs, in recent years the NLP community has shown increasing interest in parsing typologically different languages, as evident, for instance, by the *SPMRL* initiative (*spmrl.org*) and its successor the *Universal Dependencies* (UD) initiative (*universaldependencies.org*). The UD datasets contain manually annotated treebanks in 82 languages, where the annotation scheme explicitly decompose the raw words into sub-word units, each of which gets assigned its own morphological signature. Both SPMRL [1] formats include annotated information of word segmentation and morphological properties as well as syntax.

Following the development of the UD framework, the *CoNLL-UL* project [41] addresses the need for a universal representation of morphological analysis which on the one hand can capture a range of different alternative morphological analyses of surface words, and on the other hand is compatible with the segmentation and morphological annotation guidelines prescribed for UD treebanks. The project's website[2] provides static lattice files generated for the CoNLL18 shared task [77].

To further facilitate MA for the UD treebanks, Sagot [51] produced a collection of multilingual lexicons in the CoNLL-UL format covering many of the UD languages. These lexicons are available on the project's website[3]

[1] CoNLL-X [10] and CoNLL-U (*https://universaldependencies.org/format.html*

[2] https://github.com/conllul/conllul.github.io

[3] http://atoll.inria.fr/ sagot/UDLexicons.0.2.zip

and can be used by Finite-State-Transducer MA to generate morphological lattices as was also done for the CONLL 2018 UD SHARED TASK .

The Universal Morphology (UniMorph) project[4] is another collaborative effort to improve how NLP handles complex morphology in the world's languages. It contains morphological data annotated in a canonical schema available in many languages which have been shown to improve low resource machine translation [58]. Unfortunately the UniMorph schema currently is not compatible with the UD schema.

3.4 Sequence Labeling Architectures

In recent years, deep learning approaches have obtained very high performance on many NLP tasks, without requiring the traditional task-specific feature engineering.

Recurrent Neural Networks RNN in general, and *Long Short-Term Memory* (LSTM) [24] models in particular, have been proven very successful for various NLP tasks, especially those involving sequential data labeling. These models have the capacity of capturing syntactic and semantic features through word-level embedding [14], and subword features through character level embedding [52]. In addition they can capture linguistic relationships between the elements in the input sequence and be used to output a sequence of labels, one label per input element.

As previously mentioned, for MRLs, the sequence length of the morphological segments can differ from the length of the raw input words. The input length depends on the number of words and the output lengths depends on the number of morphemes which might vary depending on context and this fact should be reflected in the modeling choices. When the output sequence length differs from the input sequence length, RNN-based classifiers are not a natural modeling choice. Another important point to notice is that for *MA&D*, the input to the MD is not a sequence but rather a lattice which is by definition a partial order and so does not naturally lend itself to be encoded by an LSTM-based classifier.

[4] https://unimorph.github.io

3.5 Word Embedding and Language Modeling

Pre-trained word representations have become an integral part of modern neural NLP models. Word embedding is a popular vector representation capturing syntactic and semantic information learned from large collections of unlabeled text.

Word embedding techniques such as Word2Vec [37], and GloVe Pennington et al. [48] generate word representations used in standard components of NLP models. These approaches for learning word vectors suffer from OOV cases, that is they cannot assign vectors to words that were unseen in the training data. Hebrew, with its rich morphology and complex orthography, has a higher probability to encounter OOV words because these word embedding techniques do not try to take into account sub-token information. Unsupervised sub-word based word embedding techniques [9] attempt to remedy this situation but with mixed results; while these models learn orthographic similarities between seen and unseen words, they fail to learn the functions of sub-word units [4, 71]. In this work we propose 2 methods that can produce morphologically aware dense vectors – the first combines the output of a MA to generate morpheme level embedding (chapter 5), while the second approach extracts morphological segment representations from a word-level language model (chapter 6).

A drawback of the word embedding techniques described above is they generate static vectors. Peters, Neumann, Iyyer, Gardner, Clark, Lee, and Zettlemoyer [49] introduced ELMo – a method for generating dynamic contextualized word representation. They introduced a novel approach, learning the internal states of a deep bidirectional LSTM, pre-trained on a large text corpus thus extracting context-sensitive feature representation of words. Following ELMo, Devlin, Chang, Lee, and Toutanova [17] inroduced BERT. They applied a masked language model objective in a bidirectional transformer based architecture and demonstrated significant improvement on a wide range of NLP tasks that require real word knowledge. Liu, Ott, Goyal, Du, Joshi, Chen, Levy, Lewis, Zettlemoyer, and Stoyanov [34] carefully searched and evaluated the impact for various hyper-parameter values used in training BERT models. In chapter 6 we deploy their published set of hyper-parameters for optimizing our BERT model.

Pre-trained language models encode real world information which needs to be extracted before used by real world applications. Howard and Ruder [25] propose an effective and efficient transfer learning method (ULMFiT) that can be applied to any NLP task. They also present several fine-tuning

techniques providing robust learning in many tasks. In chapter 6 we follow some of the steps that are relevant to our use case namely, we pre-train a large language model which we then fine-tune to fit a morphological disambiguation task (referred to as "Target task classifier fine-tuning" in [25]) with gradual unfreezing.

Recently, Minh Van Nguyen and Nguyen [38] developed a multilingual Transformer-based NLP toolkit (Trankit) that combined a pre-trained multilingual model (XML-Roberta) with task specific Adaptors in a plug-and-play framework thus sharing the same language model while learning to perform tokenization, joint morpho-syntactic parsing and named entity recognition. In our work we focus on a language model specifically for Hebrew and use fine-tuning to develop task specific models that performs better on morphological-level tasks in Hebrew as our chapter 6 results indicate.

4 Universal Morpho-syntactic Parsing

The Universal Dependencies (UD) initiative[1] is an international, cross-linguistic and cross-cultural initiative aimed at providing annotated data sets for over 80 languages under a unified, harmonized, morpho-syntactic annotation scheme.

The UD scheme [46] adheres to two main principles: (i) there is a single set of POS tags, morphological properties, and dependency labels for all treebanks, and their annotation obeys a single set of annotation principles, and (ii) the text is represented in a two-level representation, clearly mapping the written space-delimited source tokens to the (morpho)syntactic words which participate in the dependency tree.

The CoNLL 2018 UD Shared Task is a multilingual parsing evaluation campaign wherein, contrary to previous shared tasks corpora are provided with raw text, and the end goal is to provide a complete morpho-syntactic representation, including automatically resolving all of the token-word discrepancies. Contrary to the previous SPMRL shared tasks [53, 54] the output of all systems obeys a single annotation scheme, allowing for reliable cross-system and cross-language evaluation.

We investigate the contribution of an external lexicon and a standalone morphological component, and show that inclusion of such lexica can lead to above 10% LAS improvement in dependency parsing.

Our investigation demonstrates the importance of sharing not only syntactic treebanks but also lexical resources among the UD community, and we propose the UD-compatible CoNLL-UL standard [42] for sharing broad-coverage lexical resources in the next UD shared tasks, and in general.

[1] `universaldependencies.org`

4.1 CoNLL 2018 UD Shared Task Submission

The parsing system presented for this task is based on *yap — yet another parser*, a transition-based parsing system that relies on the formal framework of Zhang and Clark [78], an efficient computational framework designed for structure prediction and based on the generalized perceptron for learning and beam search for decoding. This section briefly describes the formal settings and specific models available via *yap*.[2]

4.1.1 Formal Settings

Formally, a transition system is a quadruple (C, T, c_s, C_t) where C is a set of configurations, T a set of transitions between the elements of C, c_s an initialization function, and $C_t \subset C$ a set of terminal configurations. A transition sequence $y = t_n(t_{n-1}(...t_1(c_s(x))))$ for an input x starts with an initial configuration $c_s(x)$ and results in a terminal configuration $c_n \in C_t$.

In order to determine *which* transition $t \in T$ to apply given a configuration $c \in C$, we define a model that learns to predict the transition that would be chosen by an oracle function $O : C \rightarrow T$, which has access to the gold output.

We employ an objective function

$$F(x) = argmax_{y \in GEN(x)} Score(y)$$

which scores output candidates (transition sequence in $GEN(x)$) such that the most plausible sequence of transitions is the one that most closely resembles the one generated by an oracle.

To compute $Score(y)$, we map y to a global feature vector $\Phi(y) = \{\phi_i(y)\}$ where each feature $\phi_i(y)$ is a count of occurrences of a pattern defined by a feature function. Given this vector, $Score(y)$ is calculated as the dot product of $\Phi(y)$ and a weights vector $\vec{\omega}$:

$$Score(y) = \Phi(y) \cdot \vec{\omega} = \sum_{c_j \in y} \sum_i \omega_i \phi_i(c_j)$$

Following Zhang and Clark [78], we learn the weights vector $\vec{\omega}$ via the *generalized perceptron*, using the *early-update* averaged variant of Collins and

[2] https://github.com/OnlpLab/yap

Roark [13]. For decoding, the framework uses the *beam search* algorithm, which helps mitigate otherwise irrecoverable errors in the transition sequence.

4.1.2 Morphological Analysis

The input to the morphological disambiguation (MD) component, and to the *yap* parsing system in general, is a lattice L representing all of the morphological analysis alternatives of k surface tokens of the input stream $x = x_1, ..., x_k$, such that each $L_i = MA(x_i)$ is generated by a *morphological analysis* (MA) component, the lattice concatenate the lattices for the whole input sentence x. Each *lattice-arc* in L has a *morpho-syntactic representation* (MSR) defined as $m = (b, e, f, t, g)$, with b and e marking the start and end nodes of m in L, f a form, t a universal part-of-speech tag, and g a set of attribute=value universal features. These *lattice-arc* correspond to potential nodes in the intended dependency tree.

4.1.3 Morphological Disambiguation

The *morphological disambiguation* (MD) component of our parser is based on More and Tsarfaty [40], modified to accommodate UD POS tags and morphological features. We provide here a brief exposition of the transition system, as shall be needed for our later analysis, and refer the reader to the original paper for an in-depth discussion.

A configuration for MD $C_{MD} = (L, n, i, M)$ consists of a lattice L, an index n representing a node in L, an index i s.t. $0 \leq i < k$ representing a specific token's lattice, and a set of disambiguated morphemes M.

The initial configuration function is defined as $c_s(x) = (L, bottom(L), 0, \emptyset)$ where $L = MA(x_1) \circ ... \circ MA(x_k)$, and $n = bottom(L)$, the bottom of the lattice. A configuration is terminal when $n = top(L)$ *and* $i = k$.

To traverse the lattice and disambiguate the input, we define an open set of transitions using the MD_s transition template:

$$MD_s : (L, p, i, M) \rightarrow (L, q, i, M \cup \{m\})$$

Where $p = b$, $q = e$, and s relates the transition to the disambiguated morpheme m using a parameterized delexicalization $s = DLEX_{oc}(m)$:

$$f(x) = \begin{cases} (\_, \_, \_, t, g) & t \in OC \\ (\_, \_, f, t, g) & otherwise \end{cases} \tag{4.1}$$

In words, $DLEX$ projects a morpheme either with or without its form depending on whether or not the POS tag is an open-class with respect to the form. For UD, we define:

$$OC = \{ADJ,AUX,ADV,PUNCT,NUM,INTJ,NOUN,PROPN,VERB\}$$

We use the parametric model of More and Tsarfaty [40] to score the transitions at each step. Since lattices may have paths of different length and we use beam search for decoding, the problem of variable-length transition sequences arises. We follow More and Tsarfaty [40], using $ENDTOKEN$ transition to mitigate the biases induced by variable-length sequences.

4.1.4 Syntactic Disambiguation

A syntactic configuration is a triplet $C_{DEP} = (\sigma, \beta, A)$ where σ is a stack, β is a buffer, and A a set of labeled arcs. For dependency parsing, we use a specific variant of Arc Eager that we call Arc Zeager, that was first presented by [79]. The differences between plain Arc-Eager and the Arc-Zeager variant are detailed in the appendix.

The features defined for the parametric model also follows the definition of non-local features by Zhang and Nivre [79], with one difference: we created one version of each feature with a morphological signature (all feature values of the relevant node) and one without. this allows to capture phenomena like agreement.

4.1.5 Joint Morpho-Syntactic Processing

Tsarfaty [67] hypothesised that *joint* morpho-syntactic parsing, where morphological information may assist syntactic disambiguation and *vice versa*, may be better suited to address the tight interaction between morphology and syntax. Although not applied in this task, we point out that, given the standalone morphological and syntactic disambiguation described above, it is possible to embed the two models into a single *joint morpho-syntactic* transition system with a "router" that decides which of the transition systems to apply in a given configuration, and train the morpho-syntactic model to maximize a single objective function.

We implement such joint parser in yap but we have not used it in the task, and we thus leave its description out of this exposition. For further discussion and experiments with the syntactic and joint morpho-syntactic variants in yap we refer the reader to More et al. [43].

Model	Lexicon	Sentence Segmentation	Morphological Disambiguation	Parser	Results on dev LAS / MLAS / BLEX
UDPipe full	–	UDPipe	UDPipe	UDPipe	61.95 / 49.28 / 51.45
yap DEP	–	UDPipe	UDPipe	*yap*	59.19 / 49.19 / 33.75
yap full	Basline	UDPipe	*yap*	*yap*	52.25 / 37.85 / 29.59
	HebLex				60.94 / 39.49 / 33.85
	HebLex-Infused				71.39 / 61.42 / 41.86
yap GOLD	–	Gold	Gold	*yap*	79.33 / 72.56 / 47.62

Table 4.1: The contribution of lexical resources: analysis of the case for Modern Hebrew

4.2 A Detailed Analysis for Modern Hebrew

It is well known, as well as observed in this particular task, that *morphologically rich languages* are most challenging to parse in the raw to dependencies parsing scenarios. This is because the initial automatic segmentation and morphological disambiguation may contain irrecoverable errors which will undermine parsing performance.

In order to investigate the errors of our parser we took a particular MRL that is known to be hard to parse (Modern Hebrew, ranked 58 in the LAS ranking, with basline 58.73 accuracy) and contrasted the Baseline UDPipe results with the results of our parser, with and without the use of external lexical and morphological resources.

Table 4.1 lists the results of the different parsing models on our dev set. In all of the parsing scenarios, we used UDPipe's built in sentence segmentation, to make sure we parse the exact same sentences. We then contrasted UDPipe's full pipeline with the *yap* output for different morphological settings. We used the Hebrew UD train set for training and the Hebrew UD dev set for analyzing the empirical results.

Initially, we parsed the dev set with the same system we used for the shared task, namely, *yap* dependency parser which parses the morphologically diambiguated output by UDPipe (yap DEP). Here we see that yap DEP results (59.19) are lower than the full UDPipe pipeline (61.95).

We then moved on to experiment with yap's complete pipeline, including a data-driven morphological analyzer (MA) to produce input lattices, transition-based morphological disambiguation and transition-based parsing. The results now dropped relative to the UDPipe baseline and relative to our own yap DEP system, from 59.19 to 52.25 LAS. Now, interestingly, when we replace the baseline data-driven MA learned from the treebank

alone with an MA backed with an external broad-coverage lexicon called HebLex (adapted from Adler and Elhadad [1]), the LAS results arrive at 60.94 LAS, outperforming the results obtained by yap DEP (UDPipe morphology with yap dependencies) and close much of the gap with the UDPipe full model. This suggests that much of the parser error stems from missing lexical knowledge concerning the morphologically rich and ambiguous word forms, rather than parser errors.

Finally, we simulated an ideal morphological lattices, by artificially infusing the path that indicates the correct disambiguation into the HebLex lattices in case it has been missing. Note that we still provide an ambiguous input signal, with many possible morphological analyses, only now we guarantee that the correct analysis exists in the lattice. For this setting, we see a significant improvement in LAS, obtaining 71.39 (much beyond the baseline) without changing any of the parsing algorithms involved. So, for morphologically rich and ambiguous languages it appears that lexical coverage is a major factor affecting task performance, especially in the resource scarce case.

Note that the upper-bound of our parser, when given a completely disambiguated morphological input stream, provides LAS of 79.33, which is a few points above the best system (Stanford) in the raw-to-dependencies scenario.

4.3 Summary

In this chapter we present our CoNLL 2018 UD Shared Task submission. Our submitted system assumed UDpipe up to and including morphological disambiguation, and employed a transition based parsing model to successfully parse 81 languages in the UDv2 set, with the average LAS of 58.35, ranked 22 among the shared task participants.

A detailed post-task investigation of the performance that we conducted on Modern Hebrew, has shown that for the MRL case much of the parser errors may be attributed to incomplete morphological analyses or a complete lack thereof for the source tokens in the input stream.

Next we turn to a deep learning solution for morphological lattice dismabiguation, replacing the hand-crafted feature model with neural network based feature extraction mechanisms while still injecting symbolic linguistic knowledge provided by MA component.

5 Pointer Network Based MD

In the previous chapter we put forward a pre-neural parsing framework composed of two transition systems that can be used stand alone, separately performing MD and dependency parsing, as well as combine into a morpho-syntactic parser jointly choosing most likely sequence of morphemes and dependency relations. Remarkably, the input to the parser is an ambiguous morphological lattice, which both standalone as well as joint components disambiguate by choosing the most likely lattice path. By doing so the problem is framed as a *Morphological Analysis and Disambiguation* (MA&D) task. Several methods for MD in the pre-neural era [1, 6, 22, 32] applied a similar strategy, using weighted finite-state machines to first unravel the possible morphological decompositions, and classic machine learning models to select the most likely decomposition.

We now propose a similar architecture performing MA&D, that enjoys the power of end-to-end neural modeling while maintaining access to morphemes. Every raw token in the input sequence first goes through *Morphological Analysis* (MA) that exposes *all* of its possible morphological decompositions as a lattice (see Figure 2.1). This morphological lattice is then passed to the MD component, based on a *Pointer Network*, which selects a sequence of most likely arcs in the context of the sentence being processed. Since every lattice arc contains rich information that is made available by the MA — namely, segmentation boundaries, lemma, Part-of-Speech tag and a set of morphological features — this MA&D framework can jointly predict rich morphological layers.

Based on this architecture, we perform *segmentation and tagging* and apply it to two MRLs, Hebrew and Turkish. In *realistic* circumstances, the lexical *coverage* of the lattice may be partial, and we report MD results in both *ideal* and *realistic* scenarios. Our results on the Hebrew and Turkish UD treebanks show state-of-the-art performance for complete morphological lattices, and on the Hebrew SPMRL treebank we outperform all previous results in *realistic* scenarios. Our MA&D solution is generic and can be

applied to any language, e.g., assuming MA components as provided in More et al. [41]. In addition, our proposed architecture is suitable for any other task that encodes information in a lattice towards further disambiguation.

5.1 Proposed Method

Given an input lattice, we frame MD as a lattice disambiguation task. We modify the *lattice representation* and feed it to a *pointer network* (PtrNet) architecture. The key idea, in a nutshell, is to linearize the lattice into a sequence of partially-ordered analyses, and feed this partial order sequence to a pointer network. For each token, the network will learn to *point to* (select) the most likely analysis, preserving the linear constraints captured in the lattice structure. In other words, the input to the MD is a sequence of partially ordered morphological analyses and the goal is to pick one analysis per word. The PtrNet architecture by design captures this goal, and to the best of our knowledge we are the first to apply it in on the task of Morphological Disambiguation. Hence we propose *PtrNetMD* as a neural disambiguation model in a MA&D framework.

Lattice Serialization The MA lattice, as previously defined, is a graph representation of the ambiguous morphological analyses for each word in the input sentence. However, PtrNet takes a sequence as the input therefore our first step involves transforming the lattice into a sequence which can then get fed to the PtrNet. Given a lattice we serialize it by going over each token and listing all of its analyses. The linearization function maps a sequence of n tokens into a sequence of m analyses while preserving the partial order of the tokens, and where m is the sum of all token analyses. That is, for input tokens $t_1, ...t_n$, let a^i_j denote the i'th analysis of the j'th token. Then the following holds, such that $\sum_{i=1}^{n} k_i = m$.

$$linearize(t_1, t_2, t_3, ..., t_n) = a^1_1, ..., a^{k_1}_1, a^1_2, ..., a^{k_2}_2, ..., a^1_n, ..., a^{k_n}_n \quad (5.1)$$

An analysis a^i_j is expressed as a list of morphemes where each morpheme is represented as a tuple of morphological properties. Both the SPMRL and UD scheme specify four properties *Form, Lemma, POS Tag, Morphological Features*. For example, (3) is an analysis composed of three morphemes:

$$a_j^i := [\ \ (form_1, lemma^1, tag_1, features_1),\\ (form_2, lemma_2, tag_2, features_2),\\ (form_3, lemma_3, tag_3, features_3)]$$

Morphological Analysis Embedding Each analysis in the linearized input sequence is embedded into a vector representation. To that end we design a Morphological Embedding layer which can be thought of as an interface between the *symbolic MA* and the *neural MD*. Figure 5.1 describes the encoding of a single morphological analysis into an embedded vector: Each property is embedded and averaged across all the morphemes in a single analysis, and all of the averaged embedded properties are concatenated to form a single embedded vector of a fixed size.

Sequence to Sequence Seq2Seq networks are designed to produce outputs which may differ from the inputs in length and vocabulary [66].
Seq2Seq is composed of an encoder and a decoder. The encoder consumes and encodes the entire (embedded) input sequence. Then, the decoder is fed the entire encoded input representation and step by step produces discrete outputs which are fed back as input to the next decoding step [11].

Pointer Networks PtrNet are designed as a special case of Seq2Seq networks [72] with an additional copy attention layer. *Attention* was presented by Bahdanau, Cho, and Bengio [5] that reads as a natural extension of their previous work on the Encoder-Decoder model. It enables the network to focus on specific elements from the input sequence at each decoding step thus helping it identify long sequence relationships. *Copy* attention is a special attention case where the weights determine which input element the decoder's state is most aligned with, allowing the network to output an indices pointing back at the input sequence.

Morphological Analysis Disambiguation The sequence of embedded analyses is encoded and then in a step by step decoding process the analyses weights are computed conditioned on the entire (ambiguous) encoded context. The entire MA&D process is depicted in Figure 5.2 - given the linearized lattice as input, individual analyses in the lattice can be *pointed*, selected and copied into the output sequence, while respecting the lattice ordering constraints. The full output sequence contains a list of indices,

one per token, pointing to the selected analyses from the input lattice (Fig. 5.1).

5.2 Experimental Setup

The Data The PtrNetMD architecture we propose does not depend on any specific definition of morphological signature. To showcase this, we experiment with data from two different languages and two different annotation schemes. We use the Universal Dependencies v2.2 dataset [46] from the CoNLL 2018 UD SHARED TASK .[1] In addition we download the corresponding lattice files of each treebank from the CoNLL-UL project.[2] Since our approach is sensitive to the lexical coverage of the MA lattices, we focus on the Hebrew (he_htb) and Turkish (tr_imst) treebanks. Unlike other languages in the shared task, Hebrew and Turkish provided lattice files generated by broad-coverage analyzers (HEBLEX and TRMorph2).[3] For comparability with previous work on Modern Hebrew, we also train and test our model on the Hebrew SPMRL treebank standard split.[4]

Lattice Embedding We use pre-trained FastText [9] models to embed the lexical morphological properties - forms and lemmas. FastText models generate vectors for any word using character ngrams, thus handling Out-of-Vocabulary forms and lemmas. For non-lexical properties we instantiate and train from scratch two embedding modules - one embedding module for POS tags and one for all other grammatical features. Together, these 4 embedded properties are combined to produce a single morphological analysis vector.

Lattice Encoding The above-mentioned morphological embedding layer turns the input analysis sequence into an embedded sequence. The partially ordered sequence of embedded analyses is fed to a BiLSTM encoder

[1]The treebanks from the shared task are available at lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-2837

[2]https://conllul.github.io/

[3]The Arabic (ar_padt) Calima-Star lattice files exhibited a number of incompatibilities with the corresponding gold UD annotations and therefore cannot be considered

[4]The treebank is publicly available as open source at https://github.com/OnlpLab/HebrewResources/tree/master/HebrewTreebank

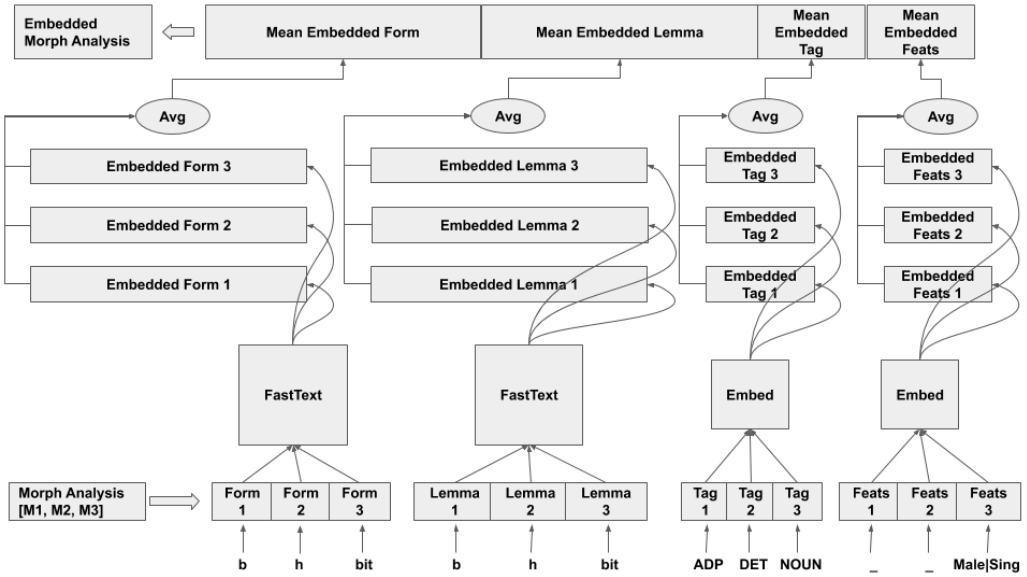

Figure 5.1: Morphological Embedding Layer Architecture. An analysis composed of 3 morphemes is transformed into a single embedded vector.

layer thus encoding the entire embedded lattice. Next a step-by-step decoding process begins in which a LSTM layer decodes the current analysis and an *Attention* mechanism scores the alignment between each of the relevant encoded word-analyses and the analysis currently being decoded. Our copy attention module is the *global dot-product* of Luong, Pham, and Manning [36] using *masking* mechanism to make sure each decoding step is focused *only* on the corresponding input word analyses (in figure 5.2 the masks are represented by the grouped arrows pointing from the decoder back to the encoded sequence). To choose the most likely analysis, the decoder chooses the highest score provided by the *Attention* component.

5.2.1 Baseline Models

Our experiments are set up to compare our PtrNetMD with various baseline models on different MD tasks. As previously discussed, MD has been considered in different ways defined by the type of output generated by the MD model. In this section we consider the four different MD tasks:

(1) *Morpheme Segmentation* Each word is segmented into surface forms, one form per morpheme.

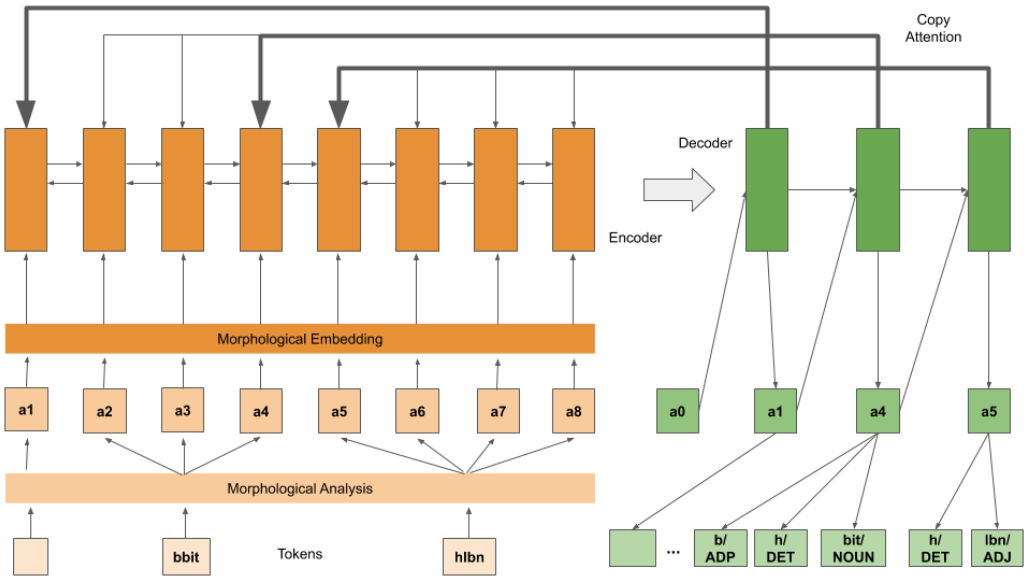

Figure 5.2: Our Proposed MA&D Architecture. A sequence of tokens is transformed into a sequence of analyses while preserving the token order. The sequence of analyses is embedded and fed into an encoder. Then at each decoding step the entire encoded representation along with the current decoded state are used as input to an attention layer, and the attention weights are used to choose an element from the input sequence.

(2) *Word Multi-tagging* The output sequence contains a single multi-tag per word.

(3) *Morpheme Tagging* The output sequence contains a single POS tag per morpheme.

(4) *Morpheme Segmentation & Tagging* Each word is segmented into surface forms, one form per morpheme, and each segment is assigned a single POS tag.

To compare our PtrNetMD architecture to existing modeling solutions we consider the following baseline end-to-end (E2E) neural MD models.

Segmentation & Tagging Pipeline Straka and Straková [65] approach the MD problem as a two-phased pipeline, first performing segmentation of the input words followed by sequence tagging on the morpheme sequence. This approach mimics the way English POS tagging is performed, with the exception that the tagging is done on the morphological forms as

opposed to directly on input words. On the one hand it is straight forward to design such disambiguation pipeline and apply it to many languages. On the other hand, POS tagging accuracy suffers from *error propagation* from the earlier segmentation. To gauge the effect of error propagation we compare tagging accuracy provided by gold (oracle) segments as opposed to predicted segments (using UDPipe), for Turkish, Hebrew, Arabic and English showing the drop in the accuracy of MRLs in comparison to English. In order to avoid error propagation we implement 2 baseline neural *tagging-only* models, each aimed at a different granularity level of the output as described next.

Word sequence multi-tagging The first tagging baseline predicts word-level *multi-tag* labels (as opposed to single tag prediction on the morpheme-level). That is, we assign a single complex label composed of multiple POS tags to each raw word. We define a multi-tag as a concatenated list of basic tags, one per morpheme. In training, a word such as *bbit*, which is gold-segmented into the basic tag sequence *b/IN, h/DET, bit/NOUN*, is assigned a single multi-tag *bbit/IN-DET-NOUN*. Notice that the output space is sparse and most complex tags are seen only a few times (or not at all) during training. Also note, that once predicted the multi-tags can be decomposed into the single tags that make up the multi-tag parts thus providing a way to compare the multi-tag output with morpheme tagging sequences (such as produced by PtrNetMD as well as our next baseline described below). We exploit the fact that the input and output sequence length are identical and use the output of a BiLSTM layer before assigning a multi-tag to each word. We use FastText for embedding the input word sequence. In addition, in order to inform the model about sub-token information, we combine each embedded token with a vector encoding the sequence of characters in the token, as suggested by Ling, Dyer, Black, Trancoso, Fermandez, Amir, Marujo, and Luís [33]. A notable disadvantage of this model compared to the pipeline, and the proposed PtrNetMD model, is that it does *not* provide any information concerning segmentation boundaries.

Morpheme sequence tagging Our multi-tagging model has the drawback of operating on a large and non-compositional output-labels space. So, it cannot assign previously unseen tag compositions to previously unseen tokens. To overcome this, we implement a morpheme-level baseline in which the input consists of raw tokens but the output is a tag sequence, of a possibly different length, predicted (decoded) one at a time by

MD Task	Example	E2E Model
Morph Seg	b h bit h lbn	Language Model (form/character)
Word Multi-Tag	bbit/ADP_DET_NOUN hlbn/DET_ADJ	Sequence Labeling - LSTM
Morph Tag	bbit/[ADP,DET,NOUN] hlbn/[DET,ADJ]	Sequence Labeling - Seq2Seq
Morph Seg&Tag	b/ADP/_ h/DET/_ bit/NOUN/M,S h/DET/_ lbn/ADJ/M,S	Language Model + Sequence Labeling

Table 5.1: MD tasks implemented E2E. MD task definition varies from segmentation-only, to tagging-only, to joint segmentation and tagging. Each task is realized with E2E model architecture that fit its task definition. *Segmentation* is considered a language modeling task realized on the form and character level. *Word tagging* is most naturally realized with LSTM where each input word is assigned a single output label. *Morpheme tagging* may produce an output sequence which differs in length from the raw word input sequence and therefore is realized with a Seq2Seq network. *Morph segmentation & tagging* is easily realized as a 2-step pipeline – segmentation followed by labeling each segment with a POS tag.

sequence-to-sequence (Seq2Seq) model. Note that the output tag sequence can vary in length, potentially predicting the number of morphological segments in each token. Hence in this case we actually use a Seq2Seq model which is able to predict output labels the differ in length from the length of the input word sequence. We again use the combined word and character embedding layer as described in the previous paragraph. This model too, does not provide explicit segmentation boundaries. Table 5.1 show examples as well as specifies the E2E model type used for each task.

In addition to E2E baselines applied on the 4 MD tasks, we also compare our PtrNetMD with state-of-the-art results on the Turkish and Hebrew portions of the UD treebanks. Finally we compare our MA&D solution head to head with the state-of-the-art Hebrew MA&D applied on the SPMRL treebank.

5.2.2 Evaluation

Aligned Segment The CoNLL 2018 UD Shared Task evaluation campaign[5] reports scores for segmentation and POS tagging[6] for all participating languages. The shared task provides an evaluation script producing various levels of F1 scores, based on aligned token-level segments. Since the focus of the shared task was to reflect word segmentation and relations between segmented words, the script discards unmatched word segments, so in effect the POS tagging scores are in fact *joint segmentation-and-tagging*. With this script we evaluate the performance of all baseline models in Turkish, Hebrew, Arabic and English and use these results to compare aligned segment tagging scores between oracle (gold) segmentation and realistic (predicted) segmentation in a pipeline model. In addition, since our PtrNetMD jointly predicts both segments and tags, we can evaluate our proposed PtrNetMD against the shared task leaders for Hebrew and Turkish.

Aligned Multi-Set The shared task evaluation script computes aligned segment scores by finding the *longest common subsequence* (LCS) of the gold and predicted word-segments, for every token in the raw input. In addition to the shared task scores, we compute F1 scores similar to the aforementioned with a slight but important difference. Word counts are based on multi-set intersections of the gold and predicted labels. In general, a multi-set (mset) is a modification of the set concept allowing multiple instances of its set items. In our case we use a multi-set to count intersections of morphological signatures in each word. To illustrate the difference between *aligned segment* and *aligned mset*, let us take for example the gold segmented tag sequence: *b/IN, h/DET, bit/NOUN* and the predicted segmented tag sequence *b/IN, bit/NOUN*. According to *aligned segment*, the first segment (*b/IN*) is aligned and counted as a true positive, the second segment however is considered as a false positive (*bit/NOUN*) and false negative (*h/DET*) while the third gold segment is also counted as a false negative (*bit/NOUN*). The *aligned mset* on the other hand is based on set difference which acknowledges the possible undercover of covert morphemes by the disambiguator. In this case both *b/IN* and *bit/NOUN* exist in the gold and predicted sets and counted as true positives, while *h/DET* is mismatched and counted as a false negative. In both cases the total

[5] https://universaldependencies.org/conll18/results.html

[6] respectively referred to as 'Segmented Words' and 'UPOS' in the CoNLL18 evaluation script

counts across the entire datasets are then incremented accordingly and finally used for computing Precision, Recall and F1.

Formally, *aligned mset* F1 metric is calculated as follows: For each token we first create a multi-set based on the morphological signatures (morphological signature is defined by the properties of interest: Segments only, POS tag only, joint segment and tag, etc.) for both the predicted (Pred) and gold (Gold) morphemes:

(4) $Pred_{token} = \uplus(p_1, p_2, ..., p_k)$
$Gold_{token} = \uplus(g_1, g_2, ..., g_l)$
$\uplus$: multi-set addition operator

We then calculate the token level true and false positives (TP, FP) as well as false negatives (FN):

(5) $TP_{token} = Pred_{token} \cap Gold_{token}$
$FP_{token} = Pred_{token} - Gold_{token}$
$FN_{token} = Gold_{token} - Pred_{token}$

Finally we add up the token counts over the entire dataset to produce the F1 metric:

(6) $TP_{total} = \sum(TP_{token})$
$FP_{total} = \sum(FP_{token})$
$FN_{total} = \sum(FN_{token})$
$Precision = TP_{total}/(TP_{total} + FP_{total})$
$Recall = TP_{total}/(TP_{total} + FN_{total})$
$F1 = \frac{2 \times Precision \times Recall)}{Precision + Recall}$

Having morphemes available even if out of order or partially, has merit to downstream tasks that consume and further process them. *Aligned mset* accounts for this quality. Furthermore, both our *multi-tagging* and *sequence-to-sequence tagging* baseline models produce a tag sequence *without* segmentation boundaries, and *aligned mset* can be used to compare them against our PtrNetMD model. Finally since this computation was also used by More et al. [43] we are able to compare our results to their non-neural MA&D framework applied to the Hebrew SPRML treebank, which is so far considered the current state-of-the-art for Hebrew segmentation and tagging.

Ideal vs Realistic Analysis Scenarios Following More et al. [43] we distinguish between two evaluation scenarios. An **Infused** scenario is an idealised scenario in which the input lattice to our model has complete lexical coverage, and is guaranteed to include the correct analysis as one of its many internal paths. An **Uninfused** scenario is a realistic case in which the lexical coverage might be partial, and might lack certain gold analyses. [7]

5.3 Results

CoNLL18 UD Shared Task Table 5.2 shows *aligned segment* F1 scores for joint segmentation-and-tagging on four languages that exhibit different degrees of morphological richness. The top two models are variants of the UDPipe pipeline system — UDPipe *Oracle* scores were obtained by running the UDPipe tagger on gold segments, and UDPipe *Predicted* scores were obtained by segmenting the raw text first and then tagging the predicted segments.[8]

The top two rows in Table 5.2 allow us to gauge the effect of error propagation for different languages, as reflected in the performance difference between tagging gold (Oracle) segments and tagging predicted segments. These results are remarkable — in an *ideal* (gold-oracle) scenario there is no significant difference in the tagging accuracy between English and MRLs, but in the *realistic* scenarios where segmentation precedes tagging, the difference is large.

The bottom three models in Table 5.2 report the leading scores from the CoNLL 2018 UD Shared Task as well as our PtrNetMD results. The PtrNetMD achieves state-of-the-art results for joint segmentation-tagging, on both Hebrew and Turkish, in infused settings. Moreover, the PtrNetMD ties state-of-the-art on the Hebrew UD treebank even with uninfused (realistic) lattices with partial lexical coverage.

In Table 5.3 we see *aligned segment* F1 scores for segmentation-only on the same four languages. The results clearly indicate that segmenting Hebrew is harder than segmenting Arabic, which is harder to segment than Turk-

[7] Like More et al. [43] we refer to the idealized scenario as **infused** since we make sure the gold annotation is present in each token lattice or else we manually infuse it. The realistic scenario is thus referred to as **uninfused**.

[8] The UDPipe Predicted model served as the baseline model for the CoNLL18 UD Shared Task participants.

	English	Turkish	Arabic	Hebrew
UDPipe Oracle	94.62	93.24	95.30	95.13
UDPipe Predicted	93.62	91.64	89.34	80.87
Shared Task Leader	95.94	94.78	93.63	91.36
PtrNetMD Infused		**96.6**		**94.41**
PtrNetMD Uninfused		89.54		91.3

Table 5.2: Joint Segmentation-and-Tagging F1, Aligned Segment, CoNLL 2018 UD Shared Task Test Set. Top two rows are pipeline baseline. Bottom three rows are PtrNetMD compared to shared task leaders.

	English	Turkish	Arabic	Hebrew
UDPipe Oracle	100.00	100.00	100.00	100.00
UDPipe Predicted	99.03	97.92	93.71	85.16
Shared Task Leader	99.26	97.92	96.81	93.98
PtrNetMD Infused		**99.41**		**96.36**
PtrNetMD Uninfused		97.78		94.74

Table 5.3: Segmentation-only F1, Aligned Segment, CoNLL 2018 UD Shared Task Test Set. Top two rows are pipeline baseline. Bottom three rows are PtrNetMD compared to shared task leaders.

ish, and English requires essentially no segmentation. As in Table 5.2, we see similar behavior comparing PtrNetMD to shared task leaders on the segmentation task — PtrNetMD with infused lattices outperforms the shared-task leader on Turkish, and it outperforms the shared-task leader in both infused and uninfused scenarios on Hebrew.

There are two possible explanations for prediction errors in uninfused scenarios. Either the correct analysis (gold annotation) is part of the lattice and the model makes a wrong selection, or, the correct analysis is not in the lattice. Acknowledging the notable gap in Table 5.2 between PtrNetMD infused and uninfused scores on Turkish, we compared the number of prediction errors with the number of missing analyses in the uninfused lattices. Out of 1028 wrong predictions, in 652 the correct analysis was missing from the lattice, so the MD had in fact no chance to make the right choice. In other words, 60% of MD errors can be accounted by lacking coverage of the MA component. Interestingly there is a 60% error reduction when moving to infused lattices, suggesting that the missing analyses account for the difference between infused and uninfused scores. The same holds for Hebrew as well: out of 850 wrong predictions, in 330 cases the

	Turkish	Arabic	Hebrew
Word Multi-Tag	92.57	94.2	93.82
Morpheme Tag	92.77	95.05	93.75
PtrNetMD infused	**96.76**		**96.40**
PtrNetMD uninfused	90.01		94.02

Table 5.4: Tagging F1, Aligned MSet, CoNLL 2018 UD Shared Task Test Set

correct analysis was missing in the lattice which is also very close to the difference between the infused and uninfused scores. In both Turkish and Hebrew, PtrNetMD is very likely to choose the correct analysis when one is available in the input lattice.

Another insight into the coverage difference between the Turkish and Hebrew lattices is revealed by the fact that the average number analyses per token on the Turkish is 2.6 while the Hebrew average number analyses per token is 10. So even with larger levels of ambiguity present in the Hebrew lattices, PtrNetMD benefits from the broad coverage offered by the MA supporting our hypothesis that PtrNetMD has the capacity to identify the correct analysis as long as it is present in the ambiguous lattice.

Table 5.4 contains the *aligned mset* scores of two baselines — word multi-tag and morpheme tag — as well as the PtrNetMD infused and uninfused settings (since both baselines don't predict segments they are inapplicable for *aligned segment* evaluation). In both Turkish and Hebrew, the infused PtrNetMD performs much better than E2E tagging models. The Hebrew PtrNetMD even outperforms both baselines in uninfused circumstances. The high infused scores on both treebanks suggest that the PtrNetMD model is more than capable to select the correct analysis as long as one is present in the lattice. The difference between infused and uninfused scores highlight the importance of generating full coverage lattices by the MA component.

These results suggests that a good strategy for MA&D would be to provide high coverage MA component which while generating large lattices covering many possibilities, they include with high probability the correct analysis. Based on the above results, our PtrNetMD has the capacity to discriminate and choose the correct analysis as long as one is present in the lattice.

	Dev-Inf	Dev-Uninf	Test-Inf	Test-Uninf
MoreMD	94.09	90.83	92.92	87.53
MoreMD-DEP	**95.49**	92.36	**93.92**	89.08
PtrNetMD	95.09	**93.9**	93.51	**90.49**

Table 5.5: Joint Segmentation-and-Tagging F1, Aligned MSet, Hebrew SPMRL treebank

SPMRL Hebrew Treebank To put our results in context, Table 5.5 compares PtrNetMD on the Hebrew SPMRL treebank with the state of the art results of More et al. [43], who used the same *aligned mset* scores for performing joint segmentation-and-tagging evaluation. The MoreMD lattice disambiguation approach is similar to our PtrNetMD, albeit non-neural, using feature-based structured perceptron for disambiguation. As can be seen in the table, the PtrNetMD outperforms the MoreMD model in all settings. The MoreMD-DEP model, jointly performs MD and dependency parsing, taking advantage of additional syntactic information that is predicted jointly with the segmentation and tags. The syntactic information contributes to the MD performance as can be seen in the Infused columns. However, our PtrNetMD handles incomplete morphological information better than MoreMD-DEP, as can be seen in the Uninfused columns.

5.4 Summary and Discussion

In this chapter we addressed the challenge of morphological disambiguation for MRLs. We design a general framework that consumes lattice files and output a sequence of disambiguated morphemes, each containing the segmentation boundary, lemma, part-of-speech tag and morphological features. Our solution is language agnostic and we apply it on two different languages and two different annotation schemes. We show that access to symbolic morphological information aids the neural disambiguation model, compared to end-to-end strong baselines that only have access to the raw tokens.

We empirically evaluate our model using two evaluation methods. The CoNLL 2018 UD Shared Task evaluation, and a multi-set evaluation, which is a more informative metric for downstream tasks that operate directly on morpheme sequences. In ideal scenarios, where full lexical coverage is assumed, our model outperformed the shared task leaders of the word segmentation task as well as the joint segmentation and tagging task,

in both Turkish and Hebrew. Furthermore, we match the leading joint segmentation and tagging scores in realistic scenario with only partial lexical coverage on Hebrew. We further show superior performance of our model compared to previous models on the Hebrew SPMRL treebank.

Our proposal shows great promise in performing lattice disambiguation and combined with a broad-coverage MA component can offer state-of-the-art morphological segmentation and tagging. This MA&D framework, however, comes at a price — the performance of the system depends on a language-specific MA component. Building high quality MA usually entails many hours of manual work that covers all the words in the language and specifying for each one all possible morphological breakdowns. In many cases, this cost is too high resulting in low quality MA which affect the overall quality of the MA&D.

In the next chapter we take a different approach saving us from relying on manually built linguistic resources. Instead we will introduce an alternative linguistic resource in the form of a language model, trained in an unsupervised settings using enormous amounts of texts. We will then exploit the linguistic information encoded in this language model by including it as a component in a morphological disambiguation framework that performs disambiguation E2E, without the intermediate lattice representation generated by an MA.

6 Pre-trained Language Model Based MD

Contextualized representations, provided by models such as BERT [16] and RoBERTa [34], has been shown in recent years to be critical for obtaining state-of-the-art performance on a wide range of Natural Language Processing (NLP) tasks — such as syntactic and semantic parsing, question answering, natural language inference, text summarization, natural language generation, and more. Contextualized word embedding is obtained by pre-training a large language model on massive quantities of unlabeled data, aiming to maximize simple yet effective language modeling objectives such as *masked word prediction* and *next sentence prediction*.

While advances for English using such models are unprecedented, reported results in Hebrew using BERT-based models are far from impressive. Specifically, the BERT-based Hebrew section of multilingual-BERT [16] (henceforth, mBERT), did not provide a similar boost in performance to what is observed for the English section of mBERT. In fact, for several reported tasks, mBERT results are on a par with pre-neural models, or neural models based on non-contextualized embedding [29, 70]. An additional Hebrew BERT-based model, HeBERT [12], has been released, yet there is no reported evidence on performance improvements on key component of the Hebrew NLP pipeline — which includes, at the very least: morphological segmentation, full morphological tagging, and full (token/morpheme-based) named entity recognition.

Current implementations of BERT models employ sub-word tokenization schemes that facilitate dynamic generation of sub-word embedded vectors which elegantly solves the problem for OOV words. It is important to keep in mind though, that the sub-word tokenization used by BERT is not informed by or related to morphological-level information. Crucially, evaluating BERT-based models on morpheme-level tasks is non trivial.

These models output word-level embedded vectors, however sub-word morpheme-level vectors which are required for morpheme based tasks are not readily available.

We first present *AlephBERT*, a Hebrew pre-trained language model, larger and more effective than any Hebrew PLM before. Moreover, we introduce a novel language-agnostic Morphological Disambiguation architecture that extracts all of the sub-word morphological segments encoded in contextualized word embedding vectors. In doing so we are taking a different direction than the MA&D solutions described in the previous two chapters. Our MD model is operating directly on word vectors, i.e. without going through an intermediate MA step, performing disambiguation End-to-End (E2E), directly extracting morphological information from the contextualized word vectors provided by the PLM. Equipped with this new MD component we unlock a new PLM evaluation pipeline of multiple Hebrew tasks and benchmarks, that cover *sentence level*, *word-level* and *sub-word level* tasks.

We show substantial improvements on all essential tasks in the Hebrew NLP pipeline, tasks tailored to fit a *morphologically-rich language*, including: **Segmentation, Part-of-Speech Tagging, full morphological tagging, Named Entity Recognition** and **Sentiment Analysis**. Since previous Hebrew NLP studies used varied corpora and annotation schemes, we confirm our results on *all* existing Hebrew benchmarks and variants. For morphology and POS tagging, we test on both the Hebrew section of the SPMRL shared task [54], and the Hebrew UD corpus which was part of the CoNLL 2018 UD Shared Task [77]. For Named Entity recognition, we test on both the corpus of Ben Mordecai and Elhadad [8] and that of Bareket and Tsarfaty [7]. For sentiment analysis we test on the facebook corpus of Amram et al. [2], as well as a newer (fixed) variant of this benchmark.

We make our pre-trained model publicly available[1]. In the near future we will release the complete *AlephBERT*-geared pipeline we developed, containing the aforementioned tasks, as means for evaluating and comparing future Hebrew PLMs, and as a starting point for developing further downstream applications and tasks. We also plan to showcase *AlephBERT*'s capacities on downstream language understanding tasks such as: Information Extraction, Text Summarization, Reading Comprehension, and more.

[1] `huggingface.co/onlplab/AlephBERT-base`

6.1 The Resource Challenge

We present a case study for PLM development for Hebrew – a *morphologically rich* and *resource poor* language, long known to be notoriously hard to parse.

The resource-scarce setting is problematic for PLM development in at least two ways. First, there are insufficient amounts of free unlabeled text for pre-training. To wit, the Hebrew Wikipedia that was the source for training multilingual BERT is of orders of magnitude smaller than the English Wikipedia (See Table 6.1).[2] Secondly, there are no large-scale open-access commonly accepted benchmarks for fine-tuning and/or evaluating the performance of Hebrew PLMs on NLP/NLU downstream tasks. Previous studies on various tasks on Hebrew data do exist, each relying on disparate data sources, with varied evaluation metrics and annotation schemes even for the same task. To investigate Hebrew PLMs and probe their ability to capture linguistic structure, we introduce and evaluate Hebrew PLMs on the full set of tasks, sentence-based, token-based and morpheme-based tasks, including specific task variants and evaluation metrics.

Language	Oscar Size	Wikipedia Articles
English	2.3T	6,282,774
Russian	1.2T	1,713,164
Chinese	508G	1,188,715
French	282G	2,316,002
Arabic	82G	1,109,879
Hebrew	**20G**	**292,201**

Table 6.1: Corpora Size Comparison: High-resource (and Medium-resourced) languages vs. Hebrew.

6.2 AlephBERT Pre-Training

Data One of the most important factors driving the success of PLMs in other languages is the availability of enormous amounts of text to learn from. The PLM termed here *AlephBERT* is trained on a larger dataset and

[2]Of course, ample Hebrew data does exist online, but most of it is closed due to copyright issues and paywalls.

Corpus	File Size	Sentences	Words
Oscar (deduped)	9.8GB	20.9M	1,043M
Twitter	6.9GB	71.5M	774M
Wikipedia	1.1GB	6.3M	127M
Total	**17.9GB**	**98.7M**	**1.9B**

Table 6.2: Data Statistics for AlephBERT's training sets.

a larger vocabulary than any Hebrew BERT instantiation before. The Hebrew portions of Oscar and Wikipedia provides us with a training set size which is an order of magnitude smaller compared with resource-savvy languages, as shown in Table 6.1. In order to build a strong PLM we need a considerable boost in the amount of text that the PLM can learn from, which in our case comes form massive amounts of tweets added to the training set. The textual utterances provided by the Twitter sample API tend be short and diverge from valid syntax and canonical language use for the most part. While the free form language expressed in tweets might differ significantly from the text found in Oscar and Wikipedia, the sheer volume of tweets helps us close the resource gap substantially with minimal effort. Data statistics are provided in Table 6.2.

Specifically, we employ the following datasets for pre-training:

- **Oscar:** A deduplicated Hebrew portion of the OSCAR corpus, which is "extracted from Common Crawl via language classification, filtering and cleaning" [47].
- **Twitter:** Texts of Hebrew tweets collected between 2014-09-28 and 2018-03-07. We slightly cleaned up the texts by removing retweet signals "*RT:*", user mentions (e.g. "*@username*"), and URLs.
- **Wikipedia:** The texts in all of Hebrew Wikipedia, extracted using Attardi [3][3]

Configuration We used the Transformers training framework from Huggingface [74] and trained two different models — a small model with 6 hidden layers learned from the Oscar portion of our dataset, and a base model with 12 hidden layers which was trained on the entire dataset. The

[3]We make the corpus available on
https://github.com/OnlpLab/AlephBERT/tree/main/data/wikipedia.

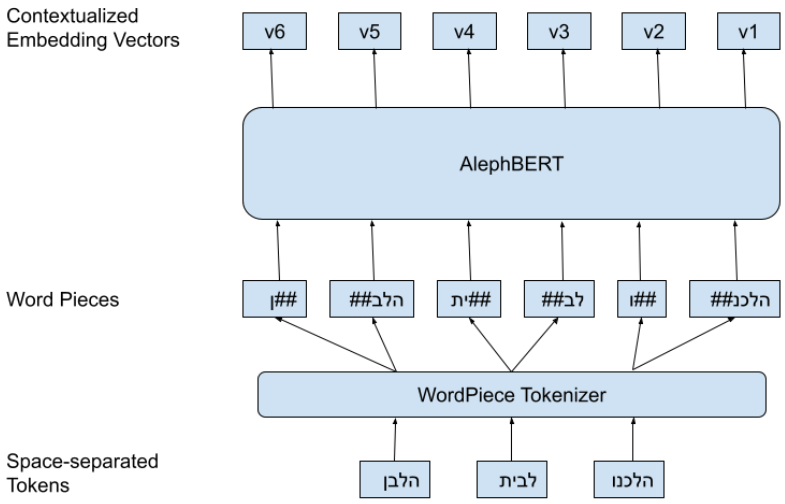

Figure 6.1: BERT is based on a fixed-sized vocabulary hence the raw input space-separated words are first transformed into word-pieces. The sequence of word pieces is then processed by the BERT model producing a sequence contextualized vectors, one per word-piece.

processing units used are wordpieces generated by training BERT tokenizers over the respective datasets with a vocabulary size of 52K in both cases. Traditionally, BERT models are optimized with an objective function optimized using both masked token prediction as well as next sentence prediction losses. Following the work on RoBERTa [34] we optimize AlephBERT with a masked-token prediction loss. We deploy the default masking configuration which is described in the appendix. Our choice of dataset forces us to ignore next sentence prediction optimization because a large portion of our data comprises of tweets which are unrelated and independent of each other (we did not attempt to reconstruct the discourse threads of retweets and replies).

Operation To optimize GPU utilization and decrease training time we split the dataset into 4 chunks based on the number of tokens in a sentence

as described in the appendix.

By limiting the number of tokens in a chunk it is possible to increase the number of samples loaded to each GPU. Consequently we are able to significantly increase the batch sizes, resulting in dramatically shorter training times

We trained for 5 epochs with learning rate set to 1e-4 followed by an additional 5 epochs with learning rate set to 5e-5 for a total of 10 epochs. We trained $\text{AlephBERT}_{\text{base}}$ over the entire dataset on an NVidia DGX server with 8 V100 GPUs which took us 8 days. $\text{AlephBERT}_{\text{small}}$ was trained over the Oscar portion only using 4 GTX 2080ti GPUs taking 5 days in total.

6.3 Experiments

Goal We set out to pre-train Hebrew PLMs and evaluate them empirically on a range of Hebrew NLP tasks. We evaluated the two AlephBERT variants (small and base) on the different tasks, in order to empirically gauge the effect of model size and data size on the quality of the language model. In addition, we compared the performance of our models to existing Hebrew BERT-based instantiations (mBERT [16] and HeBERT [12]). We evaluated the PLMs on all key tasks of the Hebrew NLP pipeline.

Benchmarks We evaluate our BERT-based models on various Hebrew NLP tasks using the following benchmarks:

- **Word Segmentation, Part-of-Speech Tagging, Full Morphological Tagging:**
  - The Hebrew Section of the SPMRL Task [54]
  - The Hebrew Section of the UD[4] treebanks collection [46]
- **Named Entity Recognition:**
  - Token-based NER evaluation based on the corpus of Ben-Mordecai and Elhadad [8]
  - Token-based and Morpheme-based NER evaluation based on the Named Entities and MOrphology (henceforth NEMO) corpus [7]

[4] `https://universaldependencies.org`

- **Sentiment Analysis:**
  - Sentiment Analysis evaluation based on the corpus of Amram et al. [2].
  - The aforementioned corpus is reported to leak between the test and train partitions, thus we provide a cleaned up version and evaluate on the updated split.

6.4 Tasks and Modeling Strategies

A key question when assessing BERT-based PLM performance for Hebrew concerns how to develop models for the different levels of granularity. Here we briefly sketch our modeling strategies, starting with the easiest (classification) tasks and continuing to the more challenging setups, involving the use of PLMs to predict the tokens' internal structures.

6.4.1 Sentence-Based Modeling

Sentiment Analysis The first task we report on is a simple sentence classification task, predicting the sentiment of a given sentence to one of three values: negative, positive, neutral. We trained and evaluated BERT-based sentence classification on two variants of the Hebrew Sentiment dataset of Amram et al. [2].

The first variant is the original sentiment dataset with an additional split to create a dev set (the original paper had only train and test split, and the test set remains the same). The dev set contains 10% of the train data which leaves us with a split of 70-10-20.

Unfortunately, the original dataset had a significant data leakage between the splits, with duplicates in the data samples. After removing the duplicates out of the original 12,804 sentences, we are left with a dataset of size 8,465.[5]

Training sentiment models entails adding a classification head on top of our AlephBERT PLM and running 15 fine-tuning epochs using each of the sentiment datasets.

[5] https://github.com/OnlpLab/Hebrew-Sentiment-Data

Raw input	*lbit hlbn*				
Space-delimited tokens	*hlbn*		*lbit*		
Segmentation	*lbn*	*h*	*bit*	*h*	*l*
POS	ADJ	DET	NOUN	DET	ADP
Morphology	Gender=Masc|Number=Sing	PronType=Art	Gender=Masc|Number=Sing	PronType=Art	-
Token-level NER	E-ORG		B-ORG		
Morpheme-level NER	E-ORG	I-ORG	I-ORG	B-ORG	O

Table 6.3: Illustration of Evaluated Token and Morpheme-Based Downstream Tasks. The input is the two-word input phrase "*lbit hlbn*" (*to the white house*).

6.4.2 Token-Based Modeling

Named Entity Recognition For the NER task, we initially assume a token based sequence labeling model. The input comprises of the sequence of tokens in the sentence, and the output contains BIOES tags indicating entity spans. The token-based model is a simple fine-tuned model using the Transformer's token-classification script of Wolf et al. [74].

We evaluate this model on two corpora. The first is by Ben Mordecai and Elhadad [8], henceforth, the BMC corpus, who annotated entities at token level. This means that a Hebrew token containing both a preposition and an entity mention will not deliver the entity mention boundaries. The BMC contains 3294 sentences and 4600 entities, and has seven different entity categories (DATE, LOC, MONEY, ORG, PER, PERCENT, TIME). To remain compatible with the original work we train and test the models on the 3 different splits as in Bareket and Tsarfaty [7].[6] For the BMC corpus we report token-based F1 scores on the detected entity mentions.

The second corpus is an extension of the SPMRL dataset with Named Entities annotation, also marked by BIOSE tags, respecting the precise (token-internal) morphological boundaries of NEs (henceforth, NEMO, standing for Named Entities and MOrphology) [7]. This corpus provides both a token-based and a morpheme-based annotation of the entities, where the latter contains the accurate (token-internal) entity boundaries. The NEMO corpus has nine categories (ANG, DUC, EVE, FAC, GPE, LOC, ORG, PER, WOA). It contains 6220 sentences and 7713 entities, and we used the standard SPMRL Train-Dev-Test, as in Bareket and Tsarfaty [7].

We trained each NER model over 15 epochs and report token-based F1 scores on the detected entity mentions.

[6] `https://github.com/OnlpLab/HebrewResources/tree/master/BMCNER`

6.4.3 Morpheme-Based Modeling

Modern Hebrew is a Semitic language with rich morphology and complex orthography. As a result, the basic processing units in the language are typically smaller than a given token's span. To probe AlephBERT's capacity to accurately predict such token-internal linguistic structure, we test our models on four tasks that require knowledge of the internal morphology of the raw tokens:

- **Segmentation**
  Input: A Hebrew sentence containing raw space-delimited tokens
  Output: A sequence of morphological segments representing basic processing units.[7]
- **Part-of-Speech Tagging**
  Input: A Hebrew sentence containing raw space-delimited tokens
  Output: Segmentation of tokens into basic processing units as above, where segments are tagged with single disambiguated POS tag.
- **Morphological Tagging**
  Input: A Hebrew sentence containing raw space-delimited tokens
  Output: Segmentation of tokens into basic processing units as above, where each segment is tagged with a single POS tag and a set of morphological features.[8]
- **Morpheme-Based NER**
  Input: A Hebrew sentence containing raw space-delimited tokens
  Output: Segmentation of the tokens to basic processing as above where segment is tagged with a BIOSE tags indicating entity spans, along with the entity-type label.

An illustration of these tasks is given in Table 6.3.

As opposed to fine-tuning the PLM model parameters, as done in sentence based and token based classification tasks, segmented morphemes are not readily available in the BERT representation. In order to provide proper

[7]Complying with 2-level representation of tokens defined by UD – each unit corresponds to a single POS tag. https://universaldependencies.org/u/overview/tokenization.html

[8]Equivalent to the AllTags evaluation metric defined in the CoNLL18 shared task. https://universaldependencies.org/conll18/results-alltags.html

segmentation and labeling for the four aforementioned tasks we developed a model designated to produce the morphological segments of each token in context.

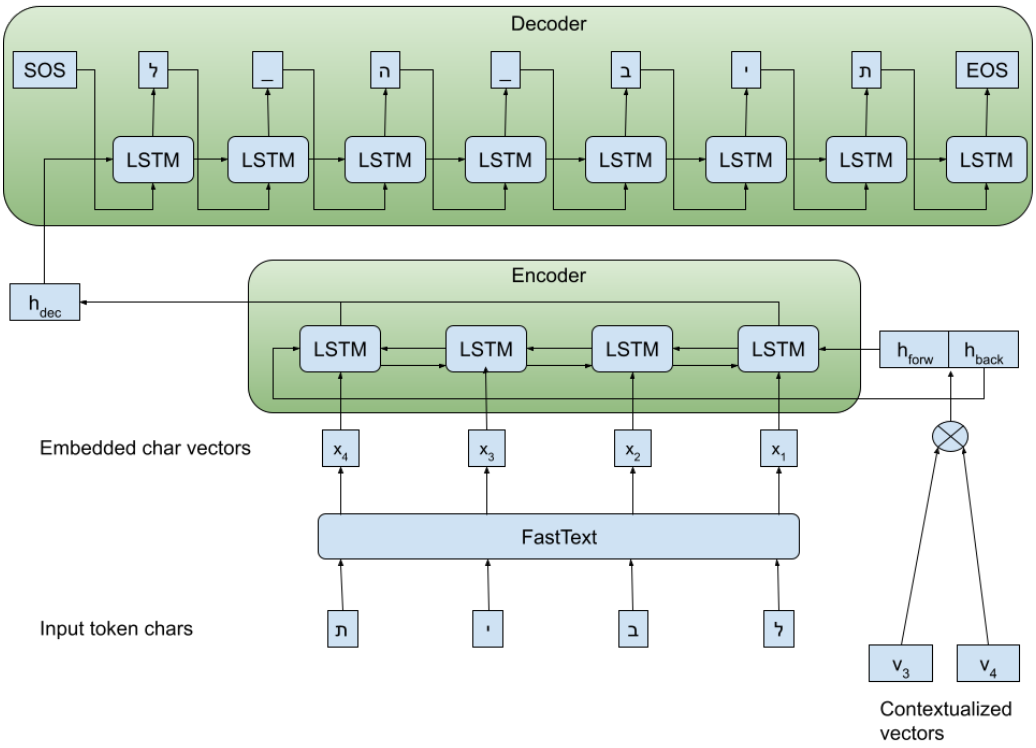

Figure 6.2: Given a word in the original sentence, we first combine the embedded vectors associated with the word-pieces (v3 and v4 representing the word-piece vectors generated in Figure 6.1) thus generating the required word context vector. This context vector is used as the initial hidden state of the BiLSTM which encodes the characters of the origin word. The decoder is an LSTM which outputs a sequence of characters, where the special symbol '_' indicates a morphological segment boundary (we use '_' in this figure as a representation of the space symbol which we actually use to signal segment boundaries in our implementation).

The morphological segmentation model which we designed is composed of a PLM responsible for transforming input tokens into contextualized embedded vectors, which we then feed into a char based seq2seq module that extracts the output segments. The seq2seq module is composed of an encoder implemented as a simple char-based BiLSTM, and a decoder implemented as a char-based LSTM generating the output character sym-

bols, or a space symbol signalling the end of a morphological segment. The morphological segmentation model architecture is illustrated in Figure 6.2. We train the model for 15 epochs, optimizing next-character prediction loss function.

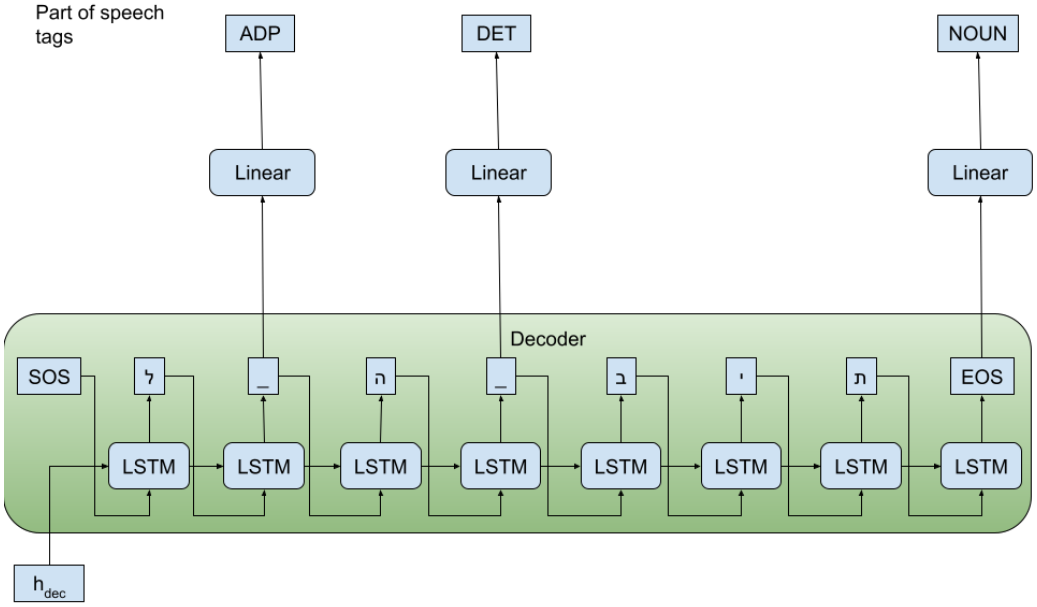

Figure 6.3: Based on the segmentation model, whenever the decoder predicts a morphological segment boundary, the current state of the decoder is used as input into a linear layer that is acting a multi-label classifier, assigning a score to each label. In this example we are performing Part of Speech Tagging for each predicted segment.

For the other tasks, involving both segmentation and labeling we deploy an MTL (multi-task learning) setup. That is, when generating an end-of-segment symbol, the model then predicts task labels which can be one or more of the following: POS-tag, NER-tag, morphological features. In order to guide the training to learn we optimize the combined segmentation and label prediction loss values. The morphological multitask model architecture is illustrated in Figure 6.3.

Currently we simply add together the loss values, but we note that as a future improvement it is likely that assigning different weights to the different loss values could prove to be beneficial. All morphological labeling models are evaluated on the Hebrew portions of the CoNLL 2018 UD Shared Task [77] and SPMRL shared task [54].

In addition, we design another setup for morphological NER sequence la-

beling task in which we first segment the text (using the above-mentioned segmentation model) and then perform fine-tuning with a token classification attention head directly applied to the PLM (similar to the way we fine-tune the PLM for the token-based NER task described in the previous section). In this pipeline setup we utilize the PLM twice; as part of the segmentation model to generate segments, which we then feed directly into the PLM (augmented with a token classification head) which is fine-tuned for the specific labeling task. We acknowledge the fact that we are fine-tuning the PLM using morphological segments even though it was originally pre-trained without any knowledge of sub-token units. But, as we shall see shortly, this seemingly unintuitive strategy performs surprisingly well.

6.5 Results

Sentence-Based Tasks The Sentiment analysis experimental results are provided in Table 6.4. As can be seen, all BERT-based models substantially outperform the original CNN Baseline reported by Amram et al. [2]. Interestingly, both AlephBERT$_{\text{small}}$ and AlephBERT$_{\text{base}}$ outperform all BERT-based variants, with BERT-base setting new SOTA results on the new (fixed) dataset.

Token-Based Tasks For our two NER benchmarks, we report the NER F1 scores on the token-based fine-tuned model in Table 6.5.

Here, we see noticeable improvements for the mBERT and HeBert variants over the current SOTA, but the most significant increase is in the AlephBERT$_{\text{base}}$ model. We also see a substantial difference between the AlephBERT$_{\text{small}}$ and AlephBERT$_{\text{base}}$ models, with the latter providing a new SOTA results on these both data sets. Crucially, this holds for the *token-based* evaluation metrics (as defined in Bareket and Tsarfaty [7]).

Morpheme-Based Tasks As a particular novelty of this work, we report BERT-based results on sub-token (segment-level) information. Specifically, we evaluate segmentation F1, POS F1, Morphological Features F1 and morphem-base NER F1, compared against the disambiguated labeled segments. In all cases we use raw space-delimited tokens as input, letting the BERT-based models perform *both* the segmentation and labeling.

	Old(leak) token	Old(leak) morph	New(fixed) token	New(fixed) morph
Previous SOTA	89.2	87.5	NA	NA
mBERT	92.12	92.18	84.21	85.58
HeBERT	92.48	92.27	87.13	86.88
$\text{AlephBERT}_{\text{small}}$	**93.15**	**92.70**	88.3	87.38
$\text{AlephBERT}_{\text{base}}$	91.63	92.01	**89.02**	**88.71**

Table 6.4: Sentiment Analysis Scores on the Facebook Corpus. Previous SOTA is reported by Amram et al. [2].

	NEMO	BMC
Previous SOTA	77.75	85.22
mBERT	79.07	87.77
HeBERT	81.48	89.41
$\text{AlephBERT}_{\text{small}}$	78.69	89.07
$\text{AlephBERT}_{\text{base}}$	**84.91**	**91.12**

Table 6.5: Token-Based NER Results on the NEMO and the Ben-Mordecai Corpora. Previous SOTA on both corpora has been reported by the NEMO models of Bareket and Tsarfaty [7].

Table 6.6 presents the segmentation, POS tags, and morphological tags F1 for the SPMRL dataset, all evaluated at the granularity of morphological segments. We report the aligned multiset F1 Scores as in previous work on Hebrew [43].

We see that segmentation results for all BERT-based models are similar, and they are already at the higher range of 97-98 F1 scores, which are hard to improve further.[9] For POS tagging and morphological features, all BERT-based models significantly outperform the previous SOTA provided by [56] (referred to as PtrNet) for POS tags and [43] (referred to as YAP) for morphological features. With respect to all BERT-based variants, we see an improvement for AlephBERT on all other alternatives, but on a small scale. That said, we do notice a repeating trend that places $\text{AlephBERT}_{\text{base}}$ as the best model for all of our morphological tasks, indicating that the improvement provided by the depth of the model and a larger dataset does also improve the ability to capture token-internal structure.

These trends are replicated on the UD Hebrew corpus, for two different evaluation metrics — the Aligned MultiSet F1 Scores as in previous work on Hebrew [43], [56], and the Aligned F1 scores metrics in the UD shared task [77] — as reported in Tables 6.7 and 6.8 respectively. AlephBERT ob-

[9] Some of these errors are due to annotation errors, or truly ambiguous cases.

	Segmentation	POS	Morph Features
Previous SOTA	NA	90.49	85.98
mBERT-morph	97.36	93.37	89.36
HeBERT-morph	97.97	94.61	90.93
$\text{AlephBERT}_{\text{small}}$-morph	97.71	94.11	90.56
$\text{AlephBERT}_{\text{base}}$-morph	**98.10**	**94.90**	**91.41**

Table 6.6: Morpheme-Based Aligned MultiSet (mset) F1 Results on the SPMRL Corpus. Previous SOTA is as reported by Seker and Tsarfaty [56] (POS) and More et al. [43] (morphological features)

	Segmentation	POS	Morph Features
Previous SOTA	NA	94.02	NA
mBERT-morph	97.70	94.76	90.98
HeBERT-morph	98.05	96.07	92.53
$\text{AlephBERT}_{\text{small}}$-morph	97.86	95.58	92.06
$\text{AlephBERT}_{\text{base}}$-morph	**98.20**	**96.20**	**93.05**

Table 6.7: Morpheme-Based Aligned MultiSet (mset) F1 Results on the UD Corpus. Previous SOTA is as reprted by Seker and Tsarfaty [56] (POS)

	Segmentation	POS	Morph Features
Previous SOTA	96.03	93.75	91.24
mBERT-morph	97.17	94.27	90.51
HeBERT-morph	97.54	95.60	92.15
$\text{AlephBERT}_{\text{small}}$-morph	97.31	95.13	91.65
$\text{AlephBERT}_{\text{base}}$-morph	**97.70**	**95.84**	**92.71**

Table 6.8: Morpheme-Based Aligned (CoNLL shared task) F1 Results on the UD Corpus. Previous SOTA is as reported by Minh Van Nguyen and Nguyen [38]

tains the best results for all tasks, even if not by a large margin.

Morpheme-Based NER Earlier in this section we considered NER as a token-based task that simply requires fine-tuning on the token labels. However, this setup is not accurate enough and less useful for downstream tasks, since the exact entity boundaries are often token internal [7]. We hence also report here morpheme-based NER evaluation, respecting the exact boundaries of the Entity mentions. To obtain morpheme-based

Architecture Segmentation Scores (aligned mset)	Pipeline (Oracle) Seg	 NER	Pipeline (Predicted) Seg	 NER	MultiTask Seg	 NER
Previous SOTA (NEMO)	100.00	79.10	95.15	69.52	97.05	77.11
mBERT	100.00	77.92	97.68	72.72	97.24	72.97
HeBERT	100.00	82	98.15	76.74	97.92	74.86
$\text{AlephBERT}_{\text{small}}$	100.00	79.44	97.78	73.08	97.74	72.46
$\text{AlephBERT}_{\text{base}}$	100.00	83.94	**98.29**	**80.15**	98.19	79.15

Table 6.9: Morpheme-Based NER F1 Evaluation on the NEMO Corpus. Previous SOTA is as reported by Bareket and Tsarfaty [7] for the Pipeline (Oracle), Pipeline (Predicted) and a Hybrid (almost-joint) Scenarios, respectively.

labeled-span of Named Entities as discussed above we could either employ a pipeline, first predicting segmentation and then applying a fine tuned labeling model *directly on the segments*, or we can use the MTL model and predict NER labels *while* performing the segmentation.

Table 6.9 presents segmentation and NER results for three different scenarios: (i) a pipeline assuming gold segmentation (ii) a pipeline assuming the best predicted segmentation (as predicted above) (iii) obtaining the segmentation and NER labels jointly in the MTL setup.

As our results indicate, $\text{AlephBERT}_{\text{base}}$ consistently scores highest in both pipeline (oracle and predicted) and multi-task setups. Pipeline-Predicted scores show clear correlation between a higher segmentation quality of a PLM and its ability to produce better NER results. Moreover, the differences in NER scores between the models are considerable (unlike the subtle differences in segmentation, POS and morphological features scores) and draw our attention to the relationship between the size of the PLM, the size of the pre-training data and the quality of the final NER models. Specifically, HeBERT and $\text{AlephBERT}_{\text{small}}$ were pre-trained with similar datasets and comparable vocabulary sizes. HeBERT, with its 12 hidden layers, performs significantly better compared to $\text{AlephBERT}_{\text{small}}$ which is composed of only 6 hidden layers. It thus appears that semantic information is learned in those deeper layers which helps in both learning to discriminate entities and improve the overall morphological segmentation capacity.

Comparing HeBERT to $\text{AlephBERT}_{\text{base}}$ we point to the fact that they are both modeled with the same 12 hidden layer architecture, the only dif-

ferences between them are in the size of their vocabularies (30K vs 52K respectively) and the size of the training data. The improvements exhibited by AlephBERT$_{\text{base}}$, compared to HeBERT, suggests that it is a result of the large amounts of training data and larger vocabulary available in our setup. By exposing AlephBERT$_{\text{base}}$ to an amount of text which order of magnitude larger we increased its NER capacity.

Finally, our NER experiments curiously suggest that a pipeline composed of the near-to-perfect MD followed by AlephBERT$_{\text{base}}$ augmented with a token classification head applied on the disambiguated segments is the best strategy for generating morphologically-aware NER labels.

6.6 Summary

In this chapter we build and use a large pre-trained language model for Modern Hebrew. Most importantly, with our AlephBERT$_{\text{base}}$ we are able to provide an E2E MD component that obtains state-of-the-art results on the tasks of segmentation, Part of Speech Tagging, Named Entity Recognition, and Sentiment Analysis. We outperform both general multilingual PLMs (mBERT) as well as language specific instantiations (HeBERT). Using the new AlephBERT model we are now gaining similar benefits as achieved in high resource languages from PLMs.

7 Conclusion

Modern Hebrew, a morphologically rich and resource-scarce language, has for long suffered from a gap in the resources available for NLP applications, and lower level of empirical results than observed in other, resource-rich languages. This work steps towards remedying the situation by addressing the challenges involved with morphological disambiguation for MRLs.

We describe 3 different approaches – feature-engineered joint morpho-syntactic parser (chapter 4), neural pointer network MD (chapter 5), and pre-trained language model (a.k.a AlephBERT) fine-tuning (chapter 6) – each taking a different path to modeling morphologically-aware tasks. We empirically evaluate our models on morphological level benchmarks using two evaluation methods. The CoNLL 2018 UD Shared Task evaluation script provides an evaluation based on aligned morphemes. In addition we advocate for an aligned token multi-set intersection evaluation which is more informative to downstream tasks that operate on morpheme sequences.

Chapters 4 and 5 both deploy MA&D strategy to morphological disambiguation by exploiting morphological information provided by a broad-coverage lexicon-backed MA component. In both cases the task is framed as a lattice disambiguation problem – given an ambiguous morphological lattice containing all possible analyses for each word provided by a *Morphological Analyzer* (MA), a *Morphological Disambiguation* (MD) model chooses the most likely sequence of morphemes for the given sentence.

The pre-neural dependency parser described in chapter 4 takes a disambiguated morphological sequence as input to a transition-based dependency parser that chooses the most likely set of dependency relations between the input morphemes. In such a pipeline, the ability of the parser to generate optimal parse trees depends on the quality of the disambiguated sequence of morphemes (wrong morphological segmentation will neces-

sarily lead to a sub-optimal parse tree). The accuracy of the MD component depends on the lexical coverage offered by the MA component that comes before it. Our investigation into the impact of the MA on the final dependency parser accuracy establishes the impact of incomplete lattices produced by the MA on the overall degraded accuracy of the generated parse trees. Thus we recommend the UD community to invest in broad coverage lexicons that would back high quality MA components which in turn would improve the overall dependency parsing performance especially in MRLs.

Chapter 5 follows the same lines – deploying a lexicon-backed MA component to generate ambiguous morphological lattices which are then passed as input to a neural MD model. Combining rich morphological information available via MA with the power of deep learning provided by a Pointer Network MD proved to be very powerful. We design a general framework that consumes lattice files and output a sequence of disambiguated morphemes, each containing segmentation boundary, lemma, part-of-speech tag and morphological features. The neural MD is language agnostic and our analysis of the results on both Hebrew and Turkish suggest that with a broad-coverage (i.e. high recall) MA, our MD has the capacity to find the correct analysis for each word (i.e. high precision). We show that access to symbolic morphological information aids the neural disambiguation model compared to strong baselines that only have access to the raw tokens. Additionally our MA&D solution achieved better results compared to the non-neural MD framework mentioned in chapter 4.

For over 15 years, having access to lexical resources has served Hebrew MD frameworks well. These frameworks were able to learn from rich morphological information explicitly provided by the MA. They transformed the raw token sequence into an ambiguous lattice and framed the problem as a lattice disambiguation task. The MA&D approach does come at a price though – it depends heavily on building and maintaining high quality lexicons as well as supporting various annotation schemes (e.g. SPMRL, UD) which is labor intensive, time consuming and costly.

In chapter 6 we take a different approach capitalizing on the latest algorithmic achievements in language modeling, freeing us from manually-generated lexicon-based MA as well as allowing us to develop MD framework processing directly on the sequence of raw tokens. Our pre-trained language model, named AlephBERT, specifically trained for Hebrew encodes the relevant morphological signal, thus enabling E2E state-of-the-art

solutions that outperforms all previous models on a variety of tasks that cover sentence level, token level and morphological level tasks. Specifically for E2E MD we take advantage of the contextualized word embedding vectors provided by AlephBERT and use them as input into a sequence-to-sequence neural network that extracts the morphological segments. We then combine this MD framework with various classifiers that can assign to each segment other grammatical labels thus enriching the morphological signatures.

The 3 works chronologically highlight the algorithmic advancements contributing to NLP in general and specifically to the MD task. Ever since the introduction of deep learning in NLP we have been witnessing a growing trend in training general purpose neural networks that can learn from large amounts of unlabeled text and integrated into task specific supervised models.

This revolution has shifted the focus of NLP researchers from modeling task-specific linguistic complexities to learning general language models that can generate contextualized word vectors capturing various linguistic signals. Given such a language model, a small annotated dataset is then needed to train a relatively simple network that is able to extract the specific information required for each task. This approach serves us well to overcome the small amounts of labeled data available at our disposal for the various tasks in the Hebrew NLP pipeline.

Besides uploading AlephBERT and making it publicly available, we will publish both data and code used to implement, train and evaluate the models described in this work. Looking ahead we are now working on developing a complete language processing framework for automatic annotation of Modern Hebrew Texts. We intend to release it as a python library which would be easy to install and to use. The framework will at first cover the basic most significant linguistic layers – morphological segmentation, POS tags, grammatical features as well as NER labels. Pending further research we will then enrich the output with deeper sentence-level annotations such as dependency parsing and semantic parsing. We hope the availability of an open-source, accurate, and easy to-use system for NLP in Hebrew will benefit the local NLP open-source community and greatly advance Hebrew language technology research and development, in academia and in the industry.

8 Appendix

8.1 Hebrew Transliteration

a	b	g	d	h	v	z	x	t	i	k	l	m	n	s	e	p	c	q	r	f	t
א	ב	ג	ד	ה	ו	ז	ח	ט	י	כ	ל	מ	נ	ס	ע	פ	צ	ק	ר	ש	ת

Table 8.1: Transliteration mapping of Hebrew letters as defined by Sima'any et al. [60]

8.2 Treebank Statistics

We use the train/dev/test split provided by the UD treebanks (as well as the Hebrew SPMRL treebank) The number of sentences in each split is described in Table 8.2

	English UD	Turkish UD	Arabic UD	Hebrew UD	Hebrew SPMRL
Train Sentences	12543	3685	6075	5241	4937
Development Sentences	2002	975	680	484	500
Test Sentences	2077	975	909	491	716

Table 8.2: Dataset splits

Table 8.3 gives some insight into the morphological degree of each language based on the respective UD test sets. For Turkish and Hebrew we delve deeper into the lattices to get a sense of the lexical coverage provided by the UD language MA component.

	English	Turkish	Arabic	Hebrew
Avg. Tokens per Sentence	12.08	10.26	41.56	25.01
Avg. Disambiguated Morphemes per Token	1.0	1.03	1.17	1.4
Avg. Ambiguous Analyses per Token		2.6		7.19

Table 8.3: UD Test Set Statistics

8.3 YAP

The joint transition systems implemented by the YAP morpho-syntactic parser is illustrated in Figure 8.1

Arc Eager:	Conf.	$c = (\sigma, \beta, A)$	$\sigma_h = A$ *second, 'head' stack*
	Initial	$c_s(x = x_1, ..., x_n) = ([0], [1, ..., n], \emptyset)$	
	Terminal	$C_t = \{c \in C \vert c = ([0], [], A)\}$	
	Transitions	$(\sigma, [i\vert\beta], A) \rightarrow ([\sigma\vert i], \beta, A)$	(SHIFT)
		$([\sigma\vert i], [j\vert\beta], A) \rightarrow ([\sigma\vert i\vert j], \beta, A \cup \{(i, l, j)\})$	(ArcRight$_l$)
		if $(k, l', i) \notin A$ *and* $i \neq 0$ *then*	
		$([\sigma\vert i], [j\vert\beta], A) \rightarrow ([\sigma], [j\vert\beta], A \cup \{(j, l, i)\})$	(ArcLeft$_l$)
		if $(k, l', i) \in A$ *then*	
		$([\sigma\vert i], \beta, A) \rightarrow (\sigma, \beta, A \cup \{(i, l, j)\})$	(REDUCE)

Arc ZEager:	Conf.	$c = (\sigma, \sigma_h, \beta, A)$	$\sigma_h = A$ *second, 'head' stack*
	Initial	$c_s(x = x_1, ..., x_n) = ([]_\sigma, []_h, [1, ..., n], \emptyset)$	Note: no root
	Terminal	$C_t = \{c \in C \vert c = ([]_\sigma, \sigma_h, [], A)\}$	For any σ_h, A
	Transitions	$([i]_\sigma, \sigma_h, []_\beta, A) \rightarrow ([]_\sigma, \sigma_h, []_\beta, A)$	(POPROOT)
		$([\sigma\vert i], \sigma_h, []_\beta, A) \rightarrow (\sigma, \sigma_h, []_\beta, A)$	(REDUCE$_2$)
		if T_L! =REDUCE *then*	T_L = Last Transition
		$(\sigma, \sigma_h, [i\vert\beta], A) \rightarrow ([\sigma\vert i], [\sigma_h\vert i], \beta, A)$	(SHIFT)
		if $\vert\beta\vert > 0$ *and* $(\vert\sigma\vert > 1$ *and* $(\vert\beta\vert > 1$ *or* $\vert\sigma_h\vert = 1))$ *then*	
		$([\sigma\vert i], \sigma_h, [j\vert\beta], A) \rightarrow ([\sigma\vert i\vert j], \sigma_h, \beta, A \cup \{(i, l, j)\})$	(ArcRight$_l$)
		if $\vert\beta\vert > 0$ *and* $\vert\sigma\vert > 0$ *then*	
		$([\sigma\vert i], \sigma_h, \beta, A) \rightarrow ([\sigma], \sigma_h, \beta, A \cup \{(i, l, j)\})$	(REDUCE$_1$)
		if $\vert\beta\vert > 0$ *and* $\vert\sigma\vert > 0$ *and* $(k, l', i) \notin A$ *and* $i = k$ *then*	
		$([\sigma\vert i], [\sigma_h\vert k], [j\vert\beta], A) \rightarrow ([\sigma], [\sigma_h], [j\vert\beta], A \cup \{(j, l, i)\})$	(ArcLeft$_l$)

Figure 8.1: Arc-Eager [30, Chapter 3] and Arc-ZEager [79] Systems.

8.4 Pointer Network MD

8.4.1 PtrNetMD Validation Performance

Table 8.4 and Table 8.5 show *Segmentation-only* and *Joint Segmentation-and-Tagging* scores on the CoNLL 2018 UD Shared Task validation sets.

Table 8.6 shows F1 aligned-mset scores on the UD treebank validation sets.

	English	Turkish	Arabic	Hebrew
UDPipe Oracle	100.00	100.00	100.00	100.00
UDPipe Predicted	99.09	97.93	93.64	84.69
Shared Task Leader				
PtrNetMD Infused		**99.57**		**96.61**
PtrNetMD Uninfused		97.77		95.22

Table 8.4: Segmentation-only F1, Aligned Segment, CoNLL18 UD Shared Task Validation Set. Top two rows are pipeline baseline. Bottom three rows are PtrNetMD compared to shared task leaders.

	English	Turkish	Arabic	Hebrew
UDPipe Oracle	94.57	92.52	95.61	96.03
UDPipe Predicted	93.72	91.02	89.81	84.69
Shared Task Leader				
PtrNetMD Infused		**96.68**		**94.18**
PtrNetMD Uninfused		89.59		92.05

Table 8.5: Joint Segmentation-and-Tagging F1, Aligned Segment, CoNLL18 UD Shared Task Validation Set. Top two rows are pipeline baseline. Bottom three rows are PtrNetMD compared to shared task leaders.

8.4.2 PtrNetMD Runtime Speeds

The average runtime of PtrNetMD on the test sets is 22.3 sentences per second on a GPU (GTX TITAN X). The token multi-tag processed 32 sentences per second while the token sequence-2-sequence processed 23 sentences per second. Processing was performed serially, one sentence at a time.

8.4.3 PtrNetMD and Baselines Implementation Details

We used PyTorch (v1.5) as the framework for all our models (token multi-tagging, token sequence-to-sequence and PtNetMD) We used the latest FastText (v0.9.2) models to produce embedding vectors for tokens (used by the baseline models) as well as forms and lemmas (used by PtrNetMD). All encoding and decoding layers in each of our models are based on a

	Turkish	Arabic	Hebrew
Token Multi-Tag	91.9	95.06	94.84
Token Seq-Tag	92.36	95.47	93.59
PtrNetMD infused	**96.74**		**96.21**
PtrNetMD uninfused	89.65		94.97

Table 8.6: Tagging F1, Aligned MSet, CoNLL18 UD Shared Task Validation Set

LSTM module. In all experiments we used AdamW [35] optimizer with a 0.001 learning rate. We trained each model for a total of 18 epochs, and selected the checkpoint that got the highest *aligned mset* evaluation score on the development set. PtrNetMD checkpoints were selected based on the uninfused joint segmentation and tagging *aligned mset* scores. We searched for hyper-parameters manually by trying out various combinations of standard values:

(1) LSTM layers: 1,2,3

(2) LSTM hidden size: 32,64,100,200,300

(3) LSTM Dropout: 0.0,0.1,0.3,0.5,0.7

One configuration detail we found useful was freezing (i.e. not updating the weights during the optimization step) the FastText embedding vectors used for tokens, lemmas and forms. The thought behind this decision is to make sure to avoid overfitting to the specific textual elements in the data which improved the generalization capabilities in all models.

The optimal values used by all models are:

(1) LSTM layers: 2

(2) LSTM hidden size: 64

(3) LSTM Dropout: 0.0

(4) FastText embedding: Freeze.

PtrNetMD Model Parameters

```
(lattice_emb): AnalysisEmbedding(
    (form_emb): Embedding(20267, 300, padding_idx=0)
    (form_dropout): Dropout(p=0.0, inplace=False)
    (lemma_emb): Embedding(9601, 300, padding_idx=0)
    (lemma_dropout): Dropout(p=0.0, inplace=False)
    (tag_emb): Embedding(20, 32, padding_idx=0)
    (feats_emb): Embedding(109, 32, padding_idx=0)
)
(lattice_encoder): LSTM(984, 64, num_layers=2,
             batch_first=True, bidirectional=True)
(analysis_decoder): LSTM(984, 128, batch_first=True)
(analysis_attn): SequenceStepAttention()
```

Token Mulit-Tag Model Parameters

```
(input_emb): TokenCharEmbedding(
    (token_emb): Embedding(19115, 300, padding_idx=0)
    (token_dropout): Dropout(p=0.0, inplace=False)
    (char_emb): Embedding(89, 300, padding_idx=0)
    (char_lstm): LSTM(300, 32, batch_first=True)
)
(encoder): BatchEncoder(
    (rnn): LSTM(332, 32, num_layers=2,
            batch_first=True, bidirectional=True)
)
(dropout): Dropout(p=0.0, inplace=False)
(classifiers): ModuleList(
    (0): Linear(in_features=64, out_features=44,
                bias=True)
)
```

Token Sequence-to-Sequence Parameters

```
(enc_emb): TokenCharEmbedding(
    (token_emb): Embedding(19115, 300, padding_idx=0)
    (token_dropout): Dropout(p=0.0, inplace=False)
    (char_emb): Embedding(89, 300, padding_idx=0)
```

```
    (char_lstm): LSTM(300, 32, batch_first=True)
)
(encoder): LSTM(332, 64, num_layers=2,
           batch_first=True, bidirectional=True)
(dec_emb): Embedding(18, 32, padding_idx=0)
(decoder): SequenceStepDecoder(
    (rnn): LSTM(364, 128, batch_first=True)
    (output): Linear(in_features=128, out_features=18,
                bias=True)
)
```

8.5 AlephBERT Pre-training Details

For reference and to make our work reproducible we specify here the main steps taken and parameters used during training of AlephBERT. We utilized the Huggingface Transformers framework with most of the default training parameter values. Table 8.7 lists all of the training parameters that we have manually specified in our code. We also list the values used by the other models.

Training our AlephBERT-base model using the entire dataset proved to be technically challenging due to the model size and data size. Training the entire dataset without splitting it into chunks did not utilize the full processing capacity of the GPUs and would have taken several weeks to complete. To overcome this issue we followed the advice to split the dataset into chunks based on the number of tokens in a sentence. The first chunk consisted of 70M senetences with 32 or less tokens. By limiting the maximum number tokens we consequently limit the size of the training matrices used by this chunk which consequently allowed for significantly increasing the batch size which resulted in dramatically shorter training time - these 70M sentences took only 2.5 days to complete 5 epochs. The second chunk consisted of sentences having between 32 and 64 tokens, the third chunk between 64 and 128 and the final last chunk all sentences with more than 128 tokens:

	chunk1	chunk2	chunk3	chunk4
max num tokens	0>32	32>64	64>128	128>512
num sentences	70M	20M	5M	2M

	AlephBERT-base	AlephBERT-small	HeBERT	mBERT-cased
max_position_embeddings	512	512	512	512
num_attention_heads	12	12	12	12
num_hidden_layers	12	6	12	12
vocab_size	52K	52K	30K	120K†

Table 8.7: Huggingface BERT Configurations Comparison. †Only 2450 vocabulary entries contain Hebrew letters

8.5.1 Masked Language Model (MLM)

To train a deep bidirectional representation, we use MLM to mask 15% of the input tokens at random and then predict those masked tokens. MLM is like "fill in the blanks" where we randomly mask 15% of the input tokens to predict the original vocabulary id. We use [MASK] tokens only for pre-training, and they are not used for fine-tuning.

Of the 15% randomly chosen masked tokens

- 80% of the time masked words are replaced with [MASK] token
- 10% of the time, masked words are replaced with with a random token
- Remaining 10% of the time, masked words are unchanged

8.5.2 Fine Tuning

The fine-tuning method allows the language model to be tweaked through backpropagation. For fine-tuning the AlephBERT model, all of the parameters are fine-tuned using labeled data from the downstream tasks. Pre-training is expensive and is a one-time procedure, but fine-tuning is inexpensive. Pre-trained AlephBERT already encodes a lot of semantics, syntactic as well as morphological information about the language. Hence, it takes less time to train the fine-tuned model. Using pre-trained AlephBERT, we need very minimal task-specific fine-tuning and hence need less data for better performance for any of the NLP tasks.

Bibliography

[1] Meni Adler and Michael Elhadad. An unsupervised morpheme-based HMM for Hebrew morphological disambiguation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 665–672, Sydney, Australia, July 2006. Association for Computational Linguistics. doi: 10.3115/1220175.1220259. URL https://www.aclweb.org/anthology/P06-1084.

[2] Adam Amram, Anat Ben-David, and Reut Tsarfaty. Representations and architectures in neural sentiment analysis for morphologically rich languages: A case study from modern hebrew. In *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, pages 2242–2252, 2018. URL https://www.aclweb.org/anthology/C18-1190/.

[3] Giusepppe Attardi. Wikiextractor. https://github.com/attardi/wikiextractor, 2015.

[4] Oded Avraham and Yoav Goldberg. The interplay of semantics and morphology in word embeddings. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 422–426, Valencia, Spain, April 2017. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/E17-2067.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[6] Roy Bar-haim, Khalil Sima'an, and Yoad Winter. Part-of-speech tagging of modern hebrew text. *Nat. Lang. Eng.*, 14(2):223–251, April 2008. ISSN 1351-3249. doi: 10.1017/S135132490700455X. URL http://dx.doi.org/10.1017/S135132490700455X.

[7] Dan Bareket and Reut Tsarfaty. Neural modeling for named entities and morphology (nemo^2). *CoRR*, abs/2007.15620, 2020. URL https://arxiv.org/abs/2007.15620.

[8] Naama Ben Mordecai and Michael Elhadad. Hebrew named entity recognition. 2005.

[9] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. ISSN 2307-387X.

[10] Sabine Buchholz and Erwin Marsi. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City, June 2006. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W06-2920.

[11] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL https://www.aclweb.org/anthology/D14-1179.

[12] Avihay Chriqui and Inbal Yahav. Hebert —& hebemo: a hebrew bert model and a tool for polarity analysis and emotion recognition, 2021.

[13] Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics. doi: 10.3115/1218955.1218970.

[14] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 999888:2493–2537, November 2011. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2078183.2078186.

[15] Ryan Cotterell and Hinrich Schütze. Morphological word-embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1287–1292, Denver, Colorado, May–June 2015. Association for Computational Linguistics. doi: 10.3115/v1/N15-1140. URL https://www.aclweb.org/anthology/N15-1140.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://www.aclweb.org/anthology/N19-1423.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://www.aclweb.org/anthology/N19-1423.

[18] Kevin Duh and Katrin Kirchhoff. POS tagging of dialectal Arabic: A minimally supervised approach. In *Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages*, pages 55–62, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W05-0708.

[19] Yoav Goldberg. A primer on neural network models for natural language processing. *J. Artif. Int. Res.*, 57(1):345–420, September 2016. ISSN 1076-9757.

[20] Yoav Goldberg and Michael Elhadad. Easy-first dependency parsing of Modern Hebrew. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 103–107, Los Angeles, CA, USA, June 2010. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W10-1412.

[21] Nizar Habash and Owen Rambow. Arabic tokenization, part-of-speech tagging and morphological disambiguation in one fell swoop. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 573–580, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219911. URL https://www.aclweb.org/anthology/P05-1071.

[22] Nizar Habash, Ryan Roth, Owen Rambow, Ramy Eskander, and Nadi Tomeh. Morphological analysis and disambiguation for dialectal Arabic. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 426–432, Atlanta, Georgia, June 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/N13-1044.

[23] Dilek Z. Hakkani-Tür, Kemal Oflazer, and Gökhan Tür. Statistical morphological disambiguation for agglutinative languages. In *COLING 2000 Volume 1: The 18th International Conference on Computational Linguistics*, 2000. URL https://www.aclweb.org/anthology/C00-1042.

[24] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[25] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1031. URL https://www.aclweb.org/anthology/P18-1031.

[26] Alon Itai and Shuly Wintner. Language resources for Hebrew. *Language Resources and Evaluation*, 42(1):75–98, March 2008.

[27] Salam Khalifa, Nasser Zalmout, and Nizar Habash. Morphological analysis and disambiguation for Gulf Arabic: The interplay between resources and methods. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 3895–3904, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://www.aclweb.org/anthology/2020.lrec-1.480.

[28] Stav Klein and Reut Tsarfaty. Getting the ##life out of living: How adequate are word-pieces for modelling complex morphology? In *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 204–209, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.sigmorphon-1.24. URL https://www.aclweb.org/anthology/2020.sigmorphon-1.24.

[29] Stav Klein and Reut Tsarfaty. Getting the ##life out of living: How adequate are word-pieces for modelling complex morphology? In *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology, SIGMORPHON 2020, Online, July 10, 2020*, pages 204–209, 2020. doi: 10.18653/v1/2020.sigmorphon-1.24. URL https://doi.org/10.18653/v1/2020.sigmorphon-1.24.

[30] Sandra Kübler, Ryan McDonald, and Joakim Nivre. *Dependency Parsing*. Number 2 in Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2009. ISBN 1598295969, 9781598295962.

[31] Faisal Ladhak, Ankur Gandhe, Markus Dreyer, Lambert Mathias, Ariya Rastrow, and Björn Hoffmeister. Latticernn: Recurrent neural networks over lattices. In *INTERSPEECH*, 2016.

[32] John Lee, Jason Naradowsky, and David A. Smith. A discriminative model for joint morphological disambiguation and dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 885–894, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P11-1089.

[33] Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramón Fermandez, Silvio Amir, Luís Marujo, and Tiago Luís. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1176. URL https://www.aclweb.org/anthology/D15-1176.

[34] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. 2019. URL http://arxiv.org/abs/1907.11692.

[35] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam, 2018. URL https://openreview.net/forum?id=rk6qdGgCZ.

[36] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1166. URL https://www.aclweb.org/anthology/D15-1166.

[37] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[38] Amir Pouran Ben Veyseh Minh Van Nguyen, Viet Lai and Thien Huu Nguyen. Trankit: A light-weight transformer-based toolkit for multilingual natural language processing. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, 2021.

[39] Amir More and Reut Tsarfaty. Data-driven morphological analysis and disambiguation for morphologically rich languages and universal dependencies. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 337–348, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee. URL https://www.aclweb.org/anthology/C16-1033.

[40] Amir More and Reut Tsarfaty. Data-driven morphological analysis and disambiguation for morphologically rich languages and universal dependencies. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 337–348, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee. URL http://aclweb.org/anthology/C16-1033.

[41] Amir More, Özlem Çetinoğlu, Çağrı Çöltekin, Nizar Habash, Benoît Sagot, Djamé Seddah, Dima Taji, and Reut Tsarfaty. CoNLL-UL: Universal morphological lattices for Universal Dependency parsing.

In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC'18)*, 2018.

[42] Amir More, Özlem Çetinoglu, Çagri Çöltekin, Nizar Habash, Benoît Sagot, Djamé Seddah, Dima Taji, and Reut Tsarfaty. Conll-ul: Universal morphological lattices for universal dependency parsing. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018.*, 2018.

[43] Amir More, Amit Seker, Victoria Basmova, and Reut Tsarfaty. Joint transition-based models for morpho-syntactic parsing: Parsing strategies for MRLs and a case study from modern Hebrew. *Transactions of the Association for Computational Linguistics*, 7:33–48, March 2019. doi: 10.1162/tacl_a_00253. URL https://www.aclweb.org/anthology/Q19-1003.

[44] Thomas Mueller, Helmut Schmid, and Hinrich Schütze. Efficient higher-order CRFs for morphological tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 322–332, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D13-1032.

[45] Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 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)*, pages 915–932, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D07-1096.

[46] Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA). URL https://www.aclweb.org/anthology/L16-1262.

[47] Pedro Javier Ortiz Suárez, Laurent Romary, and Benoît Sagot. A monolingual approach to contextualized word embeddings for mid-resource languages. In *Proceedings of the 58th Annual Meeting of*

the Association for Computational Linguistics, pages 1703–1714, Online, July 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.acl-main.156.

[48] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL https://www.aclweb.org/anthology/D14-1162.

[49] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL https://www.aclweb.org/anthology/N18-1202.

[50] Ryan Roth, Owen Rambow, Nizar Habash, Mona Diab, and Cynthia Rudin. Arabic morphological tagging, diacritization, and lemmatization using lexeme models and feature ranking. In *Proceedings of ACL-08: HLT, Short Papers*, pages 117–120, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P08-2030.

[51] Benoît Sagot. A multilingual collection of CoNLL-u-compatible morphological lexicons. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA). URL https://www.aclweb.org/anthology/L18-1292.

[52] Cicero Dos Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging. volume 32 of *Proceedings of Machine Learning Research*, pages 1818–1826, Bejing, China, 22–24 Jun 2014. PMLR. URL http://proceedings.mlr.press/v32/santos14.html.

[53] Djame Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, D. Jinho Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola Galletebeitia, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam

Przepiórkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, Alina Wróblewska, and Villemonte Eric de la Clergerie. Proceedings of the fourth workshop on statistical parsing of morphologically-rich languages. pages 146–182. Association for Computational Linguistics, 2013. URL `http://aclweb.org/anthology/W13-4917`.

[54] Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. Introducing the spmrl 2014 shared task on parsing morphologically-rich languages. pages 103–109, 2014.

[55] E. Segal. Hebrew morphological analyzer for hebrew undotted texts. 1999.

[56] Amit Seker and Reut Tsarfaty. A pointer network architecture for joint morphological segmentation and tagging. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4368–4378, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.391. URL `https://www.aclweb.org/anthology/2020.findings-emnlp.391`.

[57] Yan Shao, Christian Hardmeier, and Joakim Nivre. Universal word segmentation: Implementation and interpretation. *Transactions of the Association for Computational Linguistics*, 6:421–435, 2018. doi: 10.1162/tacl_a_00033. URL `https://www.aclweb.org/anthology/Q18-1030`.

[58] Steven Shearing, Christo Kirov, Huda Khayrallah, and David Yarowsky. Improving low resource machine translation using morphological glosses (non-archival extended abstract). In *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Papers)*, pages 132–139, Boston, MA, March 2018. Association for Machine Translation in the Americas. URL `https://www.aclweb.org/anthology/W18-1813`.

[59] Qinlan Shen, Daniel Clothiaux, Emily Tagtow, Patrick Littell, and Chris Dyer. The role of context in neural morphological disambiguation. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 181–191, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee. URL `https://www.aclweb.org/anthology/C16-1018`.

[60] K. Sima'any, A. Itaiz, Y. Winterz, A. Altmanz, and N. Nativx. Building a tree-bank of modern hebrew text. volume 42, pages 347–380. Traitement automatique des langues, 2001.

[61] Peter Smit, Sami Virpioja, Stig-Arne Grönroos, and Mikko Kurimo. Morfessor 2.0: Toolkit for statistical morphological segmentation. In *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 21–24, Gothenburg, Sweden, April 2014. Association for Computational Linguistics. doi: 10.3115/v1/E14-2006. URL https://www.aclweb.org/anthology/E14-2006.

[62] Matthias Sperber, Graham Neubig, Jan Niehues, and Alex Waibel. Neural lattice-to-sequence models for uncertain inputs. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1380–1389, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1145. URL https://www.aclweb.org/anthology/D17-1145.

[63] Matthias Sperber, Graham Neubig, Ngoc-Quan Pham, and Alex Waibel. Self-attentional models for lattice inputs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1185–1197, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1115. URL https://www.aclweb.org/anthology/P19-1115.

[64] Milan Straka and Jana Straková. Tokenizing, POS tagging, lemmatizing and parsing UD 2.0 with UDPipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/K17-3009. URL https://www.aclweb.org/anthology/K17-3009.

[65] Milan Straka and Jana Straková. Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada, August 2017. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/K/K17/K17-3009.pdf.

[66] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances*

in Neural Information Processing Systems 27, pages 3104–3112. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf.

[67] Reut Tsarfaty. Integrated morphological and syntactic disambiguation for modern Hebrew. In *Proceedings ACL-CoLing Student Research Workshop*, pages 49–54, Stroudsburg, PA, USA, 2006. ACL. URL http://dl.acm.org/citation.cfm?id=1557856.1557867.

[68] Reut Tsarfaty. A unified morpho-syntactic scheme of Stanford dependencies. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 578–584, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P13-2103.

[69] Reut Tsarfaty, Djamé Seddah, Yoav Goldberg, Sandra Kuebler, Yannick Versley, Marie Candito, Jennifer Foster, Ines Rehbein, and Lamia Tounsi. Statistical parsing of morphologically rich languages (SPMRL) what, how and whither. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 1–12, Los Angeles, CA, USA, June 2010. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W10-1401.

[70] Reut Tsarfaty, Dan Bareket, Stav Klein, and Amit Seker. From SPMRL to NMRL: what did we learn (and unlearn) in a decade of parsing morphologically-rich languages (mrls)? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7396–7408, 2020. doi: 10.18653/v1/2020.acl-main.660. URL https://doi.org/10.18653/v1/2020.acl-main.660.

[71] Clara Vania and Adam Lopez. From characters to words to in between: Do we capture morphology? In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2016–2027, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1184. URL https://www.aclweb.org/anthology/P17-1184.

[72] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama,

and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5866-pointer-networks.pdf.

[73] Linlin Wang, Zhu Cao, Yu Xia, and Gerard de Melo. Morphological segmentation with window lstm neural networks. In *AAAI*, 2016.

[74] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[75] Eray Yildiz, Caglar Tirkaz, H. Sahin, Mustafa Eren, and Omer Sonmez. A morphology-aware network for morphological disambiguation. In *AAAI Conference on Artificial Intelligence*, 2016. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12370.

[76] Nasser Zalmout and Nizar Habash. Don't throw those morphological analyzers away just yet: Neural morphological disambiguation for Arabic. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 704–713, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1073. URL https://www.aclweb.org/anthology/D17-1073.

[77] Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. CoNLL 2018 shared task: Multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels, Belgium, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/K18-2001. URL https://www.aclweb.org/anthology/K18-2001.

[78] Yue Zhang and Stephen Clark. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1): 105–151, 2011. doi: 10.1162/coli_a_00037.

[79] Yue Zhang and Joakim Nivre. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, pages 188–193, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 978-1-932432-88-6. URL http://dl.acm.org/citation.cfm?id=2002736.2002777.

תוכן עניינים

שני המאמצים הללו מדגישים את תרומתם של משאבים לקסיקלים רחבים להגבלת מהותית של שגיאות מורפולוגיות, מה שמוביל לדיוק גבוה במשימות הצנרת. לבסוף אנו מכירים בחיסרון של מסגרות הניתוח והפגת עמימות מורפולוגית העוסקות בבניית רכיב ניתוח מורפולוגי בעל כיסוי רחב לשפה. בעקבות הכרה זו אנו בוחנים פיתרון חלופי הנהנה מסוג אחר של משאב לשוני רחב שאינו מצריך מומחיות לשונית או מאמץ ידני. אנו מציעים את אלף־ברט מודל שפה שאומן מראש במסגרת למידה ללא פיקוח על גבי קורפוס גדול המורכב מכמעט 100 מיליון משפטים בעברית מודרנית. אנו משתמשים במודל מאומן מראש זה כדי לבנות מפיג עמימות מורפולוגית ממומש קצה לקצה שמשיג תוצאות גבוהות ביותר ללא צורך בעבודה ידנית ויקרה ליצירת משאבים מילוניים או הנדסת מאפיינים לשימוש במודלים.

תקציר

בשפות עשירות מורפולוגית מילים הן דו משמעיות, מורכבות מיחידות תת מילים המכונות מורפמות.

מקרים כאלה דורשים לעיתים קרובות פירוק מורפולוגי כלומר ניבוי נכון של מילים למורפמות ושימוש במורפמות כיחידות העיבוד הבסיסיות בצינור העיבוד שפות טבעיות. מורפמות משמשות בפועל כקלט למשימות העיבוד, וכתוצאה מכך כל שגיאה שתעשה במהלך הפגת העמימות המורפולוגית תהיה בלתי אפשרית לתיקון בהמשך ותשפיע לרעה על ביצועי כל המשימות.

ישנן שתי אסטרטגיות אפשריות להפגת עמימות מורפולוגית. הראשון הוא לבנות משימה זו כתהליך דו שלבי המייצג מילים כמבנה נתונים סריג ועיצוב רכיב הפגת העמימות לבחירת הניתוח הסביר ביותר. שיטה זו ידועה בשם ניתוח והפגת עמימות מורפולוגית. האפשרות החלופית היא להפיג עמימות מורפולוגית כתהליך קצה לקצה הפועל ישירות על המלים (כלומר מבלי לעבור לייצוג ביניים). מצד אחד, מודלים של קצה לקצה חוזים ישירות מידע מורפולוגי מתוך המילים הגולמיות, אך אינם נהנים מגישה מפורשת למורפמות. מצד שני, מסגרות ניתוח והפגת עמימות מורפולוגית תלויות באיכות הסריגים מבחינת כיסוי כל הניתוחים האפשריים ובכך שלא יהיו חסרים ניתוחים. בעבודה זו אנו מיישמים 3 מודלים שונים של הפגת עמימות מורפולוגית, שניים מהם מיועדים לניתוח והפגת עמימות מורפולוגית ואילו השלישי עובד קצה לקצה.

על ידי כיסוי בחירות העיצוב השונות אנו מדגימים את השדרוגים שהושגו במשימת הפגת עמימות מורפולוגית בעברית בשנים האחרונות עם ההשפעה של למידה עמוקה בעיבוד שפה טבעית. מחקרי המקרה המוצגים בעבודה זו מתמקדים בעברית מודרנית, אך ניתן ליישמם בכל שפה. תחילה אנו מתמקדים ביישום מסגרת מורפו־תחבירית משותפת טרום־ניורונית הידוע בשמו "יאפ" על כל 82 השפות במשימה המשותפת בכנס למידת שפה טבעית מ2018 ומספקים ניתוח מעמיק של התוצאות בעברית מודרנית. לאחר מכן אנו מציעים גישה מבוססת למידה עמוקה בטופולוגיה של רשת הצבעה המיושמת הן בטורקית והן בעברית - ומשיגה שיפור דיוק משמעותי בהפגת העמימות המורפולוגית .בעברית

האוניברסיטה הפתוחה
המחלקה למתמטיקה ולמדעי המחשב

התפתחויות אחרונות באסטרטגיות להפגת עמימות מורפולוגית בעברית מודרנית

עבודת תזה זו הוגשה כחלק מהדרישות לקבלת תואר
"מוסמך למדעים" M.Sc. במדעי המחשב
באוניברסיטה הפתוחה
המחלקה למתמטיקה ומדעי המחשב

על־ידי
עמית סקר

בהנחיית פרופסור רעות צרפתי

יולי 2021