Date of acceptance Grade

Instructor

Neural models for unsupervised disambiguation in morphologically rich languages

José María Hoya Quecedo

Helsinki November 11, 2019

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
José María Hoya Quecedo

Työn nimi — Arbetets titel — Title

Neural models for unsupervised disambiguation in morphologically rich languages

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
MSc Thesis	November 11, 2019	53 pages + 0 appendices

Tiivistelmä — Referat — Abstract

The problem of morphological ambiguity is central to many natural language processing tasks. In particular, morphologically rich languages pose a unique challenge due to the large number of possible forms some words can take.

In this work, we implement and evaluate a method for morphological disambiguation of morphologically rich languages. We use deep learning techniques to build a disambiguation model and leverage existing tools to automatically generate a training data set.

We evaluate our approach on the Finnish, Russian and Spanish languages. For these languages, our method surpasses the state-of-the-art results for the tasks of part-of-speech and lemma disambiguation.

ACM Computing Classification System (CCS):

10010147.10010178.10010179.10010185
Computing methodologies Phonology / morphology

10010147.10010257.10010293.10010294
Computing methodologies Neural networks

Avainsanat — Nyckelord — Keywords
natural language processing, machine learning, morphological disambiguation

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

Acknowledgements

This work was supported in part by TEKES, the Finnish Funding Agency for Technology and Innovation, Project "iMEDL—Digital Media Evolution through Deep Learning" (number 5933/31/2017), and by Helsinki Institute for Information Technology HIIT.

I would like to thank my supervisor, Dr. Roman Yangarber, for his guidance throughout the entire process, for giving me the opportunity to work under his wing, and for sparking my interest in the field of NLP.

I would also like to acknowledge the help and valuable input given to me by my coworkers, Max Koppatz, Jue Hou, Anisya Katinskaya and Sardana Ivanova.

Finally, I would like to thank my family, my friends, and Silvia Azcárate, for their unquestioning love and support.

1 Introduction

1.1 Motivation

All natural languages have some degree of ambiguity, that is, we can find words or statements that can be interpreted in several ways. This is a consequence of the fact that languages tend to maximize the information transmitted while minimizing the encoding [25]—in the case of written language, word and sentence length—relying instead on word inter-dependencies within a sentence to convey an unambiguous idea.

Ambiguity can present itself in many different ways. Ambiguity can be either *structural*, when it occurs on the sentence or discourse level, or *lexical*, when it occurs on the word level.

One type of lexical ambiguity is *morphological ambiguity*, which takes place when there are several different possibilities as to how a word was constructed.

For example, the English word "are" can be analysed in two different ways:

- As a form of the verb "to be": you are, we are, etc.
- As the singular form of a noun meaning "100 square metres" (rarely used, in favour of its derived form "hectare").

Note that other forms are unambiguous, such as "ares," which can only be the plural of "are," or "am," which can only be the first person singular of the present indicative of "to be."

Many other types of lexical ambiguity exist, such as *word sense ambiguity*, when a single word has different meanings—as in English "plant," which can be an organism belonging to the Plantae kingdom or an industrial facility—or *homophony*, when two distinct words are pronounced in the same way—as in English "die" and "dye."

Such ambiguities are out of the scope of this work, although word sense ambiguity and morphological ambiguity overlap—since the latter always entails the former. In our previous example, the two possible analyses for "are" clearly mean two different things. However, word sense ambiguity can take place without morphological ambiguity, as with "plant," which can only be interpreted as the singular form of a noun and it has the same root regardless of its meaning in the context.

We consider a language to be *morphologically rich* when words can take many different forms depending on their exact meaning or how they are used in the sentence. These languages tend to have a high degree of ambiguity, since inevitably some forms of some words will be written in the same way as other forms of different words.

There are many cases in which morphological ambiguity presents a challenge for a computer trying to perform some automatic processing on a piece of text.

In our language learning application,[1] we have several uses for morphological disambiguation. For example, in one of the exercise types, users are provided a base form (the *lemma*, as defined in Section 2.1) for a missing word in a sentence and have to put it in the correct form [14]. In order to find the base form for an ambiguous word, we need to analyse it properly.

With automatic translation, morphological disambiguation is crucial to determining the correct meaning of a sentence and producing a translation which makes sense and where the words are in the correct form.

Another case where morphological disambiguation proves useful is in search engines, where the words of both the search query and the text are reduced to some canonical form, in order to be able to match relevant results in which the words are in a different form. For example, say we have the query "learning resource English" and the text "I learnt English with these resources." We can reduce the query to "learn" + "resource" + "English" and the text to "learn" + "English" + "resource" and thus match them easily, even though "learn" and "resource" are in different forms in the original query and text. For ambiguous words, this canonical form is not immediately obvious to the computer.

1.2 Goals

The main goal of this work is to design, implement and evaluate a method for morphological disambiguation of words in morphologically rich languages.

We need our approach to be language-independent, since there are many different morphologically rich languages in our application which currently lack a viable method for morphological disambiguation. Our focus is mainly on Finnish, but we will also use Russian and Spanish[2] to verify that the approach works for other

[1] `revita.cs.helsinki.fi`

[2] Since we have native speakers for all three and can thus manually confirm that the method is

languages.

In addition, these languages lack a sufficient amount of manually annotated data for traditional supervised methods, so our method needs to rely solely on unlabelled data and leverage existing tools, such as a morphological analyser.

Our main hypotheses are:

- For every ambiguous word in some context, the corpus will contain many morphologically *similar* words, which are *unambiguous* and which appear in *similar contexts*.
- We can capture the relationships between contexts and specific word forms.
- We can train the models by using an "automatically annotated" data set—which contains only *unambiguous instances*—which requires no expensive manual data annotation and no supervision. Automatic annotation can be performed by a morphological analyser.

1.3 Structure of this thesis

This work is structured as follows.

In Chapter 2 we define key terms used throughout this work and explore the different types of morphological ambiguity, as well as the viable approaches for each type.

In Chapter 3 we review relevant pieces of work in the field of morphological disambiguation, focusing on their merits as well as their shortcomings.

In Chapter 4 we describe in detail our solution to the problem and explain the techniques used in our approach.

In Chapter 5 we describe the evaluation procedure and the data used for evaluation.

In Chapter 6 we present the results of our experiments and discuss them.

In Chapter 7 we provide our conclusions and propose future lines of research.

---

working properly.

2 Problem description

2.1 Terminology and definitions

A **word** is the smallest unit of language which has meaning by itself. A word is defined by the following attributes:

- The **surface form** is the textual representation of the word. The surface form gives us all the information we need to infer the other attributes, although there may be several options, if it is ambiguous.
- The **lemma** is the canonical or "dictionary" form of the word. For example, for our languages, the lemma for nouns is the nominative singular and the lemma for verbs is the infinitive.[3] The criterion for which form is considered the lemma varies across languages—for instance, the lemma for verbs in Latin is the first person singular of the present indicative.
- The **part-of-speech** (POS) of a word is its morphosyntactic category. The POS determines both the morphological properties of the word (i.e., the set of morphological features) and the syntactic properties of the word (i.e., the role of the word in the sentence). Examples of POS are: verb, noun, adjective, etc.
- The **morphological features** of a word are the different parameters which define the morphology of said word. For example, nouns have the feature "number," which can take on two values in our languages, "singular" or "plural," depending on whether there is one or many of such noun; verbs have the feature "tense," which can be "present," "past," etc. depending on when the action takes place.

An **inflectional morpheme** is a string of characters that marks a set of morphological features for a given POS. In fusional languages such as Finnish[4], a single inflectional morpheme usually corresponds to several morphological features (e.g., case and number for nouns, or tense, mood and person for verbs).

For example, the Finnish word *Suomessa* ("in Finland") has the surface form *Suomessa*, the lemma *Suomi* ("Finland") and the POS "noun." The suffix *-ssa*

[3] For Finnish—which has 5 infinitives—the lemma is the first or *-a* infinitive.

[4] Note that Finnish exhibits *some degree* of fusion but is not completely fusional.

is an inflectional morpheme that marks the inessive case (meaning "in" or "inside") and the singular number.

Morphological analysis is the task of breaking down a surface form into its lemma, POS and morphological features (also called "tags") by means of a **morphological analyser**. As mentioned previously, the analyser may give more than one possible valid analysis for a surface form. Each of the possible analyses given by the analyser is referred to as a **reading**.

The morphological analysis for *Suomessa* would be:[5]

```
Suomi+N+Prop+Sg+Ine
```

Each analyser tag corresponds to a morphological feature:

- `N`: Noun.
- `Prop`: Proper noun.
- `Sg`: Singular number.
- `Ine`: Inessive case.

Furthermore, according to the inflectional pattern of a word (i.e., the set of inflectional morphemes it accepts), we can define two categories for words:

- **Declinable** words are those that accept inflectional morphemes, and so a single lemma can produce several distinct surface forms.
- **Indeclinable** words are those that do not accept inflectional morphemes—which means that the surface form for these words will always be equal to their lemma.

Whether a word is declinable is generally indicated by its POS. In our set of languages, a given POS always gives the same inflectional pattern; we do not have edge cases like English modal verbs, which are verbs—a declinable POS in general—but have a fixed, single conjugation and are thus indeclinable (e.g., can, must, etc.). In any case, it would be reasonable to create a new POS for these specific words, if we did have them.

[5] `Ine`: inessive case.

2.2 Morphological ambiguity

Morphological ambiguity occurs when a morphological analyser gives more than one reading for a surface form. Morphological ambiguity can take three different forms, which we will treat as separate problems in this work:

- **POS ambiguity** occurs when the analyser gives more than one possible POS for the surface form.
- **Lemma ambiguity** occurs when the analyser gives more than one possible lemma for the surface form.
- **Feature ambiguity** occurs when there is more than one possible reading for a single POS and lemma combination—that is, there are several different possible sets of morphological features.

For instance, take the Finnish surface form *maksaa* ("liver" / "to cost"), with the following morphological analysis:[6]

```
maksa+N+Sg+Par
maksaa+V+Act+Ind+Prs+Sg3
maksaa+V+InfA
```

This surface form has three readings: one with the lemma *maksa* ("liver") and the POS "noun," and two with the lemma *maksaa* ("to cost") and the POS "verb." The verbal readings are an example of *syncretism*, where the first infinitive coincides with the third person singular of the present indicative.

In this case, all three types of morphological ambiguity are present:

- There are two possible POS, "noun" and "verb."
- There are two possible lemmas, *maksa* and *maksaa*.
- For the lemma *maksaa* and POS "verb," there are two different possible sets of morphological features: the third person singular of the present indicative, and the first infinitive.

[6] `Par`: partitive case, `V`: verb, `Act`: active voice, `Ind`: indicative mood, `Prs`: present tense, `Sg3`: third person singular, `InfA`: first infinitive.

In addition, we can see that the problems of POS and lemma disambiguation are equivalent in this instance—for example, if we know the correct POS is "verb," we know the lemma is *maksaa*, and vice versa—and, as we will see in the following section, this is indeed the case for most instances. Furthermore, for the instances where the lemma is *maksa*, the feature ambiguity is also resolved.

In Section 1.1 we saw that, for most tasks—i.e., those requiring text canonicalization—we only need to find the lemma for a given surface form. Therefore, in this work, we tackle only the problems of POS and lemma disambiguation as a first approach that covers most major use cases. Nevertheless, the method could be extended to handle feature ambiguity, as we will briefly discuss in Section 7.2.

2.3 Types of morphological ambiguity

Based on the three types of ambiguity and the declinable-indeclinable classification, we can define a two-axis taxonomy for ambiguities which will group together ambiguities that can be solved with a certain disambiguation method. One axis indicates whether the words are declinable or indeclinable, and the other axis indicates whether there is POS ambiguity, lemma ambiguity, or both.

Note that, for simplicity, we will assume that the ambiguities are two-way—i.e., there are two possible readings—since that is by far the most common kind in our corpora.[7] For n-way ambiguities, we would have to choose the intersection of the viable methods for each pair of ambiguities .

For instance, consider the Spanish surface form *bajo*, which can be a form of the adjective *bajo* ("low"), the preposition *bajo* ("under") or the verb *bajar* ("to lower"). It is thus a three-way ambiguity with the following pairs:

1. (*bajar*, verb) and (*bajo*, adjective).
2. (*bajar*, verb) and (*bajo*, preposition).
3. (*bajo*, adjective) and (*bajo*, preposition).

As we will see later in this section, the first pair can be solved either through POS or lemma disambiguation, and the next two can only be solved through POS

[7] See Section for a description of the corpus used for each language.

disambiguation. Therefore, we can solve the three-way ambiguity using POS disambiguation. If the intersection is the empty set, then the ambiguity is unsolvable with our method.

Table 1 provides a summary of the effective approaches for each category.

In Table 2, we can see that POS disambiguation can effectively solve the majority of ambiguities, except for those in which the POS is the same across readings.

2.3.1 Ambiguity between two declinable words

Different POS, different lemma This is the easiest type to solve, since, in general, the words will only partially overlap in their surface forms: different lemmas with a different inflectional paradigm—given that the POS is different—will generate many distinct surface forms. For example, consider the Finnish word *tuli* ("fire" / "he came"), with the following morphological analysis:[8]

```
tuli+N+Sg+Nom
tulla+V+Act+Ind+Prt+Sg3
```

This surface form can be a form of either the noun *tuli* ("fire") or the verb *tulla* ("to come"). Both POS are declinable, so we can find unambiguous surface forms corresponding to each lemma.

For example, for *tuli* we have the unambiguous partitive case *tulta*:

```
tuli+N+Sg+Par
```

And for *tulla* the second person of the same tense, *tulit*, is unambiguous:[9]

```
tulla+V+Act+Ind+Prt+Sg2
```

Therefore, we expect to find enough unambiguous forms of both *tuli* and *tulla* to be able to model the context in which each of them generally appears, and so decide whether a given context is more likely to be surrounding one or the other in the ambiguous cases.

This type of ambiguity is ideal in the sense that either POS or lemma disambiguation can be used to effectively solve it.

Some examples of this type of ambiguity are:

[8] `Nom`: nominative case, `Prt`: past tense (preterite).

[9] `Sg2`: second person singular.

Language	Surface	Lemma	POS	Translation
Finnish	tuli	tuli	Noun	“fire”
		tulla	Verb	“he came”
Russian	стали	сталь	Noun	“steel”
		стать	Verb	“they became”
Spanish	vino	vino	Noun	“wine”
		venir	Verb	“he came”

Different POS, same lemma For this type of ambiguity, we can consider two different cases, depending on the POS of the ambiguous readings:

1. If the POS have the same inflectional paradigm.
2. If the POS have a different inflectional paradigm.

An example of case 1 is the Finnish word *kuusi* (“spruce” / “six”):[10]

```
kuusi+N+Sg+Nom
kuusi+Num+Sg+Nom
```

Even though the lemma is the same, since they are both declinable words and their pattern of inflection is different, there are many unambiguous surface forms for each lemma.

For instance, the genitive case (meaning “of” or indicating possession) for the noun *kuusi* (“spruce”) is *kuusen*, and it is unambiguous:[11]

```
kuusi+N+Sg+Gen
```

In contrast, the genitive case for the numeral *kuusi* (“six”) is *kuuden*, and also unambiguous:

```
kuusi+Num+Sg+Gen
```

Therefore, relabelling each lemma—for example, as $\text{kuusi}_{\text{Noun}}$ and $\text{kuusi}_{\text{Num}}$—is enough to make ambiguities in this case equivalent to the previous type (different POS, different lemma), and solvable by either POS or lemma disambiguation.

[10] `Num`: numeral.

[11] `Gen`: genitive case.

This case is particularly common in Russian, where there are many nouns which have the same lemma as a verb, and nouns and verbs have a very different set of morphological features.

For case 2, this approach is ineffective. An example of this which appears frequently in Finnish is words which can be nouns or adjectives, such as colours. Consider the analyses for *punainen* ("red"):[12]

```
punainen+A+Sg+Nom
punainen+N+Sg+Nom
```

Nouns and adjectives in Finnish have exactly the same morphological features and inflectional morphemes modify the lemma in exactly the same way. Therefore, these ambiguities can only be solved via POS disambiguation.

Examples:

Language	**Surface**	**Lemma**	**POS**	**Translation**
Finnish	kuusi	kuusi	Numeral	"six"
		kuusi	Noun	"spruce"
Russian	знать	знать	Verb	"to know"
		знать	Noun	"nobility"
Spanish	parecer	parecer	Verb	"to seem"
		parecer	Noun	"opinion"

Same POS, different lemma If the lemmas are different, then in general the surface forms they produce will only partially overlap, so, for this type of ambiguity, we expect to have many unambiguous instances of each lemma.

Since the ambiguous readings have the same POS, we cannot use POS disambiguation to solve the ambiguity. Thus, for this type we have to resort to lemma disambiguation.

It is the only type where such method is the sole viable approach and, despite being far less common than the other major types, it is equally interesting for our purposes. This type is therefore the reason why we explore lemma disambiguation at all.

Examples:

[12] `A`: adjective.

Language	Surface	Lemma	POS	Translation
Finnish	palaa	palaa	Verb	“to return”
		palata	Verb	“he burns”
Russian	белку	белка	Noun	“squirrel”
		белок	Noun	“protein”
Spanish	fui	ser	Verb	“I was”
		ir	Verb	“I went”

Same POS, same lemma If the lemma and POS are the same and both words are declinable, then there is only feature ambiguity. This is the corner case where neither POS nor lemma disambiguation suffice to solve the morphological ambiguity, and so our approach is ineffective.

This type is less relevant for most NLP tasks, but worth exploring for some more advanced ones, such as dependency parsing—i.e., finding syntactic relations within a sentence—where, for example, the case of a noun tells us if it is a direct object, or the person of a verb indicates which pronoun accompanies it.

Examples:

Language	Surface	Lemma	POS	Translation
Finnish	nostaa	nostaa	Verb	“to raise”
		nostaa	Verb	“he raises”
Russian	кота	кот	Noun	“of the cat”
		кот	Noun	“the cat”
Spanish	come	comer	Verb	“he eats”
		comer	Verb	“eat, you”

2.3.2 Ambiguity between a declinable and an indeclinable word

Different POS, different lemma This type is particularly common in Finnish, where many adverbs or post-positions originate historically from an inflected form of a semantically related noun. As an example, consider the Finnish surface form *jälkeen* (“into the footprint” / “after”), which has two readings:[13]

```
jälki+N+Sg+Ill
jälkeen+Po
```

[13] `Ill`: illative case, `Po`: postposition.

We get two lemmas, *jälki* (“footprint”) and *jälkeen* (“after”), and two POS, “noun” and “post-position.” The post-position etymologically comes from the illative case (meaning “into” or “toward”) of the noun, in the sense that “after” is synonymous with “in the footsteps of.” However, this form has become its own lemma and diverged slightly from its original meaning by being commonly used figuratively, taking on a more general and abstract meaning.

At first glance, it might seem like this ambiguity can be solved by predicting the lemma. However, in practice, we will never find unambiguous instances of the lemma for the indeclinable word—in this case, *jälkeen*—thus making it impossible to model the context for said lemma. We can know how likely it is that a given context surrounds the declinable word—since there are many unambiguous forms of *jälki*—but without a reference for how likely the indeclinable word is in comparison, we cannot say with confidence whether it is one or the other.

Therefore, the only viable approach for this ambiguity using our method is to disambiguate the POS, since there will be plenty of instances of other unambiguous words with the same POS.

Examples:

Language	**Surface**	**Lemma**	**POS**	**Translation**
Finnish	jälkeen	jälki	Noun	“into the footprint”
		jälkeen	Postposition	“after”
Russian	для	длить	Verb	“while delaying”
		для	Preposition	“for”
Spanish	cabe	caber	Verb	“he fits”
		cabe	Preposition	“next to”

Different POS, same lemma This type of ambiguity is very uncommon—the rarest in Finnish, as shown in Table 2. Many of the instances are nouns which can be used as particles or interjections, such as Finnish *helvetti* (“hell”), which can be used as an expression of surprise or annoyance. For many NLP tasks, it would be reasonable to make a rule that removes the indeclinable reading for most of these words. There are, however, a few words for which this ambiguity is significant on a morphosyntactic level.

As with the previous type, lemma disambiguation is infeasible, and so these ambiguities must be solved by disambiguating the POS.

Examples:

Language	Surface	Lemma	POS	Translation
Finnish	aika	aika	Noun	“time”
		aika	Adverb	“quite”
Russian	уж	уж	Noun	“grass snake”
		уж	Adverb	“already”
Spanish	sobre[14]	sobre	Noun	“envelope”
		sobre	Preposition	“over”

Same POS As we mentioned in 2.1, the POS determines whether the word is declinable or indeclinable. Thus it is impossible to have the same POS in an indeclinable and a declinable word, and this type of ambiguity does not exist.

2.3.3 Ambiguity between two indeclinable words

Different POS, same lemma This is a somewhat prevalent type of ambiguity across many languages, since short, indeclinable words tend to be reused for several different purposes in a sentence. For instance, in Finnish, many post-positions can also function as adverbs—such as *kanssa* (“with” / “also”), which can accompany a noun, as a post-position meaning “with,” or a verb, as an adverb meaning “also.”

Due to this, this type can be considered extra-morphological, since the difference between POS which do not accept inflections is purely syntactical—and therefore it is essentially outside the scope of this work.

As with almost all other ambiguities where the lemma is the same, this type is only solvable through POS disambiguation.

Same POS, same lemma If the lemma and POS are the same, the words must be trivially the same word, since there are no other morphological features—and so there is always only one reading for the word, and no ambiguity.

Different lemma If the words are both indeclinable—neither can be inflected—and if their lemmas are different, there is no ambiguity, as the surface forms will always differ as well. Therefore, similarly to the previous type, this type never occurs in practice.

[14] Also a form of the verb *sobrar* (“to be in excess”).

	Declinable-Declinable	Declinable-Indeclinable	Indeclinable-Indeclinable
≠ POS = lemma	POS disamb.	POS disamb.	POS disamb.
= POS ≠ lemma	lemma disamb.	n/a	n/a
≠ POS ≠ lemma	either	POS disamb.	n/a
= POS = lemma	neither	n/a	n/a

Table 1: Viable approaches for each type of ambiguity.

	Declinable-Declinable	Declinable-Indeclinable	Indeclinable-Indeclinable
≠ POS = lemma	8.78%	1.63%	6.47%
= POS ≠ lemma	8.93%	0.00%	0.00%
≠ POS ≠ lemma	40.29%	27.88%	0.00%
= POS = lemma	6.04%	0.00%	0.00%

Table 2: Incidence of each type of ambiguity in the Finnish corpus.

3 Previous work

3.1 Rule-based models

One of the simplest and most intuitive ways to approach the problem of morphological disambiguation is by manually writing rules which look at the context of a word in order to decide which is its correct analysis.

One of the earliest and most widely used frameworks for writing disambiguation rules is Fred Karlsson's constraint grammar specification [13].

The input to a constraint grammar is a sentence where every word has been morphologically analysed. With this information, the rules of the grammar remove impossible readings or select the correct reading for each word, based on conditions within the context.

For instance, suppose we have the following rule:

```
SELECT (Sg3) IF (-1 (Pron Sg3)) (0 (V));
```

This rule has two conditions, `(-1 (Pron Sg3))`, which is true if the preceding word has a singular third person pronoun reading, and `(0 (V))`, which is true if the current word has a verb reading. The result of this rule is to select the reading with the tag `(Sg3)` if both conditions hold true.

Suppose we have the ambiguous phrase *hän tuli* ("he came"). The input to the rule would be all the possible readings for each word:[15]

```
hän+Pron+Pers+Sg3+Nom

tuli+N+Sg+Nom
tulla+V+Act+Ind+Prt+Sg3
```

Applying the previous rule on this phrase, we would select the verbal reading for "tuli," since we have a reading with the tag `Sg3` in the current position, and a reading with the tags `Pron` and `Sg3` in the previous position.

Note that constraint grammars are a very powerful paradigm and allow us to do other tasks such as syntactic annotation; these rules for disambiguation are just a small subset of the functionality they provide.

[15] `Pron`: pronoun, `Pers`: personal pronoun.

As with any other approach using handmade rules, there are several advantages and disadvantages.

The main advantages are:

- They are human-readable: one can understand the rationale behind each decision made by the model.
- They behave consistently, i.e., every instance which matches a rule will produce the same result.
- Their behaviour can be easily adjusted to account for exceptions to a rule or previously uncovered cases.
- They can have an arbitrarily high precision, given rules with enough complexity and ability to generalize.

In addition, a set of rules can be used as a pre-processing step to augment any other approach, in order to solve cases which are very clear-cut, such as the previous example where a pronoun agrees in person with a verbal reading.

However, they also come with a host of disadvantages, such as:

- Writing rules is a very labour-intensive process, compared to automatic procedures.
- Expert knowledge is necessary to write the rules—they require both extensive knowledge of linguistics and of the rule language specification.
- The coverage of the set of rules—that is, the number of cases which are picked up by the rule conditions—is limited. In order to have perfect coverage, one would have to write a set of rules that covers the entire grammar for a language, as well as exceptions to grammatical rules.

There exist several approaches to learning these rules automatically in order to alleviate the first problem, which is arguably the biggest one. One such approach is by Šmerk [32], which uses inductive logic programming to learn the set of disambiguation rules.

Rules are learned by optimizing two criteria: the rules must cover the largest possible number of positive examples and the smallest possible number of negative

examples. Examples are taken from unambiguous words in a corpus which has been analysed by a morphological analyser.

One of the merits of this work is that it shows that there is a strong overlap in context between ambiguous words and unambiguous words which fulfil a similar role in the sentence. That is, an *unambiguous word* which is *similar* to an *ambiguous* word will appear in a *similar context*, thus making it possible to learn to disambiguate just by looking at the context of unambiguous examples in a corpus. This is one of the main assumptions made by most unsupervised approaches to morphological disambiguation, including our work.

However, by using an automatic procedure instead of a manual one, the possibility of errors (i.e., unaccounted for exceptions to a general rule) is introduced, making it a trade-off with regard to manual rule writing. Additionally, the rule format must be manually specified—therefore, domain knowledge is still required to build the model.

3.2 Probabilistic models

Probabilistic models work by capturing statistical dependencies between words within some unit of language—usually a sentence or document.

A very widely used probabilistic model is the Hidden Markov model (HMM), which is designed for sequential data. Let us assume that we want to use an HMM for POS tagging—that is, given a sequence of words $W = \{w_1, w_2, \ldots, w_n\}$, find the sequence of tags $T = \{t_1, t_2, \ldots, t_n\}$ corresponding to each word. Thus, W would be the sequence of *observed* states (the ones we know) and T would be the sequence of *hidden* states (the ones we do not know).

In order to make the problem tractable, two assumptions are made: that the current hidden state depends only on the previous hidden state, and that the current observed state only depends on the current hidden state. This is known as the Markov assumption.

First, we need to train the model by learning two sets of parameters:

- The *transition* probabilities—that is, the probability that, being in some hidden state t_i, the next hidden state will be some other state t_j. In our case, this would be the probability of finding some POS following the current one.
- The *emission* probabilities—that is, the probability that the current hidden

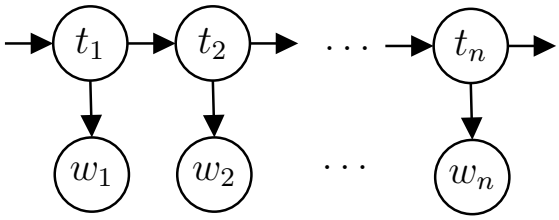

Figure 1: An HMM for POS tagging. Arrows indicate dependencies: each hidden state only depends on the previous one, and each observed state depends only on the hidden state.

> state t_i produces some observed state w_i. In our case, this would be the probability of having each word given that our POS is some value.

Once the model is trained, inference is performed by picking the sequence of hidden states T that maximizes the probability of having produced the observed states W using dynamic programming, e.g., the Viterbi algorithm [6].

Figure 1 shows a diagram for an HMM with these characteristics.

Due to these assumptions, HMMs cannot capture long-term dependencies. Furthermore, they require annotated data for training, and inference tends to be much slower than other methods (for large inputs). However, they still achieve very high precision, and were the *de facto* standard for POS tagging before the advent of neural networks.

An example of a relevant work using an HMM is that by Hakkani-Tür et al. [9]. It tackles the problem of morphological disambiguation for Turkish, which is a language with similar characteristics to Finnish—such as a high morpheme-to-word ratio and free word order.

In the above work, two models are built: one which uses sets of morphological tags as the input, and one which uses only the lemma and POS of each context word. They work with the assumption that both models are independent: morphological tags are defined by the syntactic context of a word, whereas lemmas depend on the appearance of other lemmas in the sentence. They obtain results which show that such assumption holds true to a large extent.

Another statistical approach to morphological disambiguation of Turkish is the one by Yatbaz and Yuret [30]. They first build a language model, which is a function that takes in a piece of linguistic input (word, sentence, document) and assigns a probability to it. This language model is then used to obtain the probability that a reading appears in a given context.

Such probability is estimated by computing the probability of a set of replacement words having the same POS and morphological features. The context is modelled as a window of words centred around the target word—thus utilizing words appearing after the target word, unlike the previous method.

Their approach does not require manually annotated data, relying exclusively on information given by the analyser and a large, unlabelled corpus. They obtain results comparable to a supervised baseline.

3.3 Neural models

In the past few years, neural networks have dominated the research in NLP. In particular, deep learning models have surpassed the high accuracy of more traditional models in areas such as POS tagging, which had previously seen little improvement for many years.

The most commonly used neural network model for this purpose is the bidirectional Long Short-Term Memory architecture, which we will describe in detail in Section 4.2, since it is the same model we use in our work. All the following works referenced in this section use this same architecture. Furthermore, these works use information from the morphological analyser, in order to constrain the predictions for each word to the choices given by the analyser.

For Arabic, the work by Zalmout and Habash [31] uses independent models for predicting different morphological features. For each word, each model predicts a feature based on the context. If the feature is found in one of the possible readings for the word, the score for that reading is increased. Then, the reading with the highest score is selected as the most likely one.

They test several input data representations and conclude that character-level representations result in an increase in accuracy over word-level representations.

The approach by Inoue et al. [11]—also for Arabic—is similar, but improves on the previous method by using multi-task learning. Instead of trying to predict each feature individually, the model performs a joint prediction which exploits the dependencies between features. They accomplish this by reusing model parameters across different models.

For Estonian, the work by Tkachenko and Sirts [29] uses a similar architecture to ours. They use dense representations for the context and target word which are

then encoded into a single state vector. Then, they try two different models: a simple softmax classifier over the state vector, and a decoder architecture which generates a sequence from the encoded state. Their results for each model are basically identical, with the former model being significantly simpler.

The overall results for each method show that current neural networks outperform previous methods while having significant advantages:

- They require very little or no feature engineering, that is, the model automatically extracts the features it needs from raw data.
- They do not require domain knowledge on the input data.
- They do not require us to make assumptions on how the data and the solution are structured, as with rule-based or probabilistic approaches.

They do, however, come with several disadvantages, some of them being:

- The models are effectively black boxes: in most cases it is impossible to know how some prediction was computed.
- Training is very slow and in most cases requires specialized hardware.
- They require a large amount of annotated data for training.

In our work, we aim to address this last issue, by developing a method which does not require manually annotated data.

4 Methodology

4.1 Data pre-processing

4.1.1 Tokenization

Tokenization is the task of segmenting a string of characters into its minimal relevant sub-strings for some purpose, called **tokens**. Our purpose is to transform an input string—which represents an entire document—into a list of strings which can be parsed by the morphological analyser. Thus, we consider tokens to be the surface forms of words, punctuation marks, acronyms, symbols or abbreviations—in short, any string which has meaning on its own.

For example, the sentence *Hän tuli kotiin.* ("He came home.") is given to the tokenizer as a single string:

Hän tuli kotiin.

And the tokenizer should return a list of tokens:

Hän	tuli	kotiin	.

In order to produce useful tokens, the tokenizer has to correctly identify the word boundaries within the string. For many languages, these are mostly indicated by white spaces, but some cases are non-trivial—consider, for instance, the English string "don't," which is composed of two tokens: "do" and "not."

Furthermore, abbreviations are usually followed by a full stop, which should not be split from the rest of the token, since it does not function as a punctuation mark. For example, the Finnish abbreviation "t.," short for *terveisin*, ("regards," as in the signature of a letter) should be a single token, and not split into "t" + ".." However, if the abbreviation is the last word in the sentence, then one of the full stops is omitted for readability, and the tokenizer should recover it.

Other cases where a full stop might not indicate a word or sentence boundary are numbers with decimals or dates in the standard Finnish format `DD.MM.YYYY`.

In addition, Finnish words are not always separated by a white space, and some punctuation marks like the colon have two purposes: either as a sentence separator, or as a way of attaching an inflectional ending to a number or acronym. For

example, *EU:n* ("of or pertaining to the EU") is a single token, whereas *EU-maat* ("EU countries") is composed of two tokens, *EU* and *maat*.

Other languages (e.g., Mandarin) may not at all have explicit word boundaries—such as the white space—and thus tokenization should be done based on contextual information. These languages, however, are out of the scope of this work.

For our Finnish training corpus, we use the Turku OpenNLP model for tokenization [12], which is a statistical model trained in order to deal with these problematic cases, and thus in principle better than a simpler approach such as regular expressions.

The Finnish evaluation corpus and the Russian and Spanish training and evaluation corpora are already given in tokenized form.

4.1.2 Analysis

After tokenizing the input text, we proceed to analyse each token with a morphological analyser.

For Finnish, we use the analyser from the Giellatekno platform[16] [20].

For Russian, we use the Crosslator analyser [16].

For Spanish, we use the analyser from Apertium[17] [5].

Since the goal is to disambiguate the output of the analyser, the coverage of said analyser—the percentage of tokens that have an analysis—is a relevant concern. The Finnish, Russian and Spanish analysers have 95.14%, 97.79% and 96.78% coverage on the training corpora, respectively. Most of the unknown tokens are names, foreign words or misspellings.

Each analyser has its own set of POS, since these differ slightly across languages. For example, Spanish has determiners, which Finnish and Russian lack; Finnish has postpositions, which are absent in Russian and Spanish. However, one can build a relatively small universal POS set (see for instance the Universal Dependencies set[18]), since most of the major POS are common to every language.

Moreover, as explained in Section 2.3, some ambiguities are not strictly morphological. There are groups of POS which tend to appear together for a single lemma

[16] `giellatekno.no`
[17] `apertium.org`
[18] `universaldependencies.org/u/pos`

but differ only in their syntactic role, if at all. Examples of this include lemmas which can be both postpositions and adverbs, participles and adjectives, or conjunctions and relative pronouns. We merge these POS, since they are outside the scope of morphological analysis and instead belong to the problem of syntactical parsing.

We define a set of 5 possible targets for the POS which captures all the relevant ambiguities:

- **Noun**: Nouns and noun-like POS, such as numerals or acronyms.
- **Adjective**: Adjectives and adjective-like POS, such as participles (in Russian).
- **Verb**: All types of verbs, including auxiliary verbs.
- **Adverb**: Adverbs and adverb-like POS, such as adpositions.
- **Other**: All other POS which do not fit in the previous categories—conjunctions, interjections, particles, etc.

These 5 POS make up the target set S_p.

For the lemmas, we build a set S_l of all unique lemmas in the language—or, in practice, the ones that appear in the corpus. We use the special tag `<num>` for numbers consisting solely of one or more digits and zero or more punctuation marks, and the tag `<unk>` as the lemma for surface forms for which the analyser does not give any readings.

Then, we build a vocabulary V as an enumeration $V : S \to \mathbb{N}_0$ over the target set S which assigns a natural number (including zero) to each element of the set.

The vocabulary V_p built on S_p for the POS model maps POS to unique indices:

$$V_p(\texttt{Noun}) = 0$$

$$V_p(\texttt{Adjective}) = 1$$

$$\ldots$$

The vocabulary V_l built on S_l for the lemma model maps lemmas to unique indices:

$$V_l(\texttt{<num>}) = 0$$

$$V_l(\texttt{<unk>}) = 1$$

$$V_l(\texttt{viikonloppu}^{19}) = 2$$

...

The image of V can therefore be considered the set of possible classes for each word, and the problem of disambiguation can be thought of as a classification problem, regardless of whether we are trying to predict the lemma or the POS for the word.

4.1.3 Training and testing instances

After tokenizing and analysing each document in the corpus, we proceed to generate the training and testing instances. Training instances consist of two parts: the context and the label for the target word. Testing instances consist of three parts: the context, the label and the set of possible labels.

We discard those testing instances in which the label is not in the set of possible labels, which happens occasionally due to discrepancies between the analyser output and the label manually assigned to a word—for instance, the person tagging the word might consider the given word to function as a preposition in a sentence, while the person who wrote the analyser considers that word to always be an adverb. We must also bear in mind that analysers may be incomplete, or have errors, and this might be the source of the discrepancy. We make several rules to try to resolve these, but a small proportion remain unresolved.

Context window The context is a list of surface forms which contains the target word and its neighbouring words in the document. We refer to this list as a "window," and its radius r is the number of surface forms to each side. To each position in the window that extends beyond the boundaries of the document we assign the special surface form `<pad>`. For example, given the input text *Hän meni kotiin.* ("He went home."), the context window with radius $r = 2$ and with target *meni* would be:

<pad>	Hän	**meni**	kotiin	.

We found that using a sliding window across the entire document was more effective than using whole sentences, since many sentences are very short and the

[19] *Viikonloppu* ("weekend") is the first lemma that appears in the corpus.

contextual clues needed to disambiguate the target word will be found in neighbouring sentences within the document. Besides, the model can have implicit knowledge of sentence boundaries by looking at the punctuation marks.

At this point, we can also improve the tokenization using information from the analyser readings. For Finnish—which has compounding—we split the surface form of the compounds into their "maximal" pieces, i.e., the largest parts for which there is a lemma in the analyser's lexicon.

For example, the Finnish compound word *eläinlääkäriasema* ("veterinary clinic") is made up of three elementary stems: *eläin* ("animal") + *lääkäri* ("doctor") + *asema* ("station").

However, since the analyser has *eläinlääkäri* ("veterinarian") in its lexicon, we split as *eläinlääkäri* + *asema*. This helps us keep the vocabulary smaller—since there is an extremely large number of possible compounds in Finnish—while keeping the meaning of commonly used compounds, which usually differs a little from that of the sum of its parts.

For Russian, this is not a concern, as there are generally no compound words. There may be cases in which a lemma is formed by joining two other lemmas, but this is considered a new lemma in its own right.

The same applies to Spanish, where in addition there can be clitic pronouns attached to verbs, as in *dímelo* ("tell me that"), formed by *di* ("say, you") + *me* ("to me") + *lo* ("that"), or prepositional contractions, as in *al*: *a* ("to") + *el* ("the"). These are separated by the analyser into individual tokens.

Label The label for an instance is the vocabulary index of the POS or lemma, depending on which model we are training. The set of possible labels is the vocabulary index for each POS or lemma given by the analyser in an ambiguous word.

In the previous example, since *meni* ("he went") is unambiguous, the label would be the index for *mennä* ("to go"), $V_l(\texttt{mennä})$, or the index for "Verb," $V_p(\texttt{Verb})$. The context and this label would be used as a training instance.

In the example given in Section 4.1.1, *Hän tuli kotiin.*, since the target word *tuli* ("he came") is ambiguous, the set of possible labels for the lemma model is the set of indices for its two possible lemmas *tulla* ("to come") and *tuli* ("fire"), $\{V_l(\texttt{tulla}), V_l(\texttt{tuli})\}$, and for the POS model it is the set of indices for its two

possible POS, "Verb" and "Noun," $\{V_p(\texttt{Verb}), V_p(\texttt{Noun})\}$. Being an ambiguous word, we require supervision to know that the true label is $V_l(\texttt{tulla})$ for the lemma or $V_p(\texttt{Verb})$ for the POS. Therefore, this instance would be used only as a testing instance, if found in the manually annotated corpus.

With the two previous examples, we can see the main intuition behind our approach: even if *tuli* is ambiguous, if we find another instance with a similar context but an unambiguous target word, such as *meni*, then the model can learn that the context *Hän _ kotiin.* probably surrounds a verb, and it probably surrounds a word semantically close to *meni*. In contrast, it is very unlikely that such context surrounds a noun or a word semantically close to *tuli* ("fire"). Using this information, we can predict both POS and lemma, respectively.

Ambiguities between common nouns and proper nouns are ignored, as names are out of the scope of what we try to accomplish here and should be solved using Named Entity Recognition (NER) techniques instead.

4.1.4 Word embeddings

Up to this point, our window consists of surface forms, represented as strings of characters. In order to work with these inside a neural network, we need to find a meaningful way to represent them as real-valued vectors. This is where word embeddings come in.

First, we build a vocabulary V over the set of possible input strings, as in the previous section. Next, we take our input string s and make a one-hot vector x of length $|V|$, where the element with index equal to $V(s)$ has a value of 1 and the rest have a value of 0:

$$x_i = \begin{cases} 1 & \text{if } i = V(s) \\ 0 & \text{otherwise} \end{cases}$$

An embedding function $\mathcal{E} : \{0,1\}^{|V|} \to \mathbb{R}^d$ maps x to a real-valued vector $\mathcal{E}(x)$ of length d. Since the domain of $\mathcal{E}$ is extremely sparse and each vector is independent (all distinct one-hot vectors are orthogonal), the goal is to have $d \ll |V|$ and, at the same time, have meaningful dependencies between vectors.

We use FastText [2] as the embedding function, a modification of the earlier word2vec [19] architecture. In particular, we are using the pre-trained Common Crawl[20] embeddings [8] and we do not adjust the parameters of $\mathcal{E}$ at all during

[20] `commoncrawl.org`

training—which is why we consider obtaining the embeddings to be part of the pre-processing and not part of our model architecture.

The main idea behind FastText is to break down the strings into character n-grams, and learn an embedding for those instead. The embedding for the entire string is then computed as the sum of the embedding for each of its n-grams.

Thus, we only need to build a vocabulary of n-grams V_g which is bounded in size by the number of distinct characters c in the language and the value of n $|V_g| \leq c^n$. Given that $c \approx 30$ in our tested languages and typically $3 \leq n \leq 5$, $|V|$ is relatively small. Furthermore, only certain character combinations are allowed in each language, which means that, in practice, $|V|$ is much smaller than the theoretical upper bound.

The benefit in doing so is three-fold:

1. We can get embeddings for surface forms for morphologically rich languages, which would otherwise prove infeasible due to memory constraints; in Finnish, the number of possible surface forms is very large due to the vast number of potential inflectional combinations for any word, and due to the possibility of making arbitrarily complex compound words.

2. The embeddings capture morphological features, by learning embeddings for common inflectional affixes.

3. It allows us to obtain an embedding for out-of-vocabulary words, such as misspelled words, neologisms, etc. This embedding will be close to that of existing words with a similar set of n-grams, which is usually a reasonable guess.

FastText uses the special characters < and > to denote the beginning and the end of the word, respectively. As an example of how an embedding is obtained, consider the surface form *tuli* and $n = 3$.

The set of 3-grams for *tuli* would be $\{\texttt{<tu}, \texttt{tul}, \texttt{uli}, \texttt{li>}\}$. Let x_s be the one-hot vector where the value at index $V_g(s)$ is 1 and the rest are 0. The embedding for *tuli* would then be $\mathcal{E}(x_{\texttt{<tu}}) + \mathcal{E}(x_{\texttt{tul}}) + \mathcal{E}(x_{\texttt{uli}}) + \mathcal{E}(x_{\texttt{li>}})$.

The training method for FastText is out of the scope of this work, but it is analogous to that of our own model, described in the following section.

For the special token <pad>, we return a zero-valued embedding, which will go

through the neural network without affecting the output or the network parameters.

4.2 Model architecture

Throughout this section, we will use the term *layer* to denote the basic building blocks of a neural network.

Let $x \in \mathbb{R}^n$ be the input and $L(x) \in \mathbb{R}^m$ the output of the layer. A network layer $L : \mathbb{R}^n \to \mathbb{R}^m$ consists of the following elements:

- A *weight* matrix $W \in \mathbb{R}^{n \times m}$.
- A *bias* vector $b \in \mathbb{R}^m$.
- An element-wise nonlinear *activation function* $a : \mathbb{R}^m \to \mathbb{R}^m$.

The weight matrix and bias vector define a linear function $\ell : \mathbb{R}^n \to \mathbb{R}^m$:

$$\ell(x) = Wx + b$$

The output of the layer is thus:

$$L(x) = a(\ell(x))$$

W and b are the *trainable parameters* of the layer—that is to say, the values that will be updated during training. They are initialized randomly and adjusted at each step in order to minimize some *objective function*. We will further discuss the training procedure in Section 4.2.3.

4.2.1 Long Short-Term Memory

The Long Short-Term Memory architecture (LSTM) is a type of recurrent neural network (RNN). Unlike static networks like the MLP where the input is evaluated for one point in time, RNNs allow us to process sequential data by updating the network parameters at each time step.

For this work, we use the LSTM architecture described by Gers et al. [7].

LSTM Cell An LSTM cell consists of two state vectors—the *hidden state* or output vector h and the *cell state* vector c—and three "gates": the *input gate* i, the *output gate* o and the *forget gate* f.

The input to each of the gates at each time step t is the concatenation of the input vector for that step x_t and the hidden state of the previous time step h_{t-1}, $x_t \mathbin{+\!\!+} h_{t-1}$.

The forget and output gates each consist of a single layer with an activation function σ:

$$f_t = L_f(x_t \mathbin{+\!\!+} h_{t-1}) = \sigma(W_f(x_t \mathbin{+\!\!+} h_{t-1}) + b_f)$$
$$o_t = L_o(x_t \mathbin{+\!\!+} h_{t-1}) = \sigma(W_o(x_t \mathbin{+\!\!+} h_{t-1}) + b_o)$$

where $\sigma : \mathbb{R} \to [0, 1]$ is the standard logistic function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

The input gate consists of two separate layers, one with σ activation and one with tanh activation:

$$\begin{aligned} i_t &= L_i(x_t \mathbin{+\!\!+} h_{t-1}) \odot L'_i(x_t \mathbin{+\!\!+} h_{t-1}) \\ &= \sigma(W_i(x_t \mathbin{+\!\!+} h_{t-1}) + b_i) \odot \tanh(W'_i(x_t \mathbin{+\!\!+} h_{t-1}) + b'_i) \end{aligned}$$

where $\tanh : \mathbb{R} \to [-1, 1]$ is the hyperbolic tangent function:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

and $\odot$ is the Hadamard (element-wise) product for matrices.

The rationale behind this architecture is that, at each step, the forget gate removes values from the cell state—hence the product with values in $[0, 1]$, which is the codomain of σ—and the input gate updates the state with new values—by summing values in $[-1, 1]$, which is the codomain of tanh. This allows the cell to selectively forget or remember the most useful pieces of information. Next, the information from the cell state is incorporated into the current hidden state by use of the output gate.

Figure 2 shows the internal layout of an LSTM cell.

Bidirectional LSTM A bidirectional LSTM (BiLSTM) consists of two separate LSTM cells: a left-to-right cell $LSTM^L$ which receives the input in its original

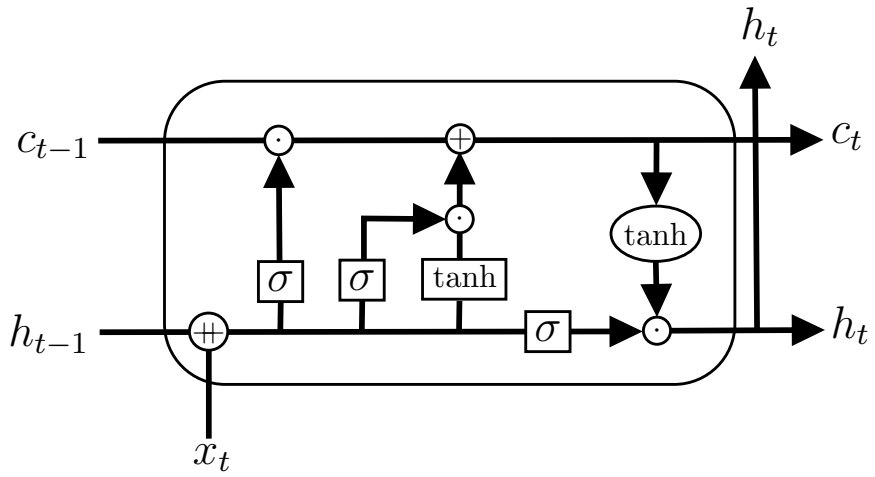

Figure 2: An LSTM cell. Rectangles represent network layers and ellipses represent element-wise operations.

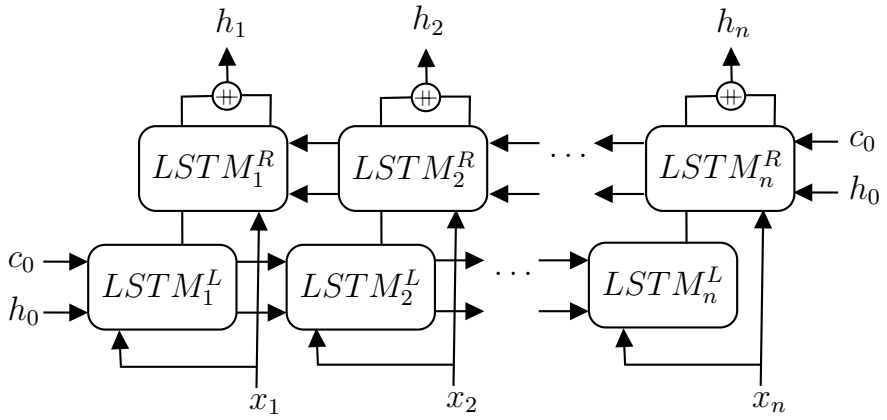

Figure 3: A BiLSTM network.

order, and a right-to-left cell $LSTM^R$ which receives the input in reversed order—that is, starting with the last element and ending with the first element. At each step t, the hidden state of $LSTM_t^L$ is concatenated with the hidden state of $LSTM_t^R$ to produce a unified hidden state $h_t = h_t^L + h_t^R$.

Figure 3 shows the layout of a BiLSTM network.

4.2.2 Multilayer perceptron

A multilayer perceptron (MLP) is a type of feed-forward neural network—that is, the connections only go from the input to the output, without any feedback loops. Therefore, unlike RNNs, an MLP only produces output for a fixed point in time.

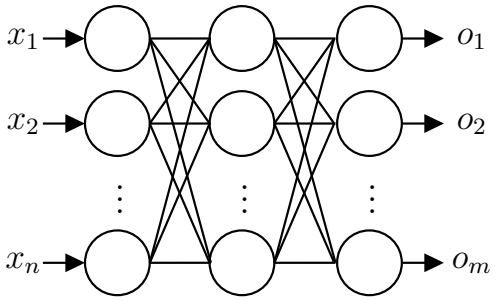

Figure 4: An MLP with n inputs, m outputs and one hidden layer.

The MLP is one of the simplest neural network architectures. It consists of at least two layers: one or more *hidden* layers, and an *output* layer. Each layer has its own weight matrix, bias vector and activation function, and the output of the MLP is the composition of all its layers. For instance, an MLP with a single hidden layer L_h and an output layer L_o will give, for an input x, the following output:

$$MLP(x) = L_o(L_h(x))$$

Provided the activation function of at least one hidden layer is nonlinear, the MLP is a universal approximator—which means it can theoretically model any function [10]. It is important to note that, since the composition of linear functions can be expressed as a single linear function, it is unproductive to use several hidden layers with linear activation.

Figure 4 shows the layout of an MLP.

For our model, we use LeakyReLU [18] as the activation function. Unlike the standard ReLU function, which assigns a value of zero to all negative input values, LeakyReLU lets a small portion of the value through. This helps with a common problem with multi-layer neural networks, where the updates to the network weights become increasingly small due to multiplying the gradients of output values close to or equal to zero. This problem is known as the vanishing gradient problem [1].

The LeakyReLU function is defined as follows.

$$LeakyReLU(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

4.2.3 Objective function

Let x be our input. Let V be the output vocabulary—i.e., the set of all possible labels. Let $j \in V$ be the true label for x. The true probability distribution $y \in \{0,1\}^{|V|}$ of x is a one-hot vector where the index corresponding to the true class has a value of 1, and the rest have a value of 0:

$$y_{t_i} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

To obtain the predicted probability distribution $\hat{y}$, we take the output of the neural network $z \in \mathbb{R}^{|V|}$ and apply the $softmax$ function $softmax : \mathbb{R}^{|V|} \to [0,1]^{|V|}$. The softmax function is a generalization of the standard logistic function for vectors:

$$softmax(z)_i = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

The softmax function takes a real-valued vector and normalizes its values such that $z_i \in [0,1]\ \forall z_i \in z$ and $\sum_i softmax(z)_i = 1$—that is, it turns vector z into a probability distribution.

The objective function for the network is the cross-entropy H of the true probability distribution y and the predicted distribution $\hat{y}$ for an instance x. This gives us a *loss* value $\mathcal{L}$ corresponding to that instance.

$$\mathcal{L} = H(y, \hat{y}) = -\sum_i y_i \log \hat{y}_i$$

In our case in particular, since y is 1 for the true label j and 0 for all other labels, $L = -\log \hat{y}_j$.

To train the model, we need to compute the gradient of H with respect to each of the trainable parameters. Since our network is simply a composition of functions, we can apply the chain rule of derivatives. The gradient of a layer L with respect to the input x is given by:

$$\frac{\partial L(x)}{\partial x} = \frac{\partial a(\ell(x))}{\partial \ell(x)} \cdot \frac{\partial \ell(x)}{\partial x}$$

The gradient of the loss $\mathcal{L}$ for a given instance x in a k-layer network is thus:

$$\nabla \mathcal{L}(x) = \frac{\partial (L_k \circ \cdots \circ L_0)(x)}{\partial x} = \frac{\partial L_k(x)}{\partial L_{k-1}(x)} \cdots \frac{\partial L_0(x)}{\partial x}$$

To train the network, at each time step t and for each trainable parameter θ, we compute the gradient of the loss $\mathcal{L}$ with respect to θ, weighted by some learning rate α, and subtract the result from the value of the parameter:

$$\theta_t = \theta_{t-1} - \alpha \nabla_\theta \mathcal{L}(x_t)$$

This method of optimization is commonly known as gradient descent. More specifically, we use *stochastic* gradient descent, where the gradient is only computed for a subset (a *batch*) of the set of training instances.

We use Adam [15] as the optimization algorithm. In contrast to standard stochastic gradient descent, Adam adapts the learning rate over time, so as to make coarse adjustments to the parameters at the beginning and do finer adjustments toward the end—thus allowing us to quickly have a reasonable estimate of the model's performance.

4.2.4 Network layout

In this section, we give an overview of the inference and training procedures for the network.

Let x be our input window of surface forms, with radius r and total length $w = 2r + 1$. Let x_r be the target word at the centre of the window. Let j be the true label for x_r, whence we obtain the true distribution for the instance, y.

Firstly, we obtain the surface form embeddings $\mathcal{E}(x_i)\ \forall i \in [0, w]$.

Secondly, we feed the embeddings to the BiLSTM to get the hidden state vectors at step r h_r^L and h_r^R for the left-to-right and right-to-left LSTMs, respectively:

$$h_r^L = LSTM_r^L(\mathcal{E}(x_{0:r}))$$
$$h_r^R = LSTM_r^R(\mathcal{E}(x_{w:r}))$$

Next, we concatenate the left and right hidden state vectors with $\mathcal{E}(x_r)$ to obtain a single hidden state vector h_r:

$$h_r = h_r^L +\!\!+ \mathcal{E}(x_r) +\!\!+ h_r^R$$

Then, we feed the hidden state vector to the MLP to obtain the network outputs z:

$$z = MLP(h_r)$$

If we want to use the model to infer the predicted label $\hat{j}$ for an unambiguous instance, we find the index with the maximum value in z among the set of k possible labels $J = \{j_0 \dots j_k\}$ which are returned by the analyser:

$$\hat{j} = \arg\max_{j_i \in J} z_{j_i}$$

For inference, no further steps are necessary. If we are training the model, we proceed to obtain the predicted distribution $\hat{y}$ by applying the $softmax$ function to z:

$$\hat{y} = softmax(z)$$

Finally, we obtain the loss $\mathcal{L}$ for the instance:

$$\mathcal{L}(x) = H(y, \hat{y})$$

And we update the network parameters as shown in Section 4.2.3. Note that the parameters of the embedding function $\mathcal{E}$ are not updated, and thus remain constant throughout the training.

In addition, we use dropout [28] during training as a form of regularization. With dropout, instead of always updating all the parameters, we set the gradient to 0 for any given parameter with some probability p. This helps avoid complex dependencies between parameters and train a model that generalizes better.

Figure 5 shows a diagram of the network layout.

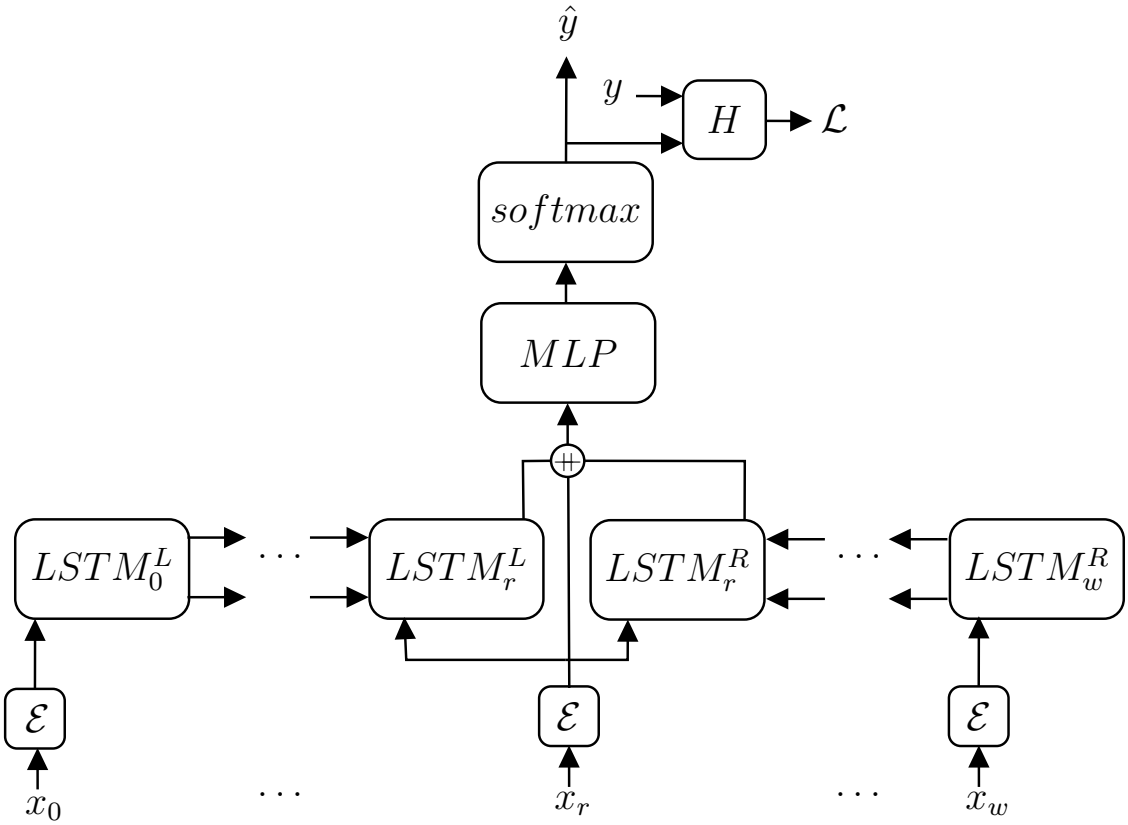

Figure 5: Overview of the network (see Section 4.2.4).

5 Evaluation

5.1 Experimental setup

For the training and evaluation data sets, we use data from the Universal Dependencies Treebank [22]. These data are in the CoNLL-U 2006/2007 format [21] and consist of automatically annotated and manually corrected sentences from various sources, mainly newspapers and magazines. They are annotated for morphology and syntax, although we only use the morphological annotations.

We map their POS set to ours, and use the unambiguous words for the training set and the ambiguous words for the evaluation set, as explained in Section 4.1.3.

For Finnish, as the number of annotated instances is quite small, we use a collection of 1M proprietary news articles to generate the training set.

For each language, we train two separate models: one to disambiguate the POS, and one to disambiguate the lemma. For each model, we obtain two different predictions:

- The **guided** prediction picks the output with the highest probability among the possible classes given by the analyser. For example, for the POS model, if the analyser says that a surface form can be analysed as a noun or a verb, it will pick only from those two classes.
- The **blind** prediction simply picks the output with the highest probability among all classes. For example, for the POS model, it would pick from the set of all POS. It therefore gives an output equivalent to that of plain, uninformed POS tagging or lemmatization.

The blind prediction is there both as a sanity check—to verify that the model is learning correctly, and not simply picking randomly from the reduced set of classes—and to measure the accuracy of the model on out-of-vocabulary words—i.e., those for which the analyser does not return any readings.

Evidently, the guided prediction accuracy is an upper bound on the blind prediction accuracy: if a class is deemed to be the most probable, the model will choose it from the set of classes given by the analyser anyway. So, to obtain the best results during inference, we should only consider the guided prediction.

Window size	21
Embedding size	300
LSTM hidden units	512
LSTM layers	1
MLP hidden neurons	1024

Table 3: Settings for the network.

Batch size	50
Dropout rate	0.1
Adam β_1	0.9
Adam β_2	0.999
Adam α	0.0001
Adam ϵ	0.001
Epochs	20

Table 4: Settings for the training hyper-parameters.

Table 3 details the settings used for the network. We set a relatively small network size due to hardware constraints. It is possible that with a more complex model the prediction accuracy could be slightly higher, especially for the lemma model, given the much larger number of possible classes.

Table 4 details the training hyper-parameters. We found that, given the amount of data and network size, the differences in prediction accuracy were minimal when adjusting these.

5.2 Evaluation metrics

For each class i, we compute the following values, based on the true class y and the predicted class $\hat{y}$ for each instance.

- The true positives (TP_i) are the number of instances where $\hat{y} = y = i$.
- The false positives (FP_i) are the number of instances where $\hat{y} = i$ but $y \neq i$.
- The true negatives (TN_i) are the number of instances where $\hat{y} \neq i$ and $y \neq i$.
- The false negatives (FN_i) are the number of instances where $\hat{y} \neq i$ but $y = i$.

The precision for a given class i is defined as:

$$Precision_i = \frac{TP_i}{TP_i + FP_i}$$

The recall for a given class i is defined as:

$$Recall_i = \frac{TP_i}{TP_i + FN_i}$$

The F_1 measure for a given class i is the harmonic mean of the precision and the recall:

$$F_{1_i} = 2\frac{Precision_i Recall_i}{Precision_i + Recall_i}$$

Let N_i be the number of instances which belong to class i (i.e., the true label is i). Let $N = \sum_i N_i$ be the total number of instances. The overall F_1 measure is given as the weighted average of the F_1 measure for each class:

$$F_1 = \frac{1}{N}\sum_i N_i F_{1_i}$$

The overall *accuracy* of the model is defined as the ratio between correctly classified instances and the total number of instances:

$$Accuracy = \frac{\sum_i TP_i}{N}$$

In addition, we define the *confidence* for a prediction as the value of the probability for the predicted class $\hat{y}$, as given by the output of the network.

For many tasks, we prefer to maximize the precision at the cost of discarding some predictions the model is "unsure" about. This is commonly known as the precision-recall trade-off.

Therefore, we compute the precision, recall and F_1 measure as a function of the prediction confidence for the POS model, in order to test whether the more confident predictions are more precise and to measure the impact of choosing only the more confident predictions, in terms of how many instances we would get a prediction for.

For the lemma model, the confidence is generally very low due to the large number of classes; we would need to perform some kind of normalization to properly evaluate this trade-off.

Language	Target	% Ambiguity	Precision	Recall	F_1 score
Finnish	POS	8.9	0.81	0.79	0.79
	Lemma	8.8	0.80	0.75	0.76
Russian	POS	7.7	0.85	0.84	0.84
	Lemma	10.2	0.82	0.77	0.80
Spanish	POS	15.3	0.83	0.81	0.81
	Lemma	15.5	0.82	0.75	0.75

Table 5: Evaluation metrics for each model.

6 Results

6.1 Evaluation

We trained models for POS and lemma disambiguation for all three languages, and computed our metrics over the evaluation set, composed solely of ambiguous instances.

Table 5 shows the results for the evaluation metrics for each model. The column **ambiguity** indicates the percentage of ambiguous tokens in the *training* corpus, for reference.

The results show that our approach works to a reasonable extent. There is still room for improvement, but clearly the models are able to learn to predict the correct lemma or POS.

One potential drawback of our method could be that it needs a large enough number of unambiguous instances to be able to train. However, in our results, we see that even though Spanish has a significantly higher percentage of ambiguous tokens than the other languages, the models perform comparably—which leads us to believe that other factors, such as the level of overlap in surface forms for each lemma[21]—are much more relevant to the quality of the trained model.

In addition, we perform error analysis for the POS model—since there are too many labels for the lemma model—so as to verify whether the model is able to predict every class. The overall accuracy could otherwise be misleading, since the model could simply be predicting the most likely label for every case, and failing

[21] That is, on average, how many surface forms overlap for a given pair of lemmas out of all the possible surface forms.

Language	Noun	Adjective	Verb	Adverb	Other
Finnish	75.8	82.8	78.3	74.7	75.5
Russian	77.4	79.8	90.7	82.6	92.5
Spanish	82.6	71.6	86.3	86.6	82.7

Table 6: Accuracy (percent) for each POS.

at predicting the less common POS from the possible label set.

There are significant differences across POS, but given that the average ambiguity "size" (i.e., length of the set of possible labels) across languages is approximately 2.5—thus a random prediction would be correct around 40% of the time—the model performs well above the random baseline for each class.

In Section 5.2 we hypothesized that we could trade off recall in order to obtain near-perfect precision for applications in which obtaining correct predictions is much more important than obtaining a prediction at all. To this end, we conducted the following experiment.

First, we define an "unsure" label which always results in a false negative for every class—i.e., an indication that the model does not return a prediction for the instance. Note that such a false negative will reduce recall, but not precision, since the latter only depends on true positives and false positives.

Next, we set a confidence threshold θ_{conf} such that any prediction with confidence below that threshold will be set to "unsure."

We ran this experiment only on the POS model, due to the reasons stated in Section 5.2—the fact that the confidence is inversely proportional to the number of classes—which were confirmed experimentally.

Figure 6 shows the results for Finnish. Figure 7 shows the results for Russian. Figure 8 shows the results for Spanish.

The results confirm that we can indeed obtain very high-precision (> 0.9) predictions by choosing only those with high confidence, while maintaining a high enough recall (≈ 0.5) that the model is still useful. This is of particular importance in applications where there is human-machine interaction—such as our language-learning application—where for the machine, being unsure is acceptable, but being wrong is not, since it actively misleads the human.

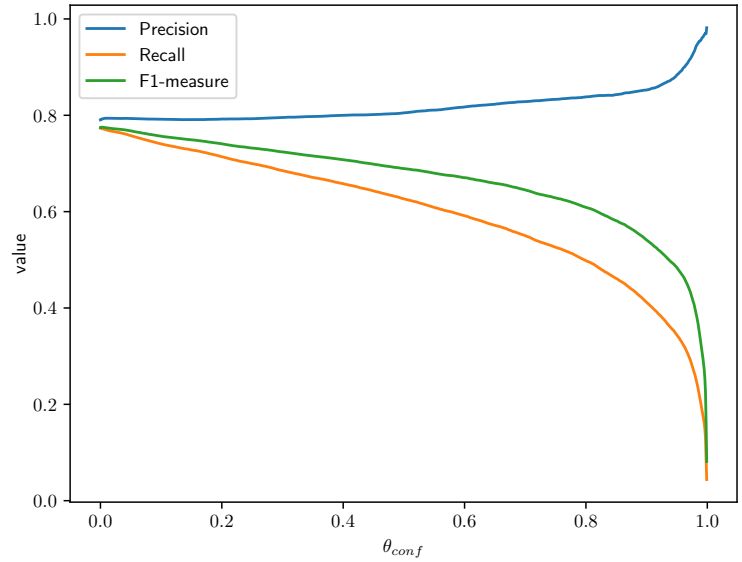

Figure 6: Finnish POS confidence vs. metrics.

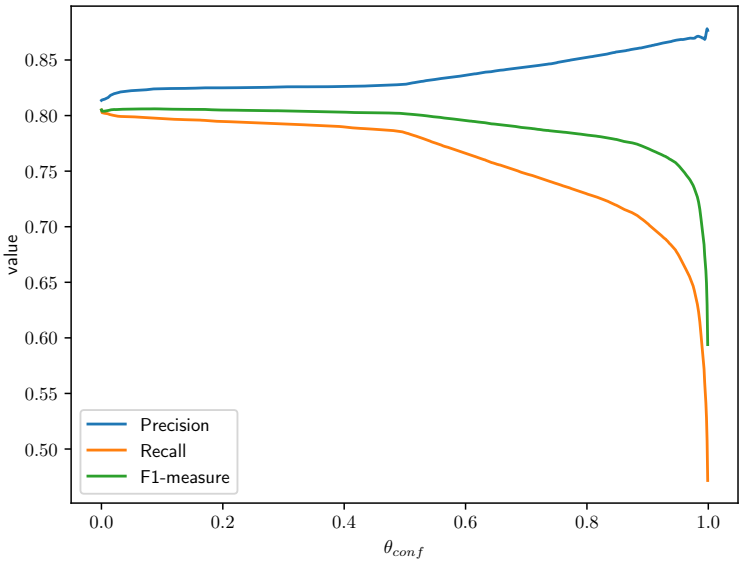

Figure 7: Russian POS confidence vs. metrics.

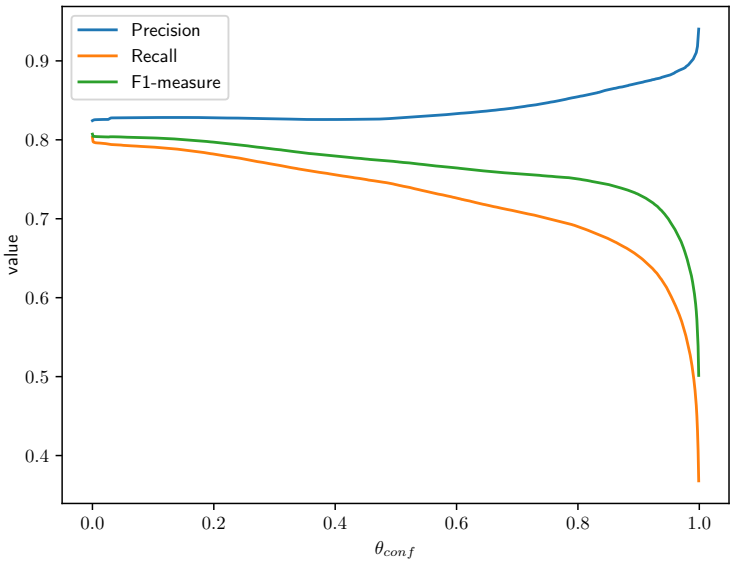

Figure 8: Spanish POS confidence vs. metrics.

6.2 Comparison with state of the art

In order to compare our results with the state of the art, we must use token-level accuracy instead of accuracy on ambiguous instances only, since state-of-the-art methods provide metrics only across the entire set of tokens.

Let A be the percentage of ambiguity in the corpus and G the guided disambiguation accuracy. The token-level accuracy T is thus $T = 100 - A + AG/100$. That is, T is the weighted average of the accuracy over unambiguous instances (which is always 100%) and ambiguous instances, expressed as a percentage.

For Russian, the best result to date for POS tagging was reported by Dereza et al. [4], achieved using TreeTagger [27], at 96.94%. We could not find lemmatization results for Russian, but the work by Korobov [17] solves the broader problem of morphological ambiguity with an accuracy of 81.7%.

For Finnish POS tagging and lemmatization, the TurkuNLP neural model [12] achieves 97.7% and 95.3% accuracy, respectively, evaluated on the same dataset as our method.

For Spanish POS tagging and lemmatization, the model by Carreras et al. [3] achieves an accuracy of 89% and 88%, respectively, according to the evaluation done by Parra Escartín and Martínez Alonso [23].

Language	Target	Blind	Guided	Token	SOTA
Finnish	POS	73.7	79.1	**98.1**	97.7 [12]
	Lemma	40.4	76.5	**97.9**	95.3 [12]
Russian	POS	81.9	83.9	**98.6**	96.9 [4]
	Lemma	60.7	83.4	**98.3**	81.7 [17][a]
Spanish	POS	74.9	80.7	**97.0**	89.0 [23]
	Lemma	55.2	74.7	**96.1**	88.0 [23]

Table 7: Comparison with state-of-the-art POS tagging and lemmatization.

[a]This is a lower bound, since they solve the more complex problem of full morphological disambiguation (i.e., including feature disambiguation).

Table 7 shows our results for accuracy alongside the results for the current state of the art.

The columns mean:

- **Blind**: The accuracy for the blind method, over only *ambiguous* tokens.
- **Guided**: The accuracy for the guided method, over only *ambiguous* tokens.
- **Token**: Overall token-level accuracy, by applying the guided method.
- **SOTA**: The state-of-the-art results.

With these results, we feel that our approach is currently a viable alternative to traditional methods that employ manually annotated data for training. Our method is not targeted to languages where the amount of annotated data is sufficient—such as English—but rather to those in which it is not, like the three languages tested here—and any other language, provided a morphological analyser exists.

7 Conclusions and future work

7.1 Conclusions

Overall, we believe that the proposed models have met the original goals for this work. We have shown that it is possible to train a model for morphological disambiguation using only unambiguous instances and verified that the approach works for several languages with no modifications. We have surpassed the performance of state-of-the-art methods—which rely on manually annotated data—for the set of evaluated languages, perhaps in doing so proving that the amount of annotated data for these languages is indeed insufficient.

However, the results show that there is still much room for improvement. In the following section, we go over some of the ideas we have for future lines of research in this problem.

7.2 Future work

7.2.1 Model enhancements

Feature disambiguation In Table 2 we saw that, although POS disambiguation works for most of the cases, around 9% of the ambiguities are only solvable by lemma disambiguation. Moreover, around 6% of the instances currently cannot be disambiguated using either method.

This puts the upper limit on accuracy to 85% for POS model, which is the better of the two. If we were to use the lemma model for the type of ambiguity that is only solvable by lemma disambiguation—two declinable words with the same POS and different lemma, as shown in Section 2.3—the upper limit on accuracy would be raised to 94%.

To push this limit to 100%, it would be necessary to use a model that disambiguates morphological features. A relatively straightforward approach to this would be to make a vocabulary of feature combinations. For example, for the Finnish word *nostaa* ("to raise" / "he raises"), with the following readings:

```
nostaa+V+Act+Ind+Prs+Sg3
nostaa+V+InfA
```

We could consider the feature sets `Ind+Prs+Sg3` and `InfA+Sg+Lat` to be two dis-

tinct labels in our vocabulary (similarly to two different lemmas or POS) and use the same method as for the POS and lemma models.

This method could work as long as the number of feature combinations in the language is finite, as is the case for the three languages we tested, and as long as the number of combinations is reasonable—the lemma model has been shown to work, while having a vocabulary size of ≈ 100000. It is unclear whether this approach would work with a larger vocabulary, although it is likely that we would need a proportionally larger training corpus.

We did not try to implement this mostly due to time constraints; the feature set used in our annotated data differs significantly from that produced by our analyser, and so it would have been necessary to produce a mapping from one to the other in order to evaluate the model.

Ensemble solution Since we can determine the type of ambiguity (according to our taxonomy) during inference, an ensemble solution that picks the best model depending on the type of ambiguity would be a natural next step.

A baseline ensemble model could simply select the model based on whether the ambiguity is solvable by said model or not. A more sophisticated model could pick the model which obtained the highest tested accuracy for each type of ambiguity.

The results show that confidence is a strong predictor for accuracy. Therefore, an improved version of the ensemble model could instead decide on a per-instance basis, taking the prediction with the highest confidence. We would have to normalize the confidence for POS, lemma and feature in order to make them comparable.

Named entity recognition As mentioned in Section 4.1.3, our method currently conflates proper noun readings with common noun readings, effectively ignoring the possible ambiguities between the two. For example, the Finnish word *Turku* can refer either to the city in Finland, or it can be a capitalized form of *turku* ("marketplace"), which is an archaic doublet of *tori* ("market" / "town square").

This type of ambiguity is quite prevalent in all of the languages we tested, since names often originate from common nouns or adjectives. When the word appears at the beginning of the sentence, the ambiguity becomes non-trivial: is it capitalized because it is a name, or because it is the first word in the sentence? In German in particular, this issue is even more widespread and harder to solve, since

all nouns are capitalized, regardless of where they appear in the sentence.

Therefore, a worthwhile addition to our method would be to perform named entity recognition before giving the data to the disambiguation model, keeping only proper noun readings for those words which are deemed names, and removing them for all others.

7.2.2 Other approaches

Alternative context features Currently, our method represents context words using only an embedding function that is trained separately from the rest of the model. It would perhaps be worth exploring replacing the embedding function with some network architecture that computes these embeddings and is trained alongside the rest of the model, in order to maximize the relevance of the learned features to the problem of morphological disambiguation.

Character-level convolutional neural networks are a popular choice for this in other state-of-the-art representation learning models, such as ELMo [24]. They do not come without downsides, however: the training is much slower, it consumes much more memory, and the context window must be a fixed length of characters—whereas in our method there is nothing preventing us from having variable-length windows.

In addition, it could be useful to enrich the context representation with features extracted from the morphological analyser. For example, the LSTM could receive the POS of each word in the context. This would probably help guide the training significantly, but there is still the issue of ambiguity in the context: if a context word is ambiguous, which POS do we give? Do we choose one randomly, give both, or give none? We would need to devise a feature representation that accounts for this issue.

Multi-task learning Another approach we have explored is the use of multi-task learning to predict both POS and lemma jointly. So far this has been somewhat unsuccessful, yielding an accuracy slightly below that of either of the single-task models, but we believe there is still much room for improvement.

The intuition behind multi-task learning is that both models should look for similar contextual cues and the information learned by one model should be useful for the other, since the two tasks are dependent to some extent.

We tried a naïve approach, reusing the LSTM parameters between the two models and alternating training steps between the two objectives. A better approach may be to combine the loss of the two tasks into a single loss value, and then try to optimize that. The main technical challenge in this approach would be to normalize the loss of the two tasks so that they are both considered equally important during training.

Language models A language model computes the likelihood of a given list of words in a language. This problem subsumes that of morphological disambiguation, given that a grammatically correct (i.e., morphologically, syntactically and semantically consistent) sentence should be far more likely than an incorrect one.

One way to leverage this information for disambiguation could be as follows. First, take a sentence with an ambiguous word. Next, for each possible reading, replace the ambiguous word with an unambiguous word that fulfils the same role in the sentence (i.e., it has the same POS), has the same set of morphological features, and is semantically close to the lemma given in the reading. Finally, compute the likelihood of each resulting sentence. The most likely sentence should be the one containing the substitute for the correct reading.

For example, consider our previous example *Hän tuli kotiin.* ("He came home."). The word *tuli* ("fire" / "he came") is ambiguous, with a nominal and verbal reading:

```
tuli+N+Sg+Nom
tulla+V+Act+Ind+Prt+Sg3
```

First, we pick a noun substitute for *tuli* ("fire"), such as *palo* ("fire, conflagration"), inflect it in the same number and case, and replace it in the sentence, thus giving us **Hän palo kotiin.* ("*He fire home."). The likelihood for this sentence should be very low, since it is grammatically incorrect, and one would be hard-pressed to find an instance of such sentence being used in the language.

Next, we pick a verb substitute for *tulla* ("to come"), such as *mennä* ("to go"), inflect it in the same tense and person, and replace it in the sentence, resulting in *Hän meni kotiin.* ("He came home."). The likelihood for this sentence should be much higher, since it is grammatically correct, and has probably been used many times in the language, in an equal or similar form.

Therefore, picking the lemma and/or POS associated with the more likely substituted sentence (in this case, *tulla* and "verb") effectively solves the ambiguity.

As for automatically picking adequate substitutes, one could compute the lemma embeddings for the language and choose, among the nearest neighbours in the embedding space for each lemma, one with the same POS.

We believe this approach has potential, especially now that language models are becoming more accurate, but there is still the issue of morphological disambiguation being a much simpler problem than language modelling, and thus in theory requiring far fewer resources and model complexity in order to solve it.

For comparison, state-of-the-art language models such as GPT-2 [26] take weeks to train on a GPU cluster, whereas our approach takes less than an hour on a desktop computer with a single GPU.

References

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

2 P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017. [Online]. Available: https://www.aclweb.org/anthology/Q17-1010

3 X. Carreras, I. Chao, L. Padró, and M. Padró, "Freeling: An open-source suite of language analyzers," in *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04)*, 2004.

4 O. Dereza, D. Kayutenko, and A. Fenogenova, "Automatic morphological analysis for Russian: A comparative study," in *Proceedings of the International Conference Dialogue*, 2016.

5 M. L. Forcada, M. Ginestí-Rosell, J. Nordfalk, J. O'Regan, S. Ortiz-Rojas, J. A. Pérez-Ortiz, F. Sánchez-Martínez, G. Ramírez-Sánchez, and F. M. Tyers, "Apertium: a free/open-source platform for rule-based machine translation," *Machine Translation*, vol. 25, no. 2, pp. 127–144, 2011.

6 G. D. Forney, "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, March 1973.

7 F. A. Gers, J. A. Schmidhuber, and F. A. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, Oct. 2000. [Online]. Available: http://dx.doi.org/10.1162/089976600300015015

8 E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov, "Learning word vectors for 157 languages," in *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

9 D. Z. Hakkani-Tür, K. Oflazer, and G. Tür, "Statistical morphological disambiguation for agglutinative languages," *Computers and the Humanities*, vol. 36, no. 4, pp. 381–410, 2002.

10 K. Hornik, “Approximation capabilities of multilayer feedforward networks,” *Neural Networks*, vol. 4, no. 2, pp. 251 – 257, 1991. [Online]. Available: http://www.sciencedirect.com/science/article/pii/089360809190009T

11 G. Inoue, H. Shindo, and Y. Matsumoto, “Joint prediction of morphosyntactic categories for fine-grained Arabic part-of-speech tagging exploiting tag dictionary information,” in *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, 2017, pp. 421–431.

12 J. Kanerva, F. Ginter, N. Miekka, A. Leino, and T. Salakoski, “Turku neural parser pipeline: An end-to-end system for the CoNLL 2018 shared task,” in *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Association for Computational Linguistics, 2018.

13 F. Karlsson, “Constraint grammar as a framework for parsing running text,” in *COLING 1990 Volume 3: Papers presented to the 13th International Conference on Computational Linguistics*, 1990.

14 A. Katinskaia, J. Nouri, and R. Yangarber, “Revita: a system for language learning and supporting endangered languages,” in *Proceedings of the joint workshop on NLP for Computer Assisted Language Learning and NLP for Language Acquisition*. Gothenburg, Sweden: LiU Electronic Press, May 2017, pp. 27–35. [Online]. Available: https://www.aclweb.org/anthology/W17-0304

15 D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *CoRR*, vol. abs/1412.6980, 2015.

16 E. Klyshinsky, N. Kochetkova, M. Litvinov, and V. Y. Maximov, “Method of POS-disambiguation using information about words co-occurrence (for Russian),” *Proceedings of GSCL*, pp. 191–195, 2011.

17 M. Korobov, “Morphological analyzer and generator for Russian and Ukrainian languages,” in *Analysis of Images, Social Networks and Texts*, M. Y. Khachay, N. Konstantinova, A. Panchenko, D. Ignatov, and V. G. Labunets, Eds. Cham: Springer International Publishing, 2015, pp. 320–332.

18 A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in *Proceedings of ICML*, vol. 30, 2013, p. 3.

19 T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013.

20 S. N. Moshagen, T. Pirinen, and T. Trosterud, "Building an open-source development infrastructure for language technology projects," in *Proceedings of NODALIDA 2013: the 19th Nordic Conference of Computational Linguistics*, Oslo, Norway, 2013, pp. 343–352.

21 J. Nivre, J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel, and D. 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)*, 2007.

22 J. Nivre, M. Abrams, and al., "Universal Dependencies 2.3," 2018, LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. [Online]. Available: http://hdl.handle.net/11234/1-2895

23 C. Parra Escartín and H. Martínez Alonso, "Choosing a Spanish part-of-speech tagger for a lexically sensitive task," 2015.

24 M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proceedings of NAACL*, 2018.

25 S. T. Piantadosi, H. Tily, and E. Gibson, "Word lengths are optimized for efficient communication," *Proceedings of the National Academy of Sciences*, vol. 108, no. 9, pp. 3526–3529, 2011. [Online]. Available: https://www.pnas.org/content/108/9/3526

26 A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, 2019.

27 H. Schmid, "Probabilistic part-of-speech tagging using decision trees," 1994.

28 N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

29 A. Tkachenko and K. Sirts, "Neural morphological tagging for Estonian," in *Human Language Technologies–The Baltic Perspective: Proceedings of the Eighth International Conference Baltic HLT 2018*, vol. 307. IOS Press, 2018, p. 166.

30 M. A. Yatbaz and D. Yuret, "Unsupervised morphological disambiguation using statistical language models," in *Pro. of the NIPS 2009 Workshop on Grammar Induction, Representation of Language and Language Learning, Whistler, Canada*, 2009, pp. 321–324.

31 N. Zalmout and N. 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*, 2017, pp. 704–713.

32 P. Šmerk, "Unsupervised learning of rules for morphological disambiguation," in *International Conference on Text, Speech and Dialogue*. Springer, 2004, pp. 211–216.