# Word Embeddings and Morphological Tools for Indian Languages


Kumar Saurav
160050057

Computer Science and Engineering

Indian Institute of Technology Bombay

*B. Tech. Thesis Project*

# Contents


**1 Introduction** **1**
- 1.1 Intricacies of Language . . . 1
- 1.2 Linguistics . . . 3
- 1.3 Natural Language Processing . . . 3
  - 1.3.1 Basic NLP Tasks . . . 4
  - 1.3.2 Pipeline . . . 6
  - 1.3.3 Ambiguities . . . 6
- 1.4 Word Embeddings . . . 10
  - 1.4.1 Characteristics . . . 10
  - 1.4.2 Representation Methods . . . 12
  - 1.4.3 Brief overview of Developments . . . 12

**2 Word Embedding Techniques** **14**
- 2.1 Preliminary . . . 14
  - 2.1.1 Neural Probabilistic Language Model . . . 14
- 2.2 Word2vec . . . 16
  - 2.2.1 CBOW . . . 16
  - 2.2.2 Skip-gram . . . 17
  - 2.2.3 Performance . . . 18
  - 2.2.4 Improvements . . . 19
- 2.3 ELMo . . . 21
  - 2.3.1 LSTM . . . 21
  - 2.3.2 Model . . . 23
  - 2.3.3 Performance . . . 24
- 2.4 BERT . . . 25
  - 2.4.1 Transformers . . . 26
  - 2.4.2 Model . . . 28
  - 2.4.3 Performance . . . 30

**3 Applications of Word Embeddings** **32**
- 3.1 Feedforward Networks . . . 32
- 3.2 Convolutional Neural Networks . . . 34

**4 Word Embeddings for Indian Languages** **35**
4.1 Prior resources . . . 35
4.2 Evaluating embeddings . . . 35
4.3 Available data . . . 36

**5 NER for Indian Languages using Word Embeddings** **38**
5.1 Previous work . . . 38
5.2 Our work . . . 38
5.2.1 GloVe . . . 38
5.2.2 Word2vec and Fasttext . . . 39
5.2.3 ELMo . . . 39
5.2.4 BERT . . . 39
5.3 Results . . . 40

**6 Morphological tasks for Indian languages** **41**
6.1 Utility . . . 41
6.2 Background and our work . . . 41
6.3 Data . . . 42

**7 Morphology using embeddings** **43**
7.1 Fasttext . . . 43
7.2 Evaluation on multiple languages . . . 44
7.3 Analysis of results on Hindi . . . 47
7.4 Improving gold standard . . . 47
7.5 ElMo . . . 49
7.6 Reverse task . . . 49
7.7 Using geometric methods . . . 50
7.8 Aligning embedding spaces of different languages . . . 51
7.9 Drawbacks . . . 53

**8 Morphology using seq2seq models** **55**
8.1 Data preparation . . . 55
8.1.1 Data augmentation . . . 56
8.2 Vanilla seq2seq model . . . 56
8.3 Attention based models . . . 57
8.4 Cross lingual training . . . 58
8.4.1 Results . . . 59
8.5 Other Methods . . . 60
8.5.1 Results . . . 60

**9 Future Work and Conclusion** **62**
9.1 Future Work . . . 62
9.2 Conclusion . . . 63

# Chapter 1

# Introduction

Language is an important part of human society. It allows us to not only exchange ideas, express feelings but also to store and distribute information across time and distance. There are a multitude of languages today, in fact at least 7000[14], and each one of them has its own idiosyncrasies which make it unique. There are several theories as to how the system of language developed, including the gestural theory [4], Putting the baby down theory[27] and From where to what theory[50] but there is no conclusive evidence for any of them. Nonetheless, language has developed into a complex and intricate form today, the quotidian use of which makes such facts oblivious to most of us.

## 1.1 Intricacies of Language

Consider the case of Hindi for example. The hindi alphabet consists of 56 letters, and yet has more than billions of words in its vocabulary. New words are added to a language constantly: vocabulary for conversation and official purposes in Hindi has gone up by over 7.5 times during 1990s to 2019 - from around 20,000 to 1.5 lakh. These counts exclude the repertoire of scientific and technical terms and their derivatives. Despite having such a huge number of words in the vocabulary, randomly picking any sequence of letters is more likely than not to be an actual word. For instance नरौऊज़ is not even allowed as per the grammatical rules in hindi - only a single मात्रा (maatra - similar to vowels in English) is allowed to associate with a single व्यंजन (vyanjan - similar to consonants in English). As another example, while ytqasd is not forbidden as per the grammatical rules in English, it is not a word which exists in the English vocabulary.

Such complications exist at the level of words in a language. The power of a language though is derived from the use of sentences - a collection of words in a "meaningful" way, where "meaningful"-ness is prescribed by the language's grammar. Consider the simplest possible structure of a sentence - Subject - Verb - Object, where subject refers to the entity doing the action, verb is the action itself and object is entity on which the action is being performed. This is the general sentence layout for English. There are multiple ways in which can we can develop a single sentence out

of this structure:

1. Subject - We can choose a proper noun, which could refer to a person, or a city, or an animal and so on. Or we could also choose a concrete noun like milk, table, pen etc. or a common noun like university, state etc.

2. Verb - Any action like eating, throwing, playing works in this place. Then we need to decide the "form" of the verb - should it be simple present, past perfect, present continuous or some other form?

3. Object - A noun again fits the requirements in this place. Which article should accompany the noun? Should the noun be plural or singular? Or should it instead be a collection of nouns like "a bat and a ball"?

Of course, these are not the only possibilities in which a sentence could have been built even if we were restrained to the SVO structure. But note that even in this case, there are sentences which fail to make any sense - like "Happiness blows cereal". And along with this, there are instances in which the same word behaves differently according to the context (context refers to the meaning suggested by the other words). Consider the sentences "He watches TV" and "She watches her diet". The former refers to the "seeing" sense of the word "watch" while the latter refers to the "monitoring" sense. It's also possible that the word may assume a different role altogether - "Watches are tools". Of course, not all such interchanges are possible and that is precisely due to the semantics.

Moreover, there are multiple representations possible for the same sentence:

- I eat lunch
- Lunch is eaten by me
- I consume lunch

It's always possible to represent the same sentences, or rather the meaning of a sentence, using a different sentence. The simplest way, as shown, is to change the sentence structure or to use synonyms (words that have the same meaning).

In Hindi, we generally use a different sentence structure: Subject - Object - Verb. Along with this difference in structure, the difference in the vocabulary makes the set of complications that arise in Hindi unique to that language. At the very basic level, verbs follow different modification rules than in English and the vocabulary mapping from Hindi to English is many-to-many. Even this is a huge complexity and makes it difficult to learn Hindi based on an acquired knowledge of English.

Note that the semantics of the language now come into the picture - along with the syntax, crudely the word-level correctness and correct word forms, we are now also interested in the meanings and associations of each word in a coherent sequence - which is what semantics refers to. Semantics is precisely why the sentence "Happiness blows cereal" is not a meaningful sentence - the sense of the word "happiness" does not relate to a an object which can "blow" and neither is "cereal" is an object likely to be "blown". For instance, the different meanings of the word "watch" manifest

are included in the semantics of the languages. We also use semantics to refer to the grammatical aspects of language which are not covered by syntax, like reference - for which word in the sentence or dialog was the word "she" used as a reference?

We already face very complex problems when we try to extract meanings from or generate sentences. On top of this, we need to make sense of a sequence of sentences which result in texts and dialogues, or in technical terms, a discourse, to extract meanings and update our knowledge of the outside world. But there is a subtlety involved here - we already incorporate knowledge of the outside world in semantics. Otherwise, how would we have known that watches are things to be worn, or cereals are meant to be eaten and not blown?

## 1.2 Linguistics

The formal study of the science of language is known as linguistics. In linguistics, a language is considered to be a set of signs. A sign, in turn, is a pair consisting of a "signifier" and the "signified" - for instance the string `human` is a signifier and its signified is the human species. In this setting, sign systems are ubiquitous - posters, clocks, road signs are all sign systems. Languages differ from these only in their complexity. It is easy to see that the former group of sign systems - clocks, diagrams etc. - can produce only a finite set of legible signs. In contrast, languages allow us to express each thought that we have and the number of signs can be endless.

Formally in linguistics, language signs are composed of four parts:

1. Phonology - It deals with the phonetic part of the language, i.e., how words are pronounced
2. Morphology - The minimal part of speech that bears meaning is a morpheme. Words may or may not be morphemes. For example, `waiting` is a combination of two morphemes `wait` and `ing`
3. Syntax - It specifies how words combine together to form a sentence
4. Semantics - this is the only branch which deals with the "signified" part of the sign, i.e., the meaning

The power of language to generate so many signs is due to the fact that it has rules to form complex signs from simple ones. The same is not true for either road signs or clocks.

Natural languages are simply languages which have developed in humans naturally through years without conscious planning or premeditation. Thus the the set of languages that humans speak belongs to set of natural languages too.

## 1.3 Natural Language Processing

Natural language processing (NLP) is a field of artificial intelligence that aims to give machines the ability to read, interpret and derive meaning from natural languages.

The processes involved can encompass multiple domains depending on the problem we are tackling. For example, the work involved in understanding a language in an end-to-end NLP task will have the following:

1. Acquiring speech - ingesting the language into the computer system. The communication can either be in audio or written format. In case of written format, it could either be a hand-written document or an image (basically in a time-static document which is not immediately readable by the machine) or it could be a document on the machine itself, which is convenient for consumption. When the input format is audio, the sound signals need to be recorded

2. Identifying word boundaries - This is difficult in case of audio signals - spaces between words will ideally be silence, but noise complicates the boundary detection. For documents on the machine, words are simply parts of texts enclosed within blanks.

3. Recognising words - Converting each audio strip or image block (based on the boundary detected) to the corresponding word

4. Understanding meaning - Apply knowledge of semantics to the generated words to get the meaning

Once we have "understood" the language, we can also ask how do we create sentences based on our understanding - sentences that are syntactically and semantically correct?

Many problems are immediately apparent in this methodology: more often than not steps 2 to 4 are not sequential - they are interlinked and information across these steps needs to be consistent to generate a valid meaning. Audio to text is the concern of a further sub-field known as Automatic Speech Recognition, which involves knowledge in the acoustics domain. Obtaining knowledge of semantics and applying it are two difficult tasks in themselves, partly because the vocabulary involved is huge and partly because the computational resources are limited. Suppose we could encode all knowledge of semantics into a machine but using this information requires expensive resources like time. Such a system is not what we aim for. Instead, we aim to build systems that reach aim to maximise metrics given the computational resources we can obtain. We can still ask which metrics though.

### 1.3.1 Basic NLP Tasks

It is readily clear that NLP lies at the intersection of information engineering, artificial intelligence, computer science and linguistics. Considering an end-to-end application, NLP is concerned with questions involving three dimensions[9]:

- Language - Which language are we working on? Which linguistic model are we using?

- Problem - Which problem are we tackling? Are we parsing or we tagging parts-of-speech or some other problem? What is the metric that we want to maximise?
- Algorithm - What algorithm do we choose to solve the problem for the given language? Is it Hidden Markov Models, or Conditional Random Fields?

Natural language analysis requires proficiency in foundational tasks such as Part of Speech Tagging, Named Entity Recognition, Coreference disambiguation, Semantics Extraction, Discourse Processing. Consider an example to illustrate these tasks

| | | |
|---|---|---|
| Mother | : | *Did your brother accompany you to the bank?* |
| Son | : | *Yeah, Gaurav did. He didn't go ballistic like last time.* |
| Mother | : | *Your mother would have been angry if he did.* |

Table 1.1: Example: Conversation in a family

Processing the dialog in 1.1 involves the following:

1. The first is a question, while the second is not
2. The person whom the mother is talking to has a brother. This requires an inferencing process
3. *Bank* has multiple meanings - a river *bank* and a financial *bank*. The latter is relevant here
4. *Gaurav* is the brother of the person the Mother is talking to
5. *Gaurav* is a proper noun. Named Entity Recognition (NER) - detection of proper nouns - in English is easier since the proper nouns have capitalised first letters
6. The second sentence is a shortened version of an affirmative reply
7. *Go ballistic* is non-compositional in the sense that individual meanings of *go* and *ballistic* cannot be composed to get the meaning.
8. *He* refers to Gaurav in the son's sentence
9. Gaurav was angry the last time he was asked to go to a bank. This is again an inference using the semantics of the second language
10. *Mother* in the third sentence is ambiguous. Does it mean that Gaurav and the other person have two mothers or is the mother referring to herself?
11. Understanding why the mother would have been angry is a complex inference process (they want the sons be with each other). The third sentence is actually referring to a hypothetical situation and the reaction of a person in that hypothetical situation.
12. *He* in the last sentence again refers to Gaurav and not the other son

### 1.3.2 Pipeline

Traditionally, NLP - for both spoken and written languages - are processed through the following pipeline[9]:

1. Phonology and Phonetics - utterances are processed at this stage. The input is an audio waveform and output is a word or a sequence of words

2. Morphology - Words are usually derived from root forms through processes of inflexion, derivation, back formation, clitics and portmanteauing[1]. Word forms are processed at this stage

3. Lexicon - We associate each word with knowledge that will be useful further down the pipeline. Examples include the knowledge of the PoS (part-of-speech) tag, morphological information, semantic tag.

4. Parsing - We identify the structure of the sentence and the hierarchical relation between words. The structure of the sentence helps us to single out the qualifiers attached to a word, and separate ideas in a sentence.

5. Semantics - We extract the meaning of the sentence at this stage. What this means is that the ideas that are communicated through the sentence needs to processed into an unambiguous form. Multiple forms like semantics net, predicate calculus etc. exist for this purpose[54].

6. Pragmatics - This involves processing user intentions, belief worlds, sentiments, modals etc.

7. Discourse - Using connected sentences, we continuously produce and update hypotheses that are likely and plausible in the current setting.

### 1.3.3 Ambiguities

In each stage in the pipeline, there are multiple ambiguities which arise due to the intricacies in language

#### Phonetics

Apart from the problems that arise due to imperfections in audio collection mechanisms and the noise in the environment, we face problems like `word boundary detection` and `homophone disambiguation`

- Word boundary detection - It is a problem particularly prevalent in rapid speech. Consider the following examples:

  `the same` or `thus aim`

  The same sounding string can be broken in two different ways, giving rise to two completely different meanings.

- Homophone - There are multiple set of words which sound the same, but have wildly different meanings. Examples include "break" (to separate into pieces) and "brake" (device for stopping a vehicle), "cell" (a jail) and "sell" (give in exchange for money). Homophones are usually a result of borrowing words from foreign languages. For example, "cell" is derived from the Latin "cella" which was a storeroom or a chamber, while "sell" is related to the Old Norse "selja" which means to sell.

- Near homophone - Words sound similar, but not same form the set of near homophones. It is a more common occurrence than homophones: "lacks" and "lax". In hindi, the same concept is referred to as श्रुतिसम भिन्नार्थक (shruti-sammbhinaarthak). Examples include अचार (achar - a pickle) and आचार (aachar - behavior), अवधि (awadhi - time) and अवधी (awadhee - a language),

### Morphology

Ambiguities arise when there are multiple options to break a word into stem, suffix as well as from choices of features[64]. For instance, the word *unlockable* can be broken in two ways:

- *unlock* + *able* - meaning that the thing cannot be unlocked

- *un* + *lockable* - meaning that the thing cannot be locked

A similar example would be *undoable*. This problem is more widespread for morphologically rich languages like Hindi and Sanskrit.

There are also words which were introduced recently to the vocabulary and are formed using techniques like portmanteau, which pose a challenge for morphological processing. Some examples of such words would be

- *edutainment* = *education* + *entertainment*: video games, television programmes, or other material, intended to be both educational and enjoyable

- *telethon* = *television* + *marathon*: a very long television program

### Lexicon

A typical processing in the lexicon pipeline would involve imbuing the word "fox" with the following information:

- *POS*: Noun

- *Semantic Tag*: Wild animal, 4-legged

- *Morphological structure*: takes 'es' in plural

The ambiguity is immediately visible - the same word can take on multiple meanings. For example, a *fox* can either refer to a wild animal, or a cunning person.

Word sense disambiguation refers to the identification of correct word sense based on the context clues available. For example, in *The shopkeeper played the customer*, using the setting, worldly knowledge and the appearance of the word *play* as a verb, we can correctly guess that the *play* takes on the the word sense verb, meaning "to trick".

### Parsing

Identifying the hierarchical structure behind the sentence involves attaching qualifiers to words. For example, in *I have a red car*, the adjective *red* (adjective - due to lexical parsing) is attached to the noun *car* instead of the pronoun *I*. As sentences become more and more complex, it can become harder to identify how and where qualifiers attach - a problem of *structural ambiguity*. We can distinguish between two types of ambiguities within this subclass:

- Scope ambiguity - *Prisons are where unlawful men and women sent.*

  By scope we mean the amount of text that a word qualifies. The scope of the word *unlawful* is not clear. Are unlawful men and all women sent to prison or only unlawful men and unlawful women? Another example would be *Indian history teacher*. Is the history teach an Indian, or does the teacher teach Indian history?

- Attachment ambiguity - *She was a person who never met a child and didn't play.*

  The intended meaning of the sentence is a person who "never met a child without playing" or "who played whenever she met a child", but can be interpreted as a person who never met a child and never played. The ambiguity arises from the confusion over which object "didn't play" should attach to.

### Semantics Processing

Assigning semantic roles to words or semantic relations between words causes ambiguities due to the multiple word senses possible. Consider the following examples:

- *We saw her duck*: It depends on which semantic role we assign to duck - is it a verb or a noun? Does the sentence paraphrase as "we saw her lower her head" or "we saw the duck which belongs to her"[32]

- *I can't tell you how much I enjoyed meeting your husband*: Does the speaker mean to say that she/he highly enjoyed the meeting or she/he is not allowed to say how the meeting went? [26]

### Pragmatics Processing

Pragmatics studies the way in which context contributes to meaning. The same sentence can take on different meanings when put in different contexts. To illustrate this idea, consider the example

- *The police is coming* - Is the speaker a criminal and the police are coming to arrest him? Or is the speaker commenting that the policing are driving down the road?

- *My friend plays tennis too* - Does the listener play tennis as well? Or is the speaker commenting about her/his multi-talented friend?

The intent, sentiment, tone, history etc. all contribute to the pragmatics processing making the task enormously complex.

### Discourse Processing

Discourse refers to the processing of connected sentences and it involves continuously producing and updating hypotheses proposed by the conversation. The following example should make this concept clearer:

- **Sentence 1:** *It is difficult to work here.*

  Who is the speaker? A corporate employee? What problems is she/he facing?

- **Sentence 2:** *The computer is not fast enough.*

  The speaker works on a computer. Is she/he a computer operator? The computer is most likely the problem mentioned earlier.

- **Sentence 3:** *I have to wait for hours before I can connect to the server.*

  The speaker is most likely not a computer operator if the work involves connecting to a server. Does the speaker work in the IT sector?

- **Sentence 4:** *I can't wait for so long; I have coursework too.*

  Since the speaker has coursework, she/he is most likely a student. Moreover, he/she is probably working in a lab, since the first line mentions problem at a workplace.

Discourse processing thus involves incrementally building a world based on the sequence of sentences that we process. Along with this, in discourse processing we also need to disambiguate coreferences (e.g. - "she" refers to which entity?), and sense and structure disambiguation of the discourse.

## 1.4 Word Embeddings

As a first step towards making machines understand language, we need to provide language as input to the machine. Language, composed of alphabets, needs to be converted to numbers or a sequence of numbers (known as `vectors`). We can choose to provide characters, sub-word, words, sentences as the basic units for conversion to a vector. The corresponding mapping from the unit to vector are called character embeddings, sub-word embeddings, word embeddings and sentence embeddings respectively. We call the mappings **embeddings** because in mathematics, an embedding is supposed to an instance of a mathematical structure contained within another mathematical structure. This is appropriate because we can assume there are a lot more numbers than units in a language that we can imagine.

In most cases, words strike the right balance between representing low-level information contained in characters and representing high-level information contained in sentences by combining sequences of words. For instance, note that documents as basic units of inputs to vectors is not a particularly good idea because we want to capture the sentiment incorporated in the document along with the details hidden in the sentences. To incorporate these details, we will have to descend to those substructure levels, failing which a document-level embedding is likely to abstract away these ideas. Another aspect to consider is that how easy is it to create and apply the mapping? Once again, falling back to the document embedding example, how do we create input instances for the embedding model? Do we take the entire document as the input? If so, then what about similar documents - will we have to map each possible variation to the same vector? On top of this, the space of inputs will be ridiculously huge and the embedding model is likely to be slow even if we tackled the hard problems that we mentioned. In these practical terms, it is most convenient to use words for embeddings. Consequently, much of the recent work has been focused towards development of word embeddings.

### 1.4.1 Characteristics

One of the most simplest forms of word embeddings that is possible is one-hot encoding, where each word in the vocabulary is a vector with only one element as 1 (rest elements are 0) and the dimension of the vector is the size of vocabulary. To make things more clear, consider an example:

**Example**: Suppose the vocabulary consists of 3 words - *I*, *play*, and *tennis*. Then, since the vocabulary size is 3, the corresponding embedding would map to a 3-dimensional space:

$$I \mapsto \begin{bmatrix}1\\0\\0\end{bmatrix}, \; play \mapsto \begin{bmatrix}0\\1\\0\end{bmatrix}, \; tennis \mapsto \begin{bmatrix}0\\0\\1\end{bmatrix},$$

We can choose to keep cased words as separate entities instead of lower-casing each word for the embedding as well. So $I$ could have been lower-cased and stored as $i$

for the embedding. One immediate flaw apparent in this embedding is the fact that the vector dimensions grows with the size of the vocabulary. Apart from this, this embedding technique does not capture relations between words and neither does it encode any properties of the word. To store any of these features, we would need to have another mechanism and we would need to include this mechanism in each application for better results in downstream tasks. The current focus is on developing and improving semantic tasks, and therefore it makes sense to instead incorporate these features in the embedding themselves instead of adding this machinery in the pipeline of each and every end-to-end application.

A similar embedding technique would have been to enumerate the words in vocabulary. The resulting mapping would grow proportional to the logarithm of the vocabulary size, but would still face the same problems of one-hot encoding. We need to therefore identify the properties of a good word embedding to evaluate its quality.

1. Encode sense - A word embedding should incorporate the knowledge of the sense of the word in the embedding. For example, the embedding should allow information about the morphological structure, POS tag, word meaning, gender etc. to be encoded in the vector.

2. Word sense disambiguation according to context - Given multiple senses of a word, an embedding should be able to identify the correct embedding in a given scenario. For example, for embeddings of the sentences "I caught the ball" and "The police caught the thief", an embedding technique should ideally be able to look at the sentence and create different embeddings corresponding to the "catching" and "arresting" sense of the word "caught" for generating the sentence embedding

3. Similarity - Relations between words in languages should be preserved in some sense between the vectors. Generally, when we learn an embedding, we try to minimise the error through a loss function, which puts a penalty on dissimilar words being clubbed together. In vector space, this similarity measure is naturally captured by the dot product metric. For instance, when we say that *excellent* and *outstanding* are similar, the dot product of the corresponding vectors is expected to be small compared to the dot products of antonyms or unrelated words. Similarly, vectors should allow us to capture information like $boy : girl :: man : woman$; the corresponding extension to vectors would be that the vectors $\mathbf{v}(boy) - \mathbf{v}(girl)$ and $\mathbf{v}(man) - \mathbf{v}(woman)$ are similar, where $\mathbf{v}$ is the mapping from string to vector. The relation between words can be either semantic (mentioned example) or syntactic ($work : working :: sleep : sleeping$).

Point 3 actually leads to evaluation metrics for word embeddings - *intrinsic evaluations* that subjectively judge a word embedding based on the quality of the most similar words that the embedding produces, and *extrinsic evaluations* that objectively judge an embedding based on the performance on analogy datasets, similar These are the characteristics that expected from a word embedding technique in a general setting. However, it may be possible that there are some tasks which do not need such powerful models; in those cases, simpler models could suffice.

### 1.4.2 Representation Methods

Word embedding techniques can differ in the way that they choose to encode the information associated with a word. Two particularly prominent representation methods are:

- Local representations - Each facet of information is encoded in a particular dimension of a vector. For example, suppose the $34^{\text{th}}$ dimension of the word vector encodes the gender in the word embedding. Then, $\mathbf{v}(girl)(34) = 1$ and $\mathbf{v}(boy)(34) = 0$ would mean that the word *girl* is a feminine entity while *boy* is a masculine entity. Similarly, one dimension could represent the POS tag, while another the morphological structure and so on.

- Distributed representations - Each pattern or feature is distributed over a number of dimensions and each dimension takes part in representing a number of features. This kind of representation is more complex for the human mind to visualise and neither is it more convenient to implement on a computer. Rather, this type of encoding allows greater efficiency in the way it makes use of processing abilities of networks of neuron-like computing elements [40].

### 1.4.3 Brief overview of Developments

Most of the language-processing techniques before the 1980s were a set of rule-based systems designed by hand[66][60]. Since the late 1980s and mid 1990s, natural language processing methodologies have been dominated by machine learning methods. Word embeddings were the result of rules learnt automatically through statistical inferences, which formed the subfield of statistical semantics. The distributional hypotheses [29], which says that words occur in the same context tend to have similar meanings, forms the basis for this subfield. The underlying idea that "a word is characterised by the company it keeps" was popularised by Firth 1957[28].

Amongst the most prominent methods in statistical semantics were Latent Semantic Analysis[24] and Latent Dirichlet Allocation[52][11]. Latent Semantic Analysis (LSA) uses the frequency of occurrence of words in documents to create a matrix (of word vs document) and then low-rank approximation of the matrix to get a condensed representation. Using this, one is able to compare documents on the basis of their similarity to an existing suite of documents. Latent Dirichlet Allocation (LDA), on the other hand, is used to extract the main topic from a document. In this model, a document is assumed to be made up of various topics and each word is attributable to one of the document's topics. The Dirichlet prior on the topic-per-document distribution and word-per-topic distribution encodes the intuition that a document that can be described by a small set of topics and a topic have a small set of frequently occurring words. Using Bayesian statistics, we can infer the topics in a document once the distribution parameters have been learnt from the corpus. Along with the distributional representations, there are also cluster based word representations, which induces a cluster over similar words using statistical information[48]

The statistical methods have been superseded by newer machine learning methods such as neural networks in recent years. Word embeddings, in particular, were popularised by Word2Vec (Mikolov et al. 2013) [41]. The class of word embeddings can be classified into *contextual* and *non-contextual* embeddings, where the embeddings vary according, and are invariant respectively, according to the context of the word. Under the non-contextual embedding umbrella, we have shallow network based and deep network based models, which refer to the complexity of the underlying neural networks. Contextual embeddings are almost exclusively the domain of deep network based models. Development of both these branches has been facilitated by the huge increase in the compute capabilities and the amount of data that is available for ingestion.

# Chapter 2

# Word Embedding Techniques

We will go into details of three particular word embedding techniques: Word2vec[41], ELMo[49] and BERT[23]. Word2vec was, in some sense, the pioneer of the word embedding techniques that are popular today. It consists of a simple, shallow feed-forward network that is trained to predict words in context.

## 2.1 Preliminary

### 2.1.1 Neural Probabilistic Language Model

Bengio et al. 2003 [8] pioneered a new approach to learn distributed representations of words using natural language text. Their proposed approach was summarised as follows:

1. Associate with each word in vocabulary a distributed feature vector, i.e., a real-valued vector in $\mathbb{R}^m$
2. Express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence
3. learn simultaneously the word feature vectors and the parameters of that probability function

The number of features, $m$, is much smaller than the size of the vocabulary. The probability function is expressed as the product of conditional probabilities of the next word given previous words. This function has parameters that can be tuned in order to maximise log-likelihood of the training data. The reason why this works is that given a sentence in the training data, similar words are expected to have similar vectors and since the probability function is a smooth function of the feature values, a small change in the feature vector should cause only a small change in the probability.

For instance, (`cat`,`dog`) and (`running`,`walking`) are pairs of similar words. If a sentence in a training corpus is `The cat is walking in the bedroom`, the model should be able to generalise to `The dog is running in the bedroom`. Therefore, the presence of one such sentence in the training corpus will increase the probability

of not that sentence, but of each of the sentences generated by swapping the word by a "neighbour" (according to the feature vector).

The objective is to learn a good model $f(w_t, \ldots, w_{t-n+1}) = P(w_t|w_{t-1}, \ldots, w_{t-n+1})$, for an $n$-gram model. This is implemented as a mapping from an element of the vocabulary to a distributed feature vector, and a feed-forward network over the feature vectors of $w_{t-1}, \ldots, w_{t-n+1}$. More clearly,

1. The words $w_{t-1}, \ldots, w_{t-n+1}$ are converted into feature vectors using the look-up matrix $C$ which is actually the mapping from word to a feature vectors. $C$ is shared across the words

2. The resulting feature vectors are concatenated and passed into a feedforward network. The authors implemented a hidden hyperbolic tangent layer followed by a softmax layer. The $i^{th}$ element of the resulting vector represents the predicted probability $P(w_i|w_{t-1}, \ldots, w_{t-n+1})$.

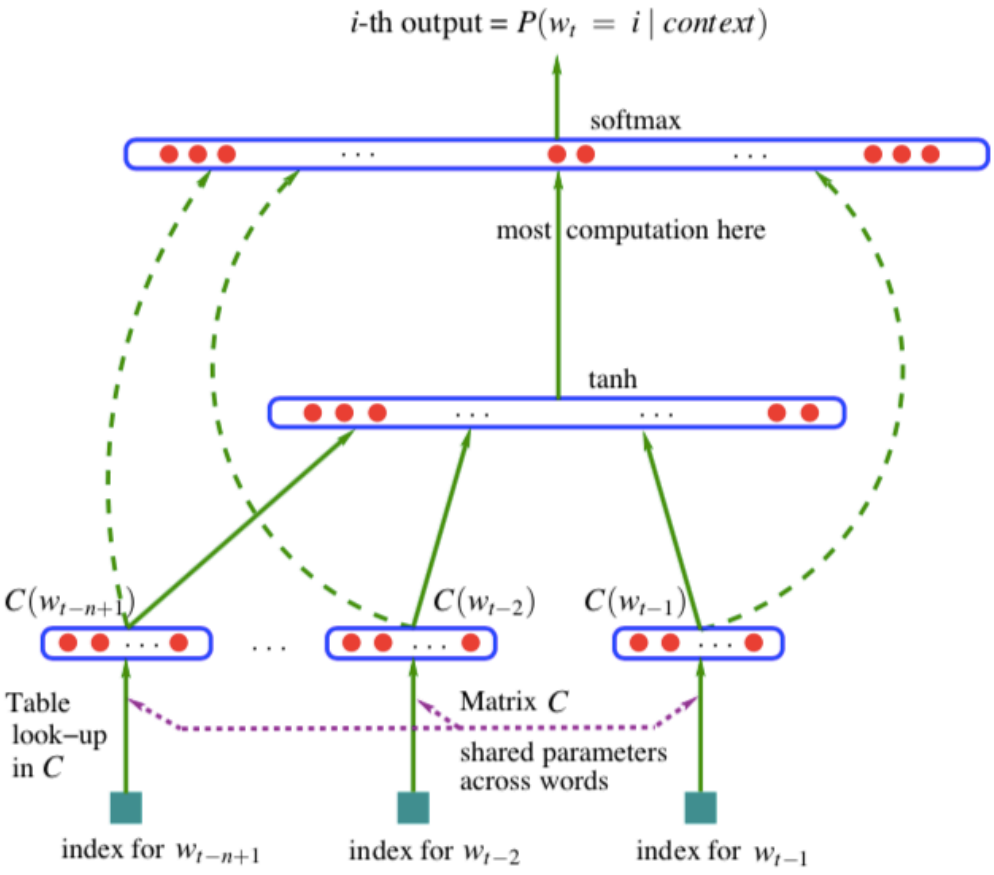



Figure 2-1: Neural Probabilistic Language Model architecture [8]

The architecture is illustrated in 2-1. The log-likelihood function that is to be

maximised for training is:

$$L = R(\theta) + \frac{1}{T}\sum_{t} \log f(w_t, \ldots, w_{t-n+1}; \theta)$$

where $R()$ is the regularisation term for the parameter $\theta$ of the function $f(\cdot)$.

## 2.2 Word2vec

The work by Mikolov et al. 2013 [42] is an improvement over the word embeddings obtained from Neural Net Language Model (Bengio et al. 2003). The aim of the models which "that might not be able to represent the data as precisely as neural networks, but can possibly be trained on much more data efficiently". The complexity of the Neural Network Language Model (NNLM) is

$$Q = N \times D + N \times D \times H + H \times V$$

where $N$ is the window size (around 10), $H$ is the hidden layer size (around 500-1000), $D$ is the feature vector size (around 300) and $V$ is the vocabulary size (greater than 20,000). The dominating term is $H \times V$, but by techniques like hierarchical softmax[44], the term reduces to $H \times \log_2(V)$. Thus most of the complexity is caused by $N \times D \times H$. In the word2vec model, the hidden layer is removed, thus reducing the bottleneck in the NNLM model.

They proposed two architectures:

### 2.2.1 CBOW

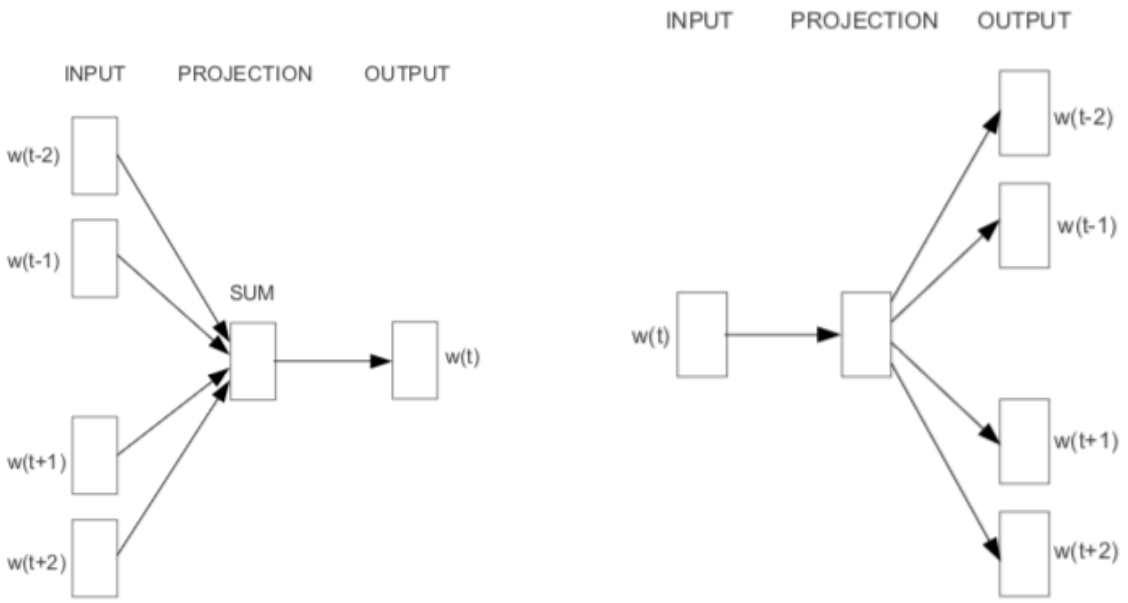
The inputs to CBOW, or continuous bag-of-words, are averaged vectors of the words surrounding the target word (both in the future and in the past), i.e., the projection layer itself is shared across the words, along with the projection matrix. The number of words for the input taken from the past and future is fixed. Given this input, the model has to predict the target word. The objective function that we maximise, formally, is:

$$\frac{1}{T}\sum_{t=1}^{T} \log p(w_t | w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c}) \tag{2.1}$$

. The training complexity is

$$Q = N \times D + D \times \log_2(|V|)$$

The architecture is illustrated in 2-2a. Since we use a continuous distributed representation, and the order of words is not relevant the architecture is called continuous bag-of-words.




(a) CBOW model[42]

(b) Skip-gram model[42]

Figure 2-2: Word2Vec models

### 2.2.2 Skip-gram

The input and output to CBOW change roles in the skip-gram model: given a word, we try and predict the context words. More precisely, we use each current word as the input to the log-linear model and predict words within a certain range before and after the current word. The complexity for this model is

$$Q = C \times D + C \times D \times \log_2(|V|)$$

where $C$ is the maximum range of words, while the log-likelihood function that we maximise in skip-gram is

$$\frac{1}{T}\sum_{t=1} T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j}|w_t) \tag{2.2}$$

where $T$ is the number of training instances. In this model, if $C = 5$, then we choose a number $R$ from the range $\{1, \ldots, C\}$ and then pluck $R$ words preceding the current word and $R$ words following the current word. Note that we make predictions for each of the $R$ words instead of a single word as in CBOW. Therefore, skip-gram models take a longer time to train. Also, the skip-gram model is better suited for learning representations of rarer words because it trains on the rarer word for more instances than in CBOW. The architecture is illustrated in 2-2b.

| Model | Vector Dimensionality | Corpus Size | Accuracy (%) | | |
|---|---|---|---|---|---|
| | | | Semantic | Syntactic | Total |
| Collobert-Weston NNLM | 50 | 660M | 9.3 | 12.3 | 11.0 |
| Turian NNLM | 50 | 37M | 1.4 | 2.6 | 2.1 |
| Turian NNLM | 200 | 37M | 1.4 | 2.2 | 1.8 |
| Mnih NNLM | 50 | 37M | 1.8 | 9.1 | 5.8 |
| Mnih NNLM | 100 | 37M | 3.3 | 13.2 | 8.8 |
| Mikolov RNNLM | 80 | 320M | 4.9 | 18.4 | 12.7 |
| Mikolov RNNLM | 640 | 320M | 8.6 | 36.5 | 24.6 |
| Huang NNLM | 50 | 990M | 13.3 | 11.6 | 12.3 |
| Mikolov NNLM | 20 | 6B | 12.9 | 26.4 | 20.3 |
| Mikolov NNLM | 50 | 6B | 27.9 | 55.8 | 43.2 |
| Mikolov NNLM | 100 | 6B | 34.2 | **64.5** | 50.8 |
| CBOW | 300 | 783M | 15.5 | 53.1 | 36.1 |
| Skip-gram | 300 | 783M | **50.0** | 55.9 | **53.3** |

Table 2.1: Performance of various models, including CBOW and skip-gram, on the syntactic-semantic dataset constructed by the authors. The total accuracy is the micro-average of the accuracy.

### 2.2.3 Performance

The performances of the models are listed in 2.1. The performance gains were complemented by significant increase in training times. On the same hardware, the speedup compared to training an NNLM model is about 7 times. It should be noted that the research at the time focused on increasing the amount of training data, but keeping the vector size relatively small. The authors also experimented with various vector sizes and concluded that an increased feature vector size (around 200-300, up from 50-100) would result in performance gains.

The NNLM model proposed by the authors was a modification to the original NNLM where they represent the vocabulary using a Huffman binary tree and use hierarchical softmax. The gains for the authors' NNLM models could have been due to the larger dataset. However, both CBOW and Skip-gram were able to perform reasonably close on a much smaller dataset. Moreover, the skip-gram model was able to produce the best results on the semantic dataset (by a large margin) and the combined dataset.

Word2vec also enabled reasoning about similarity tasks using simple algebraic operations on vector representations of words. To find a vector that is similar to *read* in the same sense *eat* is to *eating*, we can simply compute the vector $V = \mathbf{v}(eating) - \mathbf{v}(eat) + \mathbf{v}(read)$, and find the vector closest to $V$ using cosine distance and map it back to the vocabulary. High dimensional vectors trained on a large corpus were even able to capture semantic relations in the same fashion - like France is to Paris as Germany is to Berlin.

### 2.2.4 Improvements

There were several improvements made over the skip-gram model made in [42]. The authors included formulations to speed up the model and increase the accuracy by significant margins. The probability function computation in 2.2 is defined in the basic skip-gram formulation as:

$$p(w_O|w_I) = \frac{\exp(\mathbf{v}_{w_O}^\intercal \mathbf{v}_{w_I})}{\sum_{w=1}^{V} \exp(\mathbf{v}_w^\intercal \mathbf{v}_{w_I})}$$

where V is the vocabulary set. The summation over the entire vocabulary is the most intensive part computationally, which is what the authors tried to improve upon.

#### Hierarchical Softmax

The idea in hierarchical softmax is break the words into clusters to reduce the number of normalisations required to compute the softmax[44]. As a concrete example, suppose a particular set of 10,000 values, which Y can take, is broken into 100 partitions of size 100 each. Then, computing the probability $P(Y = y|X = x)$, which originally required normalising over all possible 10,000 values Y can take, can be rewritten as $P(Y = y|C = c(y), X = x) \cdot P(C = c(y)|X = x)$, which reduces the number of normalisations to 2 times 100 - once to normalise $Y$ against all possible clusters it can belong to and once to normalise the cluster to which $y$ belongs against all possible values in that cluster. Note that this was possible because of the function $c(\cdot)$ being a partition function. In the same spirit, hierarchical softmax uses a binary tree representation of the output layer, with the vocabulary words as its leaves. The tree in this case defines the multiple layers of partition. Each intermediate node is assigned a feature vector, and a walk assigns probabilities to the word. The complexity involved in a softmax calculation reduces from $|V|$ to $\log_2(|V|)$.

Formally, each word $w$ can be reached by an appropriate path from the root of the tree. Let $n(w, j)$ be the $j^{th}$ node on the path from the root to $w$, and let $L(w)$ be the length of this path, so $n(w, 1) = root$ and $n(w, L(w)) = w$. In addition, for any inner node $n$, let $ch(n)$ be a fixed child of n (decided arbitrarily) and let $[x]$ be 1 if $x$ is true and -1 otherwise. Then the hierarchical softmax defines $P(w|w_I)$ as follows:

$$P(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma\left([\![n(w, j+1) = ch(n(w, j))]\!] \cdot {v'_{n(w,j)}}^\intercal v_{w_I}\right)$$

where $\sigma = 1/(1 + \exp(-x))$. As an example, refer to 2-3. The softmax probability for $w_1$ given $w_4$ would be $P(w_1|w_4) = \sigma(v_1^\intercal v_{w_4}) \cdot \sigma(-v_2^\intercal v_{w_4})$. It is apparent that the computation requires $\log_2(|V|)$, and so does the gradient of the probability, required for backpropagation.

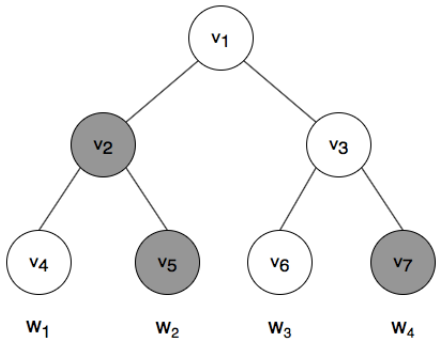



Figure 2-3: Hierarchical softmax computation illustration. Highlighted nodes are the arbitrarily chosen fixed nodes

### Noise Contrastive Elimination

Noise contrastive elimination (NCE) aims to turn a categorical classification problem into a binomial classification problem. In context of the skip-gram model, we change the model as follows:

- the input is a pair of words $(w_t, w_c)$, i.e. the target and context word, instead of a single word
- Following the embedding layer (the word to vector mapping), we take the dot product of the two vectors
- The dot product is the input to a sigmoid layer, which scales the dot product to a value between 0 and 1
- The output is compared against the true output and the error is back propagated

The training instances are created using the corpus (corresponding output - 1) and a noise distribution (corresponding output - 0). Therefore, instead of calculating the softmax function to compute probabilities over the vocabulary, we predict whether the given instance belongs to the corpus or not. The idea is that the model learns to distinguish the target word $w_O$ from draws from the noise distribution. The training objecting for NCE approximately maximises the log probability of the softmax, which is equivalent to saying that we want to push the probabilities of target word in the original softmax to 1.

### Negative Sampling

In negative sampling, we choose some $k$ words as negative samples, using draws from the noise distribution, and assuming the set of $k+1$ words ($k$ negative samples and the true output) to be the only words in the vocabulary, compute the softmax. Therefore, we reduce the computation requiring the entire vocabulary down to a constant number of words.

The objective function for negative sampling does not actually maximise the log probability of the softmax function. In fact, it does not relate to the softmax function at all. But, empirically, the results using negative sampling have been positive.

#### Subsampling frequent words

Words which appear very frequently, like the word "the", do not usually carry significant information. For instance, co-occurrence of "the" with "mango" does not tell us anything special about mangoes; "the" is as likely to occur with any other word. To counter this imbalance between rare and frequent words, a subsampling approach is applied, where a word in the training set is discarded with probability that rapidly increases with the frequency of the words above a certain threshold. This makes the learning process faster, since we have to train on fewer samples and heuristically, improves the quality of vectors of rare words.

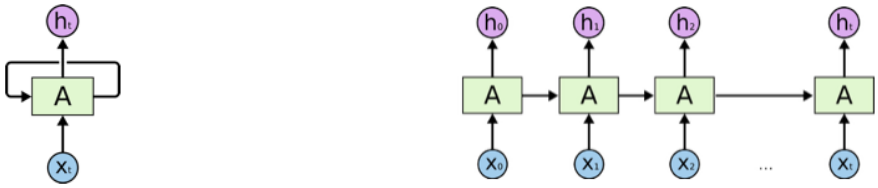
## 2.3 ELMo

Embedding from Language Models (ELMo), proposed in [49], provides contextual word embedding, in contrast to non-contextual word embedding techniques like wordv2ec. What this means is that the feature vector representation of a word can change according to the context of a word - the representation of the word `tank` in the two sentences `The water tank is leaking` and `Panther was a famous tank used in WWII` will refer to the corresponding meanings and therefore be different.

### 2.3.1 LSTM

ELMo uses an architecture known as LSTM (long-short term memory)[31], which is a sequential model (i.e., it is not parallelisable) and retains memory of past inputs to predict the next time series output based on the current input. As an example, if we build a simple character predicting model using LSTMs, we feed characters as inputs to the LSTM and we train it to predict the next character. Therefore, if the input character at time step $t$ was "o" and the previous characters were "I'm feeling go", given the training sample is "I'm feeling good today", the LSTM will be trained to output "o" as the output. At the next time step, the input will be the second "o" and expected output should be "d". After testing, to generate an output, the output at time step $t$ is fed as the input for time step $t + 1$. The model is initialised with a seed character to start the generation.

The architecture of LSTM is a more sophisticated version of the architecture of a vanilla Recurrent Neural Network (RNN), which captures the idea of a time-series input and output. To facilitate a time-series modelling in an RNN, we essentially have a loop in the network, which channels the computation/output from a previous time step as one of the inputs to the next time step, which can be represented as the network shown in 2-4a. The time-lagged input can be represented as an unrolling of the network, illustrated in 2-4b. Therefore, the same unit computes the entire

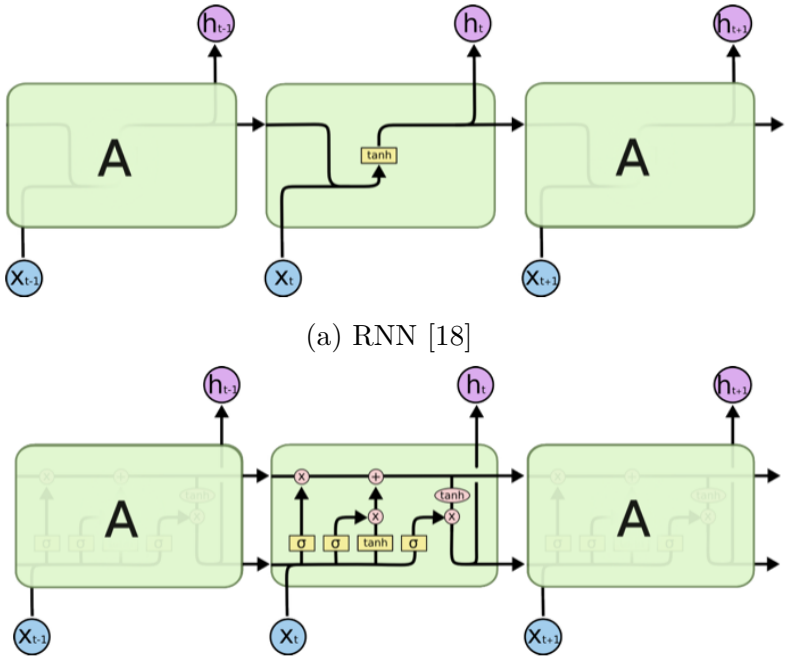



(a) Loops in RNN [18]

(b) Unrolled RNN [18]

Figure 2-4: Recurrent Neural Networks

time-series output and the parameters of the computation are the same across all time-steps. The place where vanilla RNNs and LSTMs differ is the neural network inside the unit that is used for computing the outputs. Vanilla RNNs have a single hyperbolic tangent layer as the network, while LSTMs have a combination of four separate layers to that decide which parts of input to discard, which parts of internal state to forget and update, and what to output. This particular combination of layers in an LSTM allows the model to learn associations across longer time steps. The contrasting architectures are shown in 2-5.




(a) RNN [18]

(b) LSTM [18]

Figure 2-5: Internal Layers in RNN and LSTM

Training for LSTMs and RNNs use a modified back-propagation technique, called truncated back-propagation [31][65]

### 2.3.2 Model

At the heart of ELMo is the bidirectional language model, which aims to predict a word given both the future and the past words. Training ELMo takes as input a word, computes its token representation through a neural network model, passes the representation through layers of forward and backward layer LSTMs (bidirectional LSTM model, or biLSTM), and then finally combines the outputs of the top layer in a softmax to predict the target word. The function maximising the log-likelihood of forward and backward language models, which is used as the optimisation function in the training can be described as follows:

$$\sum_{k=1}^{N} \Big( \log p(t_k | t_1, \ldots, t_{k-1}; \Theta_x, \Theta_{F-LSTM}, \Theta_s) + \log p(t_k | t_{k+1}, \ldots, t_N; \Theta_x, \Theta_{B-\vec{LSTM}}, \Theta_s) \Big)$$

where $\Theta_x$ is the parameter for token representation, $\Theta_{F-LSTM}$ and $\Theta_{B-LSTM}$ are the parameters for the forward and backward LSTM layers respectively, and $\Theta_s$ is the parameter for the softmax layer. The optimisation increases the probability of occurrence of the target word given all the other words in the sentence.

The architecture is illustrated in 2-6. The LSTM layers (both forward and backward) are 2 layers deep in ELMo. Once training is complete, the weights of the LSTM layers are frozen and not changed further. For each task, we then learn a *linear combination* of the output of the LSTM layers and the original embeddings, and use this as a complement to existing inputs to the pipeline. To be more concrete, for a downstream NLP task, suppose we have a model which takes as input an embedding of the token $x$. The embedding can be obtained using any methodology, like word2vec, GloVe etc. We pass the input sentence through the ELMo model, and obtain the output from both the forward and backward LSTM layers. In the authors' construction, we obtain 4 such vectors (2 from the forward LSTM layers, and 2 from the backward LSTM layers) along with the input token representation to ELMo, which are transformed into 3 vectors, by concatenating outputs of the same layer. We obtain a single vector using a linear combination of these 3 vectors, the parameters of the combination being learnable. This vector is concatenated to the original input to the task model and the training is done end-to-end, learning the parameters of the linear combination. Formally, for each token $t_k$, an $L$-layer LSTM will compute a set of $2L + 1$ representations:

$$\begin{aligned} R_k &= \{\mathbf{x}_k, \mathbf{h}_{k,j}^{For.LSTM}, \mathbf{h}_{k,j}^{Bac.LSTM} | j = 1, \ldots, L\} \\ &= \{\mathbf{h}_{k,j} | j = 0, \ldots, L\} \end{aligned}$$

where $\mathbf{h}_{k,0}$ is the token layer and $\mathbf{h}_{k,j} = [\mathbf{h}_{k,j}^{For.LSTM} : \mathbf{h}_{k,j}^{Bac.LSTM}]$ for each biLSTM layer. For each task, we compute a task specific weighting of all biLM layers:

$$\mathbf{ELMo}_j^{task} = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}$$

where $\mathbf{s}^{task}$ are softmax-normalised weights and $\gamma^{task}$ is used for scaling the entire ELMo vector.

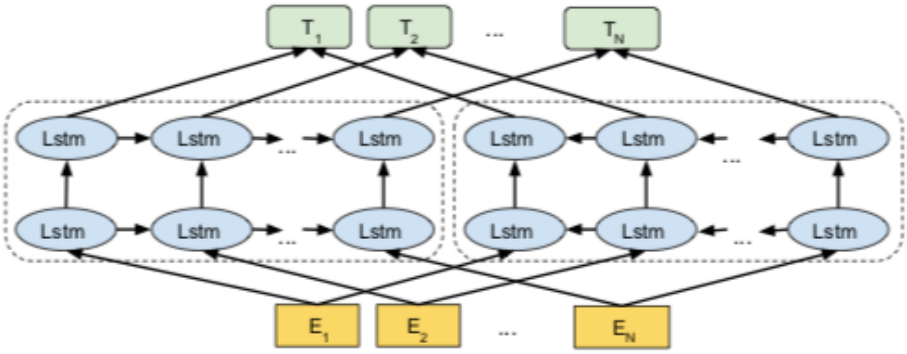



Figure 2-6: ELMo Model: Each word is a time-series input to forward and backward LSTM layers, and the output is the combination of the top layer for training purposes [23][49]

### 2.3.3 Performance

Using ELMo embeddings, the authors were able to beat state-of-the-art models across a set of 6 diverse benchmark NLP tasks. The different tasks and the datasets are:

1. Question Answering - Stanford Question Answering Dataset (SQuAD) [55]
   The task is to find a span of text in a paragraph which answers the given question. SQuAD contains 100K+ crowd-sourced question-answer pairs where the answer is a span in a given Wikipedia paragraph. The authors' baseline is a modified version of Bidirectional Attention Flow Model in [43].

2. Textual Entailment - Stanford Natural Language Inference (SNLI) [13]
   The task is to determine whether a hypotheses is true given the premise. The SNLI corpus provides approximately 550K hypotheses/premise pairs. The authors' baseline is a modification to the sequence model from [15].

3. Semantic Role Labelling - OntoNotes[51]
   The task is to answer the question "who did what to whom". The authors' baseline was a re-implementation of the model described in [30]

4. Coreference Resolution - OntoNotes coreferences from CoNLL 2012 shared task[51]
   The task is to identify and cluster words that refer to the same entity in the text/world. The authors' baseline was the end-to-end span neural network described in [34].

5. Named Entity Recognition - CoNLL 2003 NER task[59]
   The task is to identify proper nouns in a text. The dataset consists of newswire

from the Reuters RCV1 corpus tagged with four different entity types (PER, LOC, ORG, MISC). The authors' baseline is a biLSTM-CRF based model.

6. Sentiment Analysis - Stanford Sentiment Treebank (SST-5)[63]
   The task involves classifying the text into one of the specified categories based on the sentiment. The baseline model is the biattentive classification network (BCN) from [38]

### Analysis

Using intrinsic evaluation on different tasks such as Word Sense Disambiguation tasks and POS tagging tasks, it was confirmed that ELMo does indeed provide context-dependent word vectors. Additionally, it was determined that syntactic information was captured in the lower layers, while semantic information was learnt in the higher layers, which is similar to the pattern seen in Machine Translation encoders.

## 2.4 BERT

BERT is a model which is designed to pretrain deep bidirectional representations from unlabelled text. The resulting model size is huge, and requires a lot of resources to train but is fairly simple to extend to NLP tasks by attaching an output layer and training end-to-end on the task. BERT uses the Transformer encoder at its core, which allows it to use self-attention mechanisms.

### Attention

Attention is the capability of a decoder system to decide which parts of the source to focus on rather than use a single representation of the input to create the outputs. This allows information to be be spread throughout the sequence of annotations in the source sentence which can be selectively retrieved by the decoder. Introduced in [7], the mechanism allowed comprehension to be stable even for sentences longer than 50 words compared to the 30 word empirical limit of previous RNN based methods.

### Self-attention

Self-attention extends the concept of attention by allowing the encoder and decoder systems to selectively focus on other encoder and decoder outputs. For example, in the sentence `The animal didn't cross the street because it was too wide`, we would ideally want the contextual encoding of `it` to depend on `street`, while for the sentence `The animal didn't cross the street because it was too tired`, we would want `it` to be influenced by the word `tired`. Such attention mechanisms would improve the quality of word embeddings and potentially allow better retention and comprehension of text. Among the first works in self-attention was done in [16].

### 2.4.1 Transformers

Transformer is an encoder-decoder architecture, which removes the recurrence and convolutional steps involved in earlier models. The resulting architecture uses only attention mechanisms, and is much more parallelisable.

#### Attention

The transformer uses a scaled dot product attention mechanism. For a single vector which has to attend to a series of feature vectors, suppose we have a query vector $\mathbf{q}$ for the former vector and a set of key $\mathbf{k}_{1\ldots n}$ and value vectors $\mathbf{v}_{1\ldots n}$ for the latter sequence of vectors. The aim is to produce a vector which is formed by selectively focusing on the elements of sequence of vectors. The weights for the combination of the sequence of vectors can be calculated using the `softmax` function over the dot product of query and key vectors. Then, the resulting vector would simply be the weighted average of the value vectors where the weights are taken from the softmax evaluation. Formally, this means the vector due to the attention mechanism is

$$\sum_{i=1}^{n} \text{softmax}(\mathbf{q}^\intercal \mathbf{k}_i) \cdot \mathbf{v}_i$$

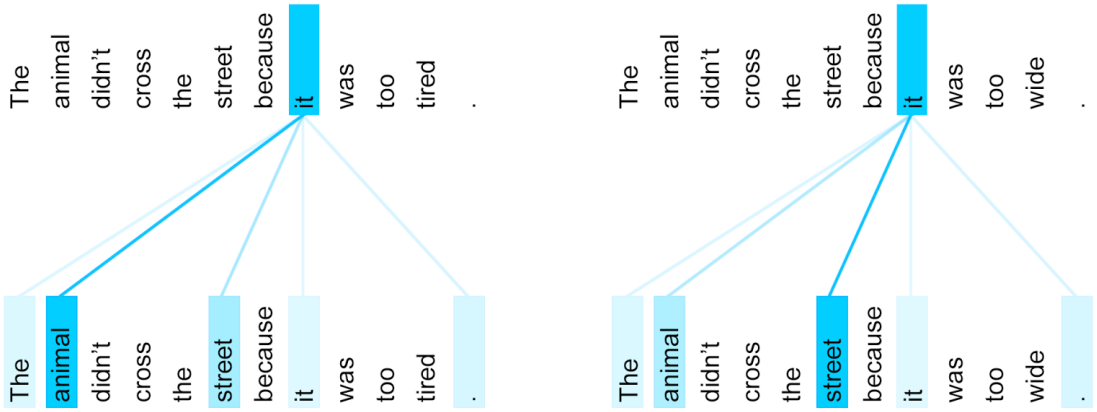



Figure 2-7: Encoder self-attention distribution for the word "it" from the 5th to the 6th layer of a Transformer trained on English to French translation (one of eight attention heads)[36]

When the input is a sequence of vectors, i.e. matrix $X$, with a corresponding query matrix $Q$ and the key and value matrices $K = [\mathbf{k}_1, \ldots, \mathbf{k}_n]$ and $V = [\mathbf{v}_1, \ldots, \mathbf{v}_n]$, the set of resulting vectors is simply

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\intercal)}{\sqrt{d_k}}\right) V$$

where $d_k$ is the dimension of the keys. The factor of $1/\sqrt{d_k}$ is present to scale the dot products. The matrices $Q$, $K$, $V$ are obtained by multiplying the input matrix $X$ with matrices $W^Q$, $W^K$, $W^V$ which are learnt during training.

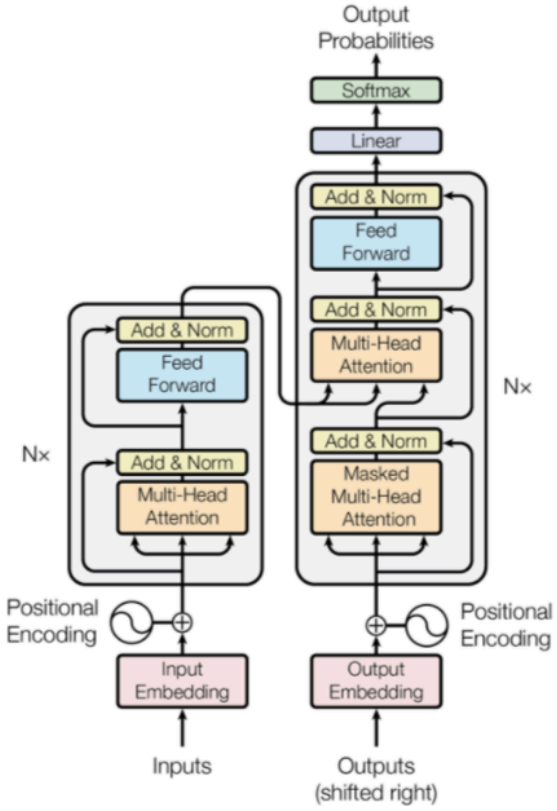



Figure 2-8: Transformer architecture. Note the attention to the encoder output at the second sub-layer. Also note the positional embeddings at both the input and output.[33]

### Multiheaded Attention

Instead of performing a single attention function, multiple such functions are computed for the same input. Each of these computations is a separate "attention head". The result of these attention heads is concatenated, and then linearly combined by multiplication with a matrix $W^O$, which is learnt during training, to allow intermixing and to scale back the vector to a uniform size.
Note that the matrix operations are what makes this model parallelisable.

### Encoder

The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sublayers: the first sublayer is a multi-head attention mechanism, while the second is a fully-connected feedforward network. The dimensions of output at each layer is

the same. A particular instance of the attending mechanism learnt by the encoder is visualised in 2-7.

**Decoder**

The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers similar to the encoder sub-layers, we also have a sub-layer that attends to the encoder output. The self-attention mechanism needs to be modified so that at position $i$, we only look at outputs known at positions less than $i$ - the positions beyond $i$ have not yet been computed.
Note that the matrix formulation of the input removes the information about the sequence of words present in the sentence. Naturally, we want to be able to enforce that words should be in a particular order. Therefore, we add positional embeddings to the input embedding.

The entire transformer architecture is represented in 2-8.

### 2.4.2 Model

The BERT model is simply a stack of multiple transformer encoders. To make BERT handle a variety of downstream NLP tasks, the input representation was crafted to unambiguously represent a single sentence or a pair of sentences (eg - <Question, Answer>) - a sentence can be an arbitratily long piece of contiguous text, rather than an actual linguistic sentence. Specifics of the input are:

- The first token of every input sequence is a special classification token `[CLS]`. The final hidden state corresponding to this token is used for aggregate sentence representation for classification tasks (eg - sentiment analysis)
- Sentence pairs are separated by a special token `[SEP]`
- A learned embedding is added to every token, denoting whether the sentence belongs to sentence `A` or `B` for sentence pairs
- Position embeddings are added to each token, as required by the transformer architecture2.4.1

There were two variants of the BERT model proposed by the authors:

1. $\text{BERT}_{\text{BASE}}$ - 12 transformer blocks, 768 dimensions of feature vector (hidden size), and 12 attention heads
2. $\text{BERT}_{\text{LARGE}}$ - 24 transformer blocks, 1024 dimensions of feature vector (hidden size), and 16 attention heads

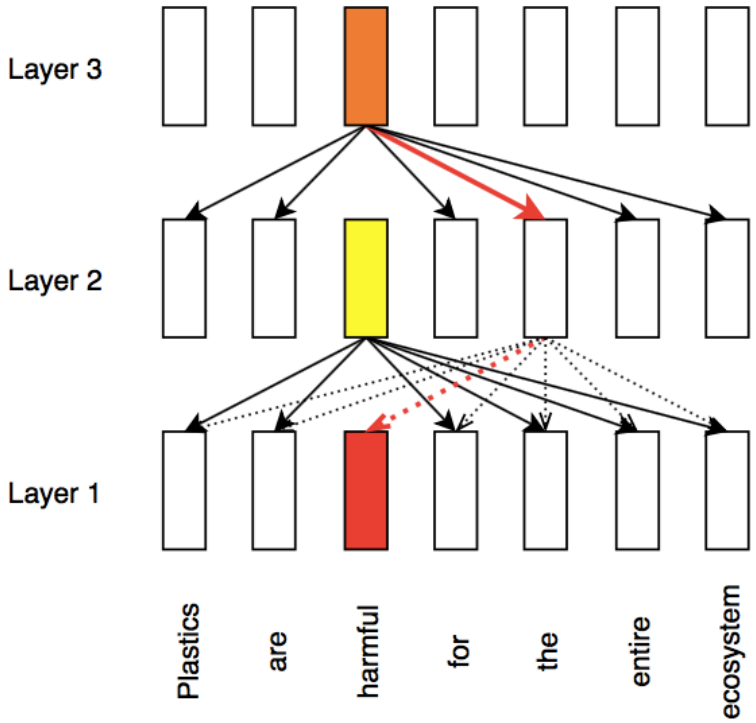



Figure 2-9: Illustration of why traditional bidirectional language modeling (where there is no masking) fails. Notice how the third layer can "see" the target word using the bidirectional context from the second layer

### Pretraining

The pretraining phase of the model involves training on unlabeled data over two pretraining tasks:

- Masked Language Model: BERT aims to employ a deep bidirectional context learning approach. If we try to do this using traditional approaches, then the model would be able to see the word in a multi-layered context (2-9). This is precisely why traditional approaches allow only *left-to-right* or *right-to-left* approaches. To avoid this, in BERT, some percentage of input tokens are masked at random and the task is to predict those masked inputs. The final hidden vectors corresponding to *only* the mask tokens are fed into an output softmax over the vocabulary (similar to a standard language modelling task). Formally, if the original word sequence is $\boldsymbol{x}$, BERT creates a corrupted version $\hat{\boldsymbol{x}}$ by setting a portion of tokens in $\boldsymbol{x}$ to a special symbol `[MASK]`. If the masked tokens are $\bar{\boldsymbol{x}}$, then the training objective is to reconstruct $\boldsymbol{x}$ given $\hat{\boldsymbol{x}}$[67]:

$$\max_{\theta} \log p_\theta(\bar{\boldsymbol{x}}|\hat{\boldsymbol{x}}) \approx \sum_{t=1}^{T} m_t p_\theta(x_t|\hat{\boldsymbol{x}}) = \sum_{t=1}^{T} m_t \log \frac{\exp(H_\theta(\hat{\boldsymbol{x}})_t^\intercal e(x_t))}{\sum_{x'} \exp(H_\theta(\hat{\boldsymbol{x}})_t^\intercal e(x'))}$$

where $m_t = 1$ indicates $x_t$ is masked, and $H_\theta$ is a block of transformers that

maps a length-$T$ text sequence $\boldsymbol{x}$ into a sequence of hidden vectors $H_\theta(\boldsymbol{x}) = [H_\theta(\boldsymbol{x})_1, H_\theta(\boldsymbol{x})_2, \ldots, H_\theta(\boldsymbol{x})_T]$. $H_\theta(\boldsymbol{x})_t$ is the $t^{th}$ vector in the sequence $H_\theta(\boldsymbol{x})$. There are two disadvantages of this approach - one is that the model implicitly assumes that the masked tokens can be independently reconstructed, which might not be true in general. For example, if the sentence `She likes mangoes` is corrupted to `[MASK] [MASK] manoges`, the reconstructions of the first two tokens are obviously not independent of each other. The other disadvantage is that there are no `[MASK]` tokens in fine-tuning (process after pre-training), and that creates a mismatch between the two phases. To mitigate this, if the $i^{th}$ token is chosen, the authors recommend replace the $i^{th}$ token with (1) the [MASK] token 80% of the time (2) a random token 10% of the time (3) the unchanged $i^{th}$ token 10% of the time.

- Next Sentence Prediction: In order to understand sentence relationships, which are the basis of some downstream NLP tasks like Question-Answering, but the idea of which is not directly captured by language modeling, the model is trained on a *next sentence prediction task*. Using a monolingual corpus, we create pairs of sentences `A` and `B` such that 50% of the time `B` is the actual next sentence that follows `A` (labeled as `IsNext` - expected output of token corresponding to `[CLS]`), and 50% of the time it is a random sentence from the corpus (labeled as `NotNext`).

When training the BERT model, Masked LM and Next Sentence Prediction are trained together, with the goal of minimizing the combined loss function of the two strategies.

**Fine tuning**

For each task, we simply supply the task-specific inputs to the BERT model and train (fine-tune) on all parameters end-to-end. At the input, sentence `A` and `B` from pre-training correspond to sentence pairs in paraphrasing, hypothesis-premise pairs in textual entailment, degenerate text-$\phi$ pair in text classification or sequence tagging. At the output, the token representations are fed into an output layer for token-level tasks, such as question-answering, and the `[CLS]` representation is fed into an output layer for classification, such as entailment or sentiment analysis.

### 2.4.3 Performance

BERT, specifically, $\texttt{BERT}_{\texttt{LARGE}}$ improves performance over 11 NLP tasks, including but not limited to sentiment analysis, questions answering, sentence continuation. Of interest here is to note that BERT has surpassed humans and experts in some of these tasks, like question answering and situations with adversarial generation (where given an input sentence, the most probable continuation of the sentence has to be chosen from 4 options).

Through ablation studies of the effect of model size, the authors also showed that scaling to extreme model sizes leads to large improvements in small-scale tasks provided the model has been sufficiently pre-trained.

# Chapter 3

# Applications of Word Embeddings

Word embeddings are used in almost all NLP tasks today. Even multimodal systems like image captioning system, visual question answering systems involve NLP along with Computer Vision. Google keypad, for instance, predicts the next probable words on the basis of the word currently typed. This is accomplished a model called seq2seq learning, which takes word embeddings as input. Along with next word prediction, word embeddings are also used as inputs to models for music recommendation systems, machine translation, sentiment analysis among others.

Word embeddings can be used in multiple architectures like Recurrent Neural Networks (RNN), Long-short Term Memory (LSTM) networks, Convolutional Netural Networks (CNN) etc. Each of these architectures are suited to particular form of data and are useful for a particular set of tasks. For instance, RNN is useful for processing sequential data, while CNN is good for processing data that varies spatially (like images). We focus on Feedforward networks and CNNs in this chapter.

## 3.1 Feedforward Networks

A neural network is a massively parallel distributed processor that has a natural propensity for storing experential knowledge and making it available for use. It resemebles the brain in two aspects:

- Knowledge is acquired by the network through a learning process
- Interconnection strengths known as synaptic weights are used to store the knowledge

Learning is simply a process by which the free parameters (i.e. synaptic weights and biases) of the neural network are adapting through a continuing process of simulation by the environment the network is embedded in.

Feedforward networks are neural networks in which there are no loops in the connection between the nodes in the network, i.e., there are no feedback connections.

The inputs to the feedforward networks feed as inputs to some *input neurons*. The output of each input neuron is in turn input to multiple *hidden neurons*. A hidden

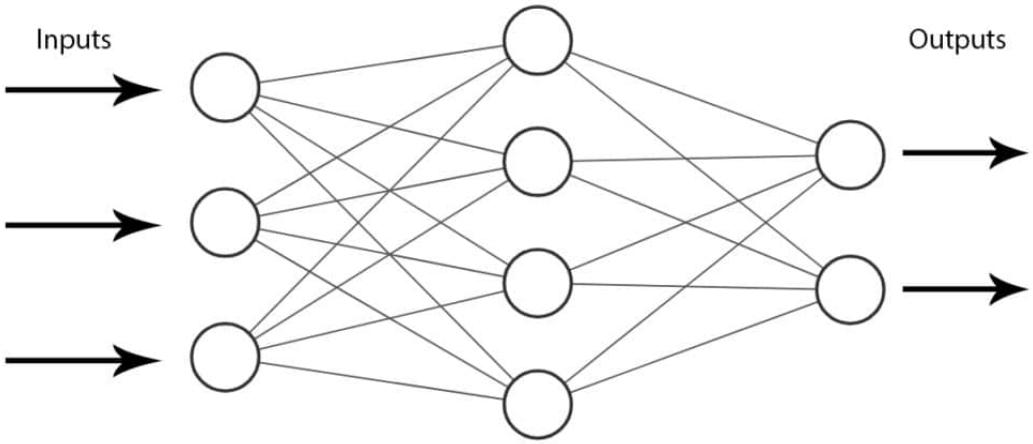



Figure 3-1: Sample feedforward network

neuron can then function as an input module for other hidden neurons or *output neurons*. The output of these output neurons are defined to be the output of the entire network.

To compute the output of a single neuron, the inputs feeding into the neuron are combined in a linear fashion (multiplied by weights and added, along with a bias term) and a non-linear function is applied to this linear combination. The inputs can from either inputs directly or neurons, in which case the output of the neuron is the input to the current one. The non-linear function is called the activation function and each neuron has an activation function associated with it. Therefore, if the inputs to the neuron are $I_1, I_2, \ldots I_k$, the parameters for linear combination are the weights $w_1, w_2, \ldots w_k$ and the bias $b$ and the activation function is $A(\cdot)$, the output of the neuron is $A(b + \sum_{i=1}^{k} I_i \cdot w_i)$. Therefore, starting from the inputs, we can calculate the output of each neuron in the network. This means that we must traverse the neurons in a topological manner so that all inputs of each neuron have been computed before trying to compute its output. The output of the network is simply the list of outputs of the output neurons.

However to get the proper weights and biases for each neuron, we need to first learn the values of these parameters so that we can approximate the required function. To do this, feedforward networks employ back-propagation learning, which consists of the following phases:

1. Calculate output of the network - Calculate the output of the network for the training input

2. Calculate error - The output of the network will rarely match the actual output. Calculate the error using an aggregated error function such as sum squared error and for each parameter, compute the gradient of the error with respect to the parameter. Computation of the gradient needs to be in reverse topological order (hence the name back-propagation).

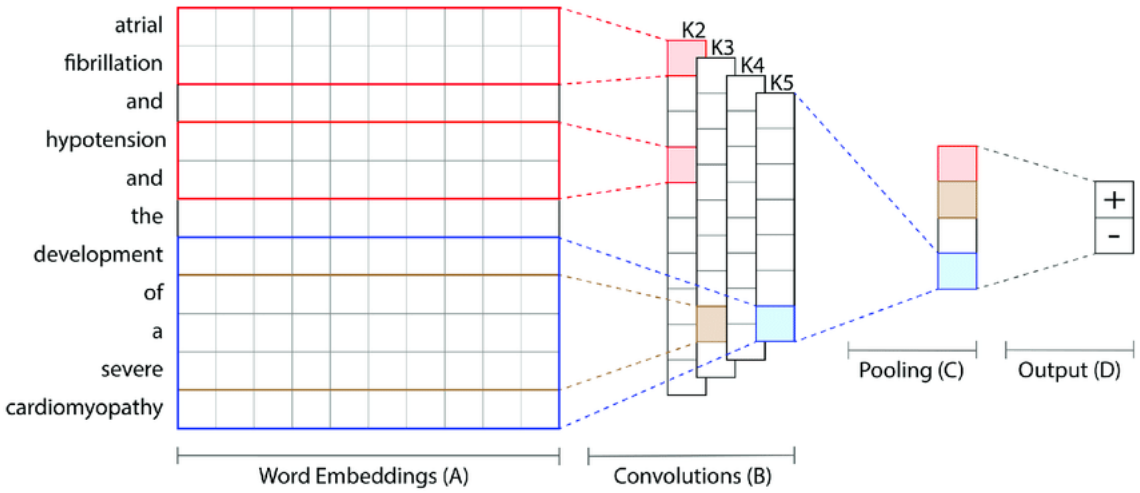



Figure 3-2: Sample 1D convolutional network for NLP

3. Adjust weights and biases - Using the computed gradients, adjust the weights and biases by adding a fraction of the gradient to the current value. This is called gradient descent.

Once all training instances are run, the training is complete.

## 3.2 Convolutional Neural Networks

Convolutional Neural Networks (or CNNs) are a subclass of feedforward neural networks. CNNs, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output.

The difference stems from the fact that CNNs operate over multi-dimensional matrices (e.g. - 3 channel image). Convolutional filters are simply convolutions of a filter matrix with the multi-dimensional matrices. A convolutional layer is a set of independent filters. Each filter in a convolutional layer is independently convolved with the image to get a multi-dimensional matrix as an output. The weights of the filters are learnt using the same back-propagation algorithm.

The convolutional layer serves to extract features from the input layers. The features captured become progressively complex as we increase the stack of convoltional layers in the network. Along with convolution layers, we also use pooling layers, which simply takes the maximum element in the input layer covered by each kernel placement. This leads to a reduction in dimensionality and therefore a decrease in computational power required to process the data.

In general, CNNs work well with data that have a spatial relationship, an immediate example of which is image. In textual domain, spatial relationships are apparent in order relationship between words in a document or text. Therefore, CNNs can be used for tasks such as document classification.

# Chapter 4

# Word Embeddings for Indian Languages

## 4.1 Prior resources

India has 22 constitutionally recognised languages with a combined speaker base of over 1 billion people. Despite such a large speaker base, the resources for Indian languages readily available for research and computational purposes is quite limited. With an an increase in connectivity and rise of smart technologies, the demand for NLP tools for Indian languages is expected to increase with each new user on the Internet.

Existing corpora for Indian languages, especially monolingual corpora, are not easily obtained. Even in cases where they do exist, the corpus is not as exhaustive as that for English or other European languages. Some sources for corpus are:

- CIIL corpus
- ILCI corpus
- CommonCrawl
- Wikipedia dumps

## 4.2 Evaluating embeddings

Even combined, the amount of data stored in these sources are no match for the data existing for English: BERT was trained on over 2.2 billion words [23], while the existing corpus of Hindi has only 3 million words and 14 Indian languages combined have amount to under 10 million words (see 5.1).

Given that we train embeddings on these sources, the question is how do we evaluate the quality of those embeddings? Once again, the evaluation tasks stand pale in comparison to those for European languages, with no datasets like adversarial sentences, textual entailment etc. close to no-existent for Indian languages. However, the possible evaluation metrics are:

- Word similarity: Finding the cosine similarity between words that are close in meaning to each other. Words that mean almost the same should have embeddings that are spaced close by in the vector space too. This would help to determine if the embeddings capture the semantics of the words.

- Word analogy: Given a relation A:B :: C:D like Man:King::Woman:Queen, can the model predict the relation? This is done by taking the word vectors for $A$, $B$ and $C$ and checking if $\boldsymbol{v}(B) - \boldsymbol{v}(A) + \boldsymbol{v}(C)$ is closest to $\boldsymbol{v}(D)$ or not. Again,this is useful to determine if the word embeddings can capture the semantics of the language

- Language modelling: Given the first $(n-1)$, can the $n^{th}$ word be predicted?

- PoS/NER tagging: Given a sentence as input, with the word embed-dings of each word representing each word, can a PoS/NER tagger give proper tags to all words?

## 4.3 Available data

To illustrate the point made in the last section, the statistics for the corpus that we have is given in 5.1.

| Language | Sentence Count | Word Count |
|---|---|---|
| Hindi | 48,115,256 | 3,419,909 |
| Bengali | 1,563,137 | 707,473 |
| Telugu | 1,019,430 | 1,255,086 |
| Tamil | 881,429 | 1,407,646 |
| Nepali | 705,503 | 314,408 |
| Sanskrit | 553,103 | 448,784 |
| Marathi | 519,506 | 498,475 |
| Punjabi | 503,330 | 247,835 |
| Malayalam | 493,234 | 1,325,212 |
| Gujarati, Konkani, Odia, Assamese, Kannada (each) | $< 500{,}000$ | $< 500{,}000$ |

Table 4.1: Corpus statistics

Compared to resources for English, the training data available for Indian languages is orders of magnitude lower. The distribution of data is not uniform in Indian languages too. Hindi has the most resources, with around 3.5 million words. The nearest language in terms of resources has 5 times smaller resources compared to Hindi. Other languages hover around the 1 million or 5 lakh mark. Most of the languages have less resources, between the range of 2.5 lakh to 5 lakh words

A point to be noted is that the table does not list all the languages spoken and written in India (including the 22 recognised languages). Some examples include

Urdu, Bhojpuri, Magahi. This goes on to show how many languages are still under-represented in the research community and which could benefit from research.

# Chapter 5

# NER for Indian Languages using Word Embeddings

Indian languages have a different system of representing proper nouns compared to some European languages. For example, it is a rule to capitalise the first letter of the proper noun in English, while in German first letter of all nouns are capitalised. In contrast, there are no separate capitalised alphabets in the Indian language system. Moreover, common nouns are frequently chosen as names of objects in Indian languages - a phenomenon which is much less common in other languages. This leads to greater ambiguity in the former set of languages.

## 5.1 Previous work

Earlier works have used a combination of rule-based, semi-supervised, supervised and unsupervised models for the NER task. [25] used a 10-fold cross-validation SVM model to get an F-score of 91.8 for Bengali, while [6] used a BiLSTM model to achieve an accuracy of 77.48 on ICON NLP Tools 2013 dataset for Hindi. [61] provides a comprehensive list of the different models implemented till 2016.

## 5.2 Our work

For the purpose of NER tagging, we collected data for pre-training from multiple sources and the resulting statistics for corpus of different corpus are listed in 5.1

Different embedding models were trained on this corpus, including word2vec[42], FastText[12], GloVe[47], ELMo[49] and BERT[23].

### 5.2.1 GloVe

GloVe embeddings of dimensions 50, 100, 200, and 300 were created from the dataset available. Words with frequency less than 2 were not included in the library. For the symmetric window size for the co-occurrence matrix, 15 was chosen according to the original GloVe[47] paper.

| Language | Sentence Count | Word Count |
|---|---|---|
| Hindi | 48,115,256 | 3,419,909 |
| Bengali | 1,563,137 | 707,473 |
| Telugu | 1,019,430 | 1,255,086 |
| Tamil | 881,429 | 1,407,646 |
| Nepali | 705,503 | 314,408 |
| Sanskrit | 553,103 | 448,784 |
| Marathi | 519,506 | 498,475 |
| Punjabi | 503,330 | 247,835 |
| Malayalam | 493,234 | 1,325,212 |
| Gujarati, Konkani, Odia, Assamese, Kannada (each) | $< 500{,}000$ | $< 500{,}000$ |

Table 5.1: Corpus statistics

| **Language** | as | bn | gu | hi | ml | mr |
|---|---|---|---|---|---|---|
| **Perplexity** | 455 | 354 | 183 | 518 | 1689 | 522 |

### 5.2.2 Word2vec and Fasttext

The gensim library[58] was used to create Word2Vec embeddings of dimensions50, 100, 200, 300 for both skip-gram and CBOW architectures. Words with a frequency less than 2 in the entire corpus were treated as unknown (out-of-vocabulary) words. For other parameters, default settings of gensim were used.

Similar settings were used for Fasttext models. We note that there are pre-existing models for all the languages that we have presented except Konkani and Punjabi. However, the corpora on which we train the Fasttext models are significantly larger.

### 5.2.3 ELMo

We train ELMo embeddings [49] of 512 dimensions. These vectors are learned functions of the internal states of a deep bidirectional language model (biLM).The training time for each language corpus was approximately 1 day on a 12 GB Nvidia GeForce GTX TitanXGPU. The batch size is reduced to 64, and the embedding model was trained on a single GPU. The number of train-ing tokens was set to tokens multiplied by 5. We choose this parameter based on the assumption that each sentence contains an average of 4 tokens. There are no pre-trained word embeddings for any of the 14 languages available on the official repository.

Table 5.2 shows the perplexity scores of the ELMo models trained by us.

### 5.2.4 BERT

We train BERT (Bidirectional Encoder Representations from Transformers) [23] embeddings of 300 dimensions. Since BERT can be used to train a single multilingual

| **Language** | kn | ko | ne | or | pa | sa | ta | te |
|---|---|---|---|---|---|---|---|---|
| **Perplexity** | 155368 | 325 | 253 | 975 | 145 | 399 | 781 | 82 |

Table 5.2: ELMo prerplexity scores

model, we combined and shuffled corpora of all languages into a single corpus and used this as the pre-training data. We use sentence piece embeddings [35] that we trained on the corpus with a vocabulary sizeof 25000. Pre-training this model was completed in less than 1 day using 3 * 12 GB Tesla K80 GPUs. The official repository for BERT provides a multilingual model of 102 languages, which includes all but 4 (Oriya, Assamese, San-skrit, Konkani) of the 14 languages. We provide a single multilingual BERT model for all the 14 languages, including these 4 languages.

## 5.3 Results

Using these embeddings supplied as an input to Flair[2], we trained an NER model and the results for the best models are listed in 5.3.

| Language | NER corpus | | | FastText (300 dim) | | Skip-gram (300 dim) | |
|---|---|---|---|---|---|---|---|
| | Train | Test | Dev | Accuracy | F1-score | Accuracy | F1-score |
| Hindi | 6735 | 962 | 1925 | 91.6 | **95.62** | 90.52 | 95.03 |
| Tamil | 4649 | 664 | 1329 | 90.63 | **95.09** | 88.71 | 94.02 |
| Malayalam | 2111 | 302 | 604 | 79.51 | **88.58** | 65.14 | 78.89 |
| Bengali | 615 | 88 | 176 | 79.26 | **88.43** | 72.73 | 84.21 |
| Marathi | 3590 | 315 | 1681 | 40.12 | **57.26** | 16.73 | 28.66 |

Table 5.3: NER results. Rows have been listed in descending order of Accuracy metric for FastText (300 dimensions)

The performance of other embedding models (FastText, CBOW, Skip-gram of dimensions 50, 100, 200, 300) were lower than the ones listed in table 5.3. The results vaguely follow the distribution of the NER training corpus size, with the exception of Marathi. Note here that the model fails to produce as good results as others for Marathi, despite having a larger corpus than Malayalam for pre-training as well as a larger training corpus for NER.

# Chapter 6

# Morphological tasks for Indian languages

A word usually contains phonological, grammatical and semantic information in it. For example, consider the word cat. The word contains several, rich morphological information: its pronounced as /kæt/, it is singular, and is an animal.

The task involved in morphological analysis is to identify morphemes, the minimal meaningful units. Words can have variations due to inflection, derivation, contraction etc.

E.g. : nouns → number, case, honorifics

E.g. : verbs → gender, number, person, tense

## 6.1 Utility

Stemming is frequently used in information retrieval systems. It is used an approximation to MA used for IR. It allows the systems to store records even though it might be present in documents or links in multiple forms.

It also helps in language modelling. Several NMT based systems use stems as the input to the system instead of the inflected words. They also feed the rich morphological features as input along with the stem so that the context is not lost. Also, morphology tasks can go the other way too: given a root word, the inflected forms can be predicted. This is useful for generating the vocabulary of a language or generating options for filling gaps in sentences.

## 6.2 Background and our work

Most of the work on morphological analysis in Indian languages have been done using rule based systems [62] [56], which require linguistic experts to model them. Even with the idea of cross training language models [57], people still relied on manually annotated rules for using the knowledge from a high resource language in a low resource setting.

With word2vec [41][42], it became computationally cheaper for people to generate embeddings, but neural networks were still not adopted for morphological tasks. Most of the work still progressed using statistical means[10][68], which meant requiring linguistic experts to solve the problems.

Most recently, seq2seq models have come up which can be trained to output the lemma of the word, given an inflected form of the word as well as morphological tags as input. With the presence of sufficient annotated data, the models were able to perform nearly as well as experts, and in some cases, even surpass them.[22][53]

Our work focuses in the same direction, utilising the power of neural networks and deep learning to our advantage in the context of Indian languages for lemmatization. Since we work in the context of Indian languages, we work around the constraint of the low resources and sparsity of annotated data. Due to this, we chose our starting point to be embeddings, which are generated in a completely unsupervised manner and thus suited to our task.

## 6.3 Data

For the embeddings, we use the embeddings trained on the corpus mentioned in the previous chapter. For evaluating the results of embeddings, we use a mixture of different datasests.

For completely unsupervised methods, the ground truths are created manually or via a heuristic. For semi supervised and supervised methods, the UD treebank is used, which provides a list of words, lemmas and a corresponding set of rich morphological features for that word. The size of the UD treebank varies greatly from language to language. For Hindi, it is around 357K sentences split into train, test and dev sets. Data is also taken from the SIGMORPHON 2019 shared task which provides annotated data for 2 Indian languages: Bengali and Kannada. However, the data provided by SIGMORPHON contains only 100 tokens.

# Chapter 7

# Morphology using embeddings

Since our work focuses on generating a lemmatizer in the low resource setting, the methods that would be the most effective would be unsupervised or semi-supervised methods with the least amount of annotated training data possible. Since embeddings are supposed to capture the semantic as well as syntactic meaning of the word, it was proposed that we could use embeddings to get the lemmas of the word. This chapter explains the numerour methods that were tried out to see the viability of using embeddings for lemmatization.

## 7.1 Fasttext

Given that fasttext was the best performing model among all the non contextual embeddings, we used it as the starting point for the experimentation described in this section. Additionally, Fasttext encodes subword level information, which might be beneficial in the task. We use the models which gave the best performance in the PoS tagging and NER detection task, namely the models which embed the words in 300 dimensions.

To start off, we extracted the top 10 closest words corresponding to each word present in the embedding space. This served as ou starting point because we thought that the embeddings could provided useful information. The relation between embedding vectors of related words was initiallly shown by Mikolov et al. 2013 [41] when he showed that the closest word to $v(king) - v(man) + v(woman)$ was the vector embedding of $queen(v()$ is a function that maps the word to its embedding in the embedding space). Since different forms of a verb are supposed to be related to each other, the lemma should be present be present near the inflected word in the embedding space.

Table 7.1 lists the sample output for 4 random words chosen from the corpus, as well as the predicted lemma from the wordnet. As can be seen, Fasttext does put a lot of emphasis on words which have the same substrings, that is, on subword level information. All the words that are shown have a common substring with the query word. Of the 4 sample outputs given in the table, 3 of them contain the lemma, which show that our initial hunch was right.

| Word | Root word (Wordnet) | Closest word (embeddings) |
|---|---|---|
| ताजा | ताज | ताज़ा, ताजाताजा, ताजातरीन, ताजातर, ताजा, ताजातम, ताजी, ताज़ागी, ताजे, नताजा, ताजाजल |
| साँसें | साँसत | साँसे, उसाँसें, सांसें, सांसे, साँसेंइसलिए, उसाँसे, खाँसें, साँसैं, उसांसें, फाँसें, साँसतें |
| दाँत | दाग | दांत,दाँतों,दाँतन,दाँतू,दाँतो,दाँती,दांतों,दाँतून,दाँत कटी,दाँत:,दाँतउगने |
| मसूढ़ों | मसूढ़ा | सूढ़ो,मसूढों.सूढ़ों,मसूढ़े,मसूढ़ा,मसूढो,नसूढ़ों,मसूढ़े, मसूडों,,मसूढ़ें,मसूढे |

Table 7.1: Sample output from Fasttext model

However, 2 things stand out. Most of the words included in the output, even though they share substrings with the query word, are more or less of the same inflection. This suggests that for some words, especially for verbs, this model might pose a problem due to the sheer number of inflections possible in verbs. Another thing to note is that the predicted root word according to the wordnet is also off. The predictions from wordnet are evaluated using edit distance from the query word and the word with the least edit distance is selected as the root word. Issues regarding the wordnet will be discussed in this chapter in the later sections.

To get an idea of the distance between words, we evaluated the Levenshtein distance between the query and the closest embedding as well as the average distance between the query and the proposed root word according to wordnet. The average distance between the query word and the best embedding is 0.77, which means that on average, we can expect that only one letter will be changed. The average edit distance between the query word and the proposed root word, however, is 0.56. On an initial look, this means that the words we get from Fasttext are going to be inflected. Still, since we take multiple outputs from Fasttext and the difference in edit distance is not too large, we evaluate it on multiple languages.

## 7.2 Evaluation on multiple languages

Wordnet provides a list of words and their glosses, all of which are in their root form. We combine all of this data and create a list of possible lemmas for that language. Table 7.2 shows the number of lemmas that we could extract out from each language:

| **Wordnet size** | as | bn | gu | hi | kn | ko |
|---|---|---|---|---|---|---|
| **Language** | 9317 | 0 | 22040 | 148958 | 16627 | 13165 |

| **Wordnet size** | ma | ne | or | pa | sa | ta | te |
|---|---|---|---|---|---|---|---|
| **Language** | 9087 | 8465 | 20396 | 18986 | 8778 | 10241 | 11195 |

Table 7.2: Dataset size of wordnet for different languages

The distribution of number of lemmas follows almost the same distribution as the dataset size that we used for the creation of embeddings. However, if we take the ratio of the corresponding two numbers for each language, we see that for each lemma we have 10-15 words in the embeddings. This means, that on average, the embeddings contain anywhere from 10-15 inflections of each word. For verbs, this might be a good number, but it might be a suitable range for nouns, which have fewer inflections.

For the evaluation, we carried out the same steps that we carried out for Hindi. We took the top 10 closest words in the embedding and checked if they matched the proposed root word that we got from wordnet. The proposed root word was selected on the basis of a simple heuristic of least edit distance.

Surprisingly, Hindi is one of the worst performers according to this criterion. The best performing language is Oriya, but that also barely crosses 10. Southern languages, in particular, seem to perform better on average than their northern counterparts. The dataset size does not seem to have any correlation with the accuracy of the model.

Based on this, there are two possible conclusions that can be made. The first one is that the embeddings might not of good quality. This would result in non-related words being close to each other which would mess up the accuracy. The other one is that the proposed root word from wordnet is not correct. So even though the model might give correct predictions, the standard that is being checked against might be wrong. Or it also possible that it might be a mix of the two.

In this section, we focus on the first part. To evaluate the quality of our embeddings, we test our model alongside the pretrained models available online. The difference between the models are in the corpus that they are trained on and the inputs they take. Our models transliterate every language to devanagari before feeding it to the embedding system. On the other hand, the pre-trained models available online use the native script of the language as input. For comparing the models, we use the same methodology as before, ie, we find the closest words and compare them with the root word that we get from the wordnet based on heuristics.

The results are shown in Table 7.3. The table has comparisons with pretrained models available online and two versions of our models. The columns 'Augmented' and 'Unaugmented' refer to the data which is added to the wordnet. Adding more lemma data should increase the accuracy of prediction. Likewise, the second model that we compared on is the same fasttext model, but with more lemmas added to the corpus and hence, to the embeddings.

As visible, the augmented columns, in most cases, give better result than in the unaugmented columns. The increase in accuracy can be justified because of the increased lemma data, which would mean better lemma prediction. The decrease in

| | External | | Self | | Self v2 | |
|---|---|---|---|---|---|---|
| | Unaugmented | Augmented | Unaugmented | Augmented | Unaugmented | Augmented |
| as | 0 | 0 | 1 | 2 | 2 | 3 |
| bn | 6 | 1 | 0 | 0 | 11 | 9 |
| gu | 1 | 1 | 6 | 8 | 8 | 13 |
| hi | 0 | 0 | 3 | 3 | 3 | 3 |
| kn | 1 | 1 | 8 | 14 | 8 | 14 |
| ko | 0 | 0 | 7 | 1 | 10 | 1 |
| ml | 0 | 0 | 0 | 0 | 3 | 5 |
| mr | 0 | 0 | 0 | 0 | 8 | 7 |
| ne | 1 | 0 | 3 | 7 | 4 | 7 |
| or | 1 | 1 | 12 | 6 | 14 | 6 |
| pa | 2 | 1 | 2 | 1 | 2 | 1 |
| sa | 1 | 1 | 2 | 2 | 4 | 2 |
| ta | 1 | 1 | 6 | 3 | 7 | 6 |
| te | 1 | 1 | 2 | 2 | 2 | 2 |

Table 7.3: Comparison of our models with pretrained models online. Augmented and augmented refer to data added to wordnet for better root prediction. The second self model refers to the fasttext model after the lemma data from the wordnet was added to the embeddings

accuracy, however, is harder to explain. This might be because a word did not have the correct lemma in the corpus before and the addition of the more data pointed it to the correct lemma. The fasttext model is the one that gives the wrong output in this case. It was somehow pointing to the incorrect lemma in the unaugmented case and still points there, because nothing has changed in the model, only in the predicted gold standard.

Another thing to note is that in the two models that we use, the second model always performs better. There are no models with zero hits anymore. Also, several languages like Kannada, Bengali, Konkani move into the 2 digit accuracy scores. This just confirms the fact that adding more data to embedding models, improves the quality of embeddings. Overall, the accuracy is best in the last column of the table, which uses additional data in both root prediction and for generating embeddings.

However, one thing to be noted is that adding data is not a feasible choice for many Indian languages. In this case, we found out an additional data source which also included lemma data, by chance. We are still working in the realm of low resource languages, which means we have to make do with lesser resources than what is the standard in English.

One conclusion that can be drawn from Table 7.3 is that our models produce embeddings that are at par, if not better, compared to the pretrained models online. This rules out the option that the error in our models was due to bad embedding quality, which means that the error lies in the predicted roots. Therefore the next phase of our research focused on improving the gold standard against which we compare

our results. Before that, we present a more exhaustive analysis of the errors

## 7.3 Analysis of results on Hindi

Table 7.4 shows the results for 6 random words picked out from the corpus along with the remarks. Multiple conclusions can be drawn from the results shown:

- The root word are in most of the cases, the subword of the root word itself. The root word in those cases can already be present in the corpus or not. In the former case, the embedding of the root word is too distant from the embedding of the query word that it does not appear in the top 10 list. In the latter, the subwords need to be induced into the embedding space so that they can be considered as a possible candidate as the lemma. Since fasttext focuses on subword information, the subwords should feature in the top 10 list
- The lemma identified via the wordnet is incorrect. This is an independent problem of the previous problems mentioned above. This means that the gold standard which we are evaluating on needs to be improved so that the accuracy is a better representative of the true score.
- The embeddings do not feature related words in the top 10 list. This means that the quality of the embeddings is suspect. Although this can be mitigated, as stated earlier, by adding more data, we need to be mindful that we are doing work in a low resource setting.
- The word itself is the root. If the top 10 list were to feature the query word itself, it would always rank at the top, because the distance of any word to itself is 0.

The errors in the outputs always show a mix of these results. The most numerous ones are the ones where the predicted root according to wordnet is incorrect and where the lemma is the word itself. The number of such cases is almost 4-5 times more numerous than other cases.

To fix this issue, we first analyse how to improve the gold standard.

## 7.4 Improving gold standard

Initially, the lemmas were predicted based on a simple Levenshtein distance metric. The words with the least edit distance were chosen as the lemmas. This section lists the methods used to improve the gold standard accuracy.

First, we used heuristics like filtering based on length, and string similarity. The first one stems from the common occurrence that the lemma is usually the shortest word in the list of all inflections of a given word. Based on this, we filter all the words which are of a longer length than the query word. The second one is based on the fact that, excluding irregular inflections, the lemma remain largely unchanged. So we filter the model based on a minimum similarity with the lemma.

| Word | Root word (Wordnet) | Closest word (embeddings) | Reason |
|---|---|---|---|
| उंदीरकानी | उंदीर | उंदीर, फूलदानी, उंदारानी, मृडानी, अडानी, कफानी, उंदीरमार, खुमानी, उंदी, चिखलानी, अंडमानी | Word found |
| औषधी | औषध | औषधीय, औषधीगुण, वनऔषधी, औषधीकोश, औषधीत, औषधिय, औषध,, ओषधी, आयुर्वेदीक, औषधिगुण, औषधि | Root word is subword and in corpus, but not in the list |
| काजीवाल | कलजीभा | देहरीवाल, कुतबीवाल, छुरीवाल, कुतीवाल, लेसड़ीवाल, ईलवाल, खीवाल, भड़वाल, गणीवाल, छीनीवाल, चूहड़वाल | Lemma is incorrect as well as closest words are different (even semantically) |
| स्वास्थ्य | स्वार्थी | पहल:स्वास्थ्य, स्वास्थ्यस्वास्थ्य, केस्वास्थ्य, कोस्वास्थ्य, औरस्वास्थ्य, उनकेस्वास्थ्य, कुस्वास्थ्य, थास्वास्थ्य, स्वास्थ्याए, स्वास्थ्कर, वास्थ्य | Root word misidentified, Word itself is root |
| बोलूं | बोल | बोलूँ, बोलूंगी, बोलूंकि, बोलूंगीकहोगे, बोलूगी, बोलूंगा, बोलूगीं, बोलूगां, बोलूगा, बोलूग, बोलूँमुँह | Root word is subword and in corpus, but not in the list |
| खबरों | खबर | ◌खबरों, ख़बरों, अखबरों, अख़बरों, खबरों, समाचारों, हैख़बरों, थेख़बरों, खबरो, साहबखबरों, खबरअखबार | Root word is subword, not in the closest embedding, although present in corpus |

Table 7.4: Analysing the output of the model for 6 random words in Hindi

These heuristics did not improve the results by much. The reason for this can be that the Levenshtein edit distance does not take into account the position where the edits occur. For example, in normal words most of the changes take place either at the end or the start of the word. The word is not modified in the middle usually. By giving more weightage to the difference in strings towards the end or the start, we could get better lemmas.

To test this, we used the Jaro-Winkler edit distance. The Jaro–Winkler distance uses a prefix scale $p$ which gives more favourable ratings to strings that match from the beginning for a set prefix length $l$. The lower the Jaro–Winkler distance for two strings is, the more similar the strings are. The score is normalized such that 0 means an exact match and 1 means there is no similarity. We set the $p$ value to be the length of the word and find the most similar words. The metric adds the term $m/s_i$ for each position $i$ in the string, where $m$ is the number of matching characters in the string till that point. Given this formula, we modified it to make it focus more on edits near the end by adding extra penalty for incorrectness.

Using the Jaro-Winkler distance did not give much improvement too. As a result, we manually curated a list of lemmas and also used rule based heuristics to get lemmas.

The rule based heuristic was based on the observation that plural words usually end in specific matras. As a result, we filtered out words based on specific matras and then manually selected the correct lemmas and their variants from the filtered list to get a list of 150 words to train on.

## 7.5 ElMo

After creating the gold standard, we evaluated the model again using Fasttext as the model. This gave us marginally better results, indicating that the embeddings needed to be improved. Additionally, even for the words that were predicted correctly, the lemmas were often present at random places in the top 10 positions: for example, "मुनाफ़े"'s root was at the 6th position, "हल्के"'s root was at the 10th position, and "वेदों"'s root was at the 2nd position.

To hopefully get better results, we shifted to ElMo. There were multiple reasons for this shift: ELMo uses BiLSTM networks, which means it encodes more information than word2vec or Fasttext. This also means that the embeddings produced by ElMo are context dependent and thus the same word appearing in different contexts would have different embeddings. This was not the case in word2vec or Fasttext which have a static embedding for each word. Additionally, ElMo uses character convolutions, which means it has the ability to process Out of Vocabulary(OOV) words. This is useful in our scenario where the resources are limited and there is no guarantee that the lemma is present in the corpus.

On inspection, the difference between ELMo and the previous models was clear in the list of closest words for any word in the embedding space. For the commonly used and frequent words, the list was accurate. For other words, there were one or two words that were not an inflected form of the word. Also, the embeddings were more tightly clustered in the ELMo embeddings than in Fasttext. All this resulted in the lemmas appearing more frequently in the top 10 list and if they were already present in the list, they were listed closer to the query words. The accuracy, however, at this point still remained below 25%.

To analyse the issues more deeply, we focused on the reverse task, ie, the task of finding inflections given a word.

## 7.6 Reverse task

The reverse task of lemmatisation focuses on finding the possible inflected forms of the verb, given the lemma. For this we used the manually curated list of lemmas as our input. We follow the same procedure as before: finding the top 10 closest words for each word. The results are shown in Table 7.5.

From the tables, it is clear that the model performs well on the reverse task, as expected. We can infer from the table that given a word, words nearby in the embedding space will be related to that word, but it is not necessary that the lemma will be in that list. Also, since the model is trained on Wikipedia dumps and other common sources, it is expected that the model outputs the more common inflections of the words first. Likewise, for words that are not so frequently used, the quality of words surrounding it in the embedding space also varies.

Another thing to note is that, based on the table, nouns perform better on the lemmatisation task as compared to the reverse task. Verbs, on the other hand, are more suited for the reverse task using embeddings because of the different variations

| Corpus word | Top Similar words |
|---|---|
| सरकार | *सरकाऱ*, कीसरकार, सरकार्र, *सरकारा*, **सरकारे**, *सरककार*, *सरकारर*, **सरकारौ**, सरकारने |
| समस्या | *समस्य्या*, **समस्याऐ**, **समस्यऐ**, *समस्सया*, *समस्याा*, समसंख्या, विसंगति, समस्था, दिक्कत |
| गेंद | *गैँद*, गेंदई, गेंदुक, **गेंदें**, *गैंद*, *गेन्द*, **गेंदः**, गेंदब, गेद |
| बेटा | **बेटी**, **बेट्ाे**, भतीजा, लाबेटा, बच्चा, बहनोई, भैया, **बेटो**, जीजा |
| खुलासा | *खुलासाः*, **खुलासत**, भंडाफोड़, **खुलास**, पर्दाफाश, दावा, ऐलान, *खुलासा*, पर्दाफ़ाश |
| देंगीं | ***देंगा***, **देंगी**, **दूगीं**, **दूंगीं**, ***दूगी***, **देगीं**, **देंगें**, **देंगीं**, **देवेंगे**, **दूंगी** |
| गेंद | *गैँद*, गेंदई, गेंदुक, **गेंदें**, *गैंद*, *गेन्द*, **गेंदः**, गेंदब, गेद |
| गाड़ने | **गाड़ने**, **गाड़ना**, झाड़ने, *गाडने*, गुड़गुड़ाने, फडाने, फाँसने, कड़कड़ाने, माड़ने, जखने |
| बैठी | **बैठा**, **बैठाई**, **बैठी**, **बैठीं**, बेठी, *बैठेती*, **बैठती**, **बैठ्ते**, **बैठंे**, **बैठु** |

Table 7.5: Output of the reverse task for nouns and verbs. Words in grey are words which represent spelling variations and are in the courpus. Words in black represent the inflected forms

present in verbs. Nouns are much more limited in the number of inflections they can go through, so the lemmas occur more frequently in the top 10 list.

Based on this information, we experimented on a few methods to improve the accuracy for the lemmatisation task. We experimented with taking the average of top 10 closed words, using clustering algorithms etc but they did not give good results. As an illustration, Fig 7-1 shows the projection of the embeddings using t-SNE and the set of closest words for a given word.

## 7.7 Using geometric methods

When Word2Vec was introduced by Mikolov et al. 2013, the model was praised for its performance on semantic as well as word analogy tasks. The authors gave the example of vector arithmetic on the embeddings to show that their model learnt real world dependencies between the meanings of the data: the vector of *king* added to the difference of the vector of *woman* and the vector of *man* gave a vector which was closest to the embedding for *queen*. This approach lies at the foundation of the approach mentioned in this section. The methodology is as follows: given the embedding of an inflected word and its root, we calculate the difference and store it. For any new word, we add this difference to the embedding of the query word and then try to find the word closest to that embedding. The following equation sums up the basic idea:

$$\text{lemma} = \text{closest}(\boldsymbol{v}(query) - \boldsymbol{v}(B) + \boldsymbol{v}(A))$$

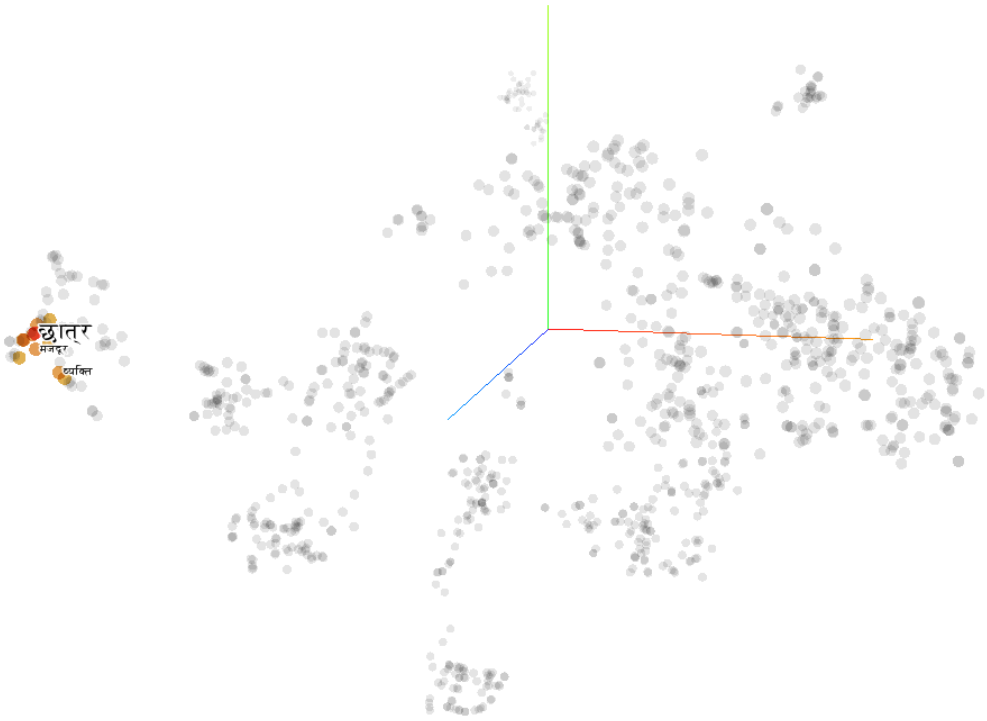



Figure 7-1: Projection of the ElMo embeddings generated by us, using t-SNE. The highlighted words represent the closest words for the word in red. credits for visualisation: Google Tensorboard

where $A$ is a root word and $B$ is an inflection of $A$ that we know beforehand.

The results vary depending upon the known word-lemma pair chosen. There was a difference in 15% between the best and worst performances that we encountered so far. Surprisingly, using a known noun word-lemma pair for a noun query does not make much of a difference as compared to using a known verb word-lemma pair.

Using this method yielded a significant increase in the accuracy to 50%. The model performs the best for nouns, followed by adjectives and then verbs. Adverbs are not counted because their frequency in the corpus is too low. The accuracies for these tags are 70.5%, 70% and 11.11% respectively.

Since the model works fairly well for verbs, we focused on using the embeddings in Hindi, which is a high resource language, for Marathi, a relatively lower resource language.

## 7.8 Aligning embedding spaces of different languages

We attempted to use the embeddings from a high resource language to the benefit of lower resource languages. The way we did this was by realigning the 2 vector spaces so that they are superimposed on each other. This way, the difference vectors used in Hindi could be used to get the lemma of inflected words in Marathi.

| | | Marathi space | Hindi space |
|---|---|---|---|
| मंदिरात | मंदिर | मंदिराच्या मंदिर राजमंदिर \| मंदिराच्या मंदिर राजमंदिर \| मंदिराच्या मंदिर राजमंदिर | शुककीट पैंटानल जुञ्ज \| शुककीट फल_वाला पैंजना \| दुरुस्तक कापचुक ठीक-ठाक |
| राजाची | राजा | जनावराची प्राणाची राजावर्तच्या \| राजावर्तच्या तराजूची जनावराची \| पंजाची राजावर्तच्या तराजूची | कवाई जूरी बेंत \| कुहुकना कालदमनी दुढ्डाचारी \| हयारोही सूबेदारी जरतारी |
| वाद्ये | वाद्य | वाद्य प्यादे पत्ते \| प्यादे वाद्य पत्ते \| वाद्य प्यादे पत्ते वास्तू | पैक्दारा प्री-क्वाटर अभोक्ता \| पायखाना गमतखाना पेंहाना \| भारोत्तोलक सपारकन पादवल्मीक |
| राणीला | राणी | राशीवाला दादला राणी \| राशीवाला रागेला दादला \| राशीवाला रागेला दादला | जूरी पतई मेहरी \| शुककीट ढ़ेचा मतला \| सद्योजात दुस्सह पतई |
| कागदपत्रं | कागदपत्र | कागदपत्र पाचकग्रंथी \| कागदपत्र संकल्पपूर्वक \| कागदपत्र पाचक-ग्रंथी | स्थानक काटर-वासी \| पनकपड़ा तबला-वादक \| नवागन्तुक पादवल्मीक |

Table 7.6: Aligning the Hindi and Marathi embedding spaces(Procrustes' alignment) using sentence vectors from a parallel corpus of 150 words

### Procustes' distance

Aligning the datasets was initially done using the Procrustes' distance as a metric. A small set of parallel corpora was created using Google Translate and then used as reference points for evaluating the distance between the 2 emebdding spaces. Procrustes' distance is usually used in imaging to align multiple images, which is where the inspiration was drawn from.

$$||WX - Y||_2$$

This is the loss function that was minimised for aligning the two vector spaces. $W$ is the translation and rotation matrix. $X$ and $Y$ are the two embedding spaces to be aligned. Both of them are converted to a 0 mean and unit variance distribution on the pivot points before the loss minimisation starts.

Tables 7.6 and 7.7 show the results for the alignment using the Procustses' metric.

### MUSE

MUSE[20] was developed to enable the creation of multilingual embeddings by aligning multiple embedding spaces. They evaluate the alignment using the Procustes' metric with a little modification. They identify different clusters within the individual datasets and realign the points within the clusters so that the spread in clusters becomes uniform.

MUSE offers 2 variants: supervised and unsupervised models. We use both of them to align the models in the embedding spaces. The unsupervised methodsFor the anchor points in the supervised setting, we use a parallel corpora available in CFILT: for Marathi

| | | Marathi space | Hindi space |
|---|---|---|---|
| मंदिरात | मंदिर | मंदिर मंदिराच्या राजमंदिर \| मंदिर राजमंदिर मंदीर \| मंदिर मंदीर मंदिराच्या | मंदिर मंदिर मानमंदिर \| मंदिर मंदिर मन्दिर \| मंदिर मानमंदिर मन्दिर |
| राजाची | राजा | तराजूची पंजाची खजानची \| राजाबाबतचा तराजूची पंजाची \| पंजाची जनावराची तराजूची | राजाका राज्ञी राज्ञैली \| राजाका प्रजाका पराजिका \| पजहस्सी पन्थकी जपनीय |
| वाद्ये | वाद्य | वाद्य ओळ प्यादे \| वाद्य तंतुवाद्य वादळ \| वाद्य विवाद्य धनदायी | धात्र खाद्योज कंदूरी \| पनकपड़ा पायखाना गमला \| दिव्यास्त्र इंद्रजालिक लौकिक |
| राणीला | राणी | राणी रातराणी पराणी \| राणी राणीवसा रातराणी \| रातराणी राणी राणीवसा | राज्ञैली राज्ञी जुर्जी \| पुत्रवाला कबीला लौआ \| पजहस्सी राज्ञी बैदाई |
| कागदपत्रं | कागदपत्र | कागदपत्री कागदपत्र अंदाजपत्रक \| कागदपत्र कागदपत्री काचकागद \| कागदपत्री कागदपत्र अधिपादप | खाद्योज किगूटी स्फाटक \| पनकपड़ा पायखाना फल_वाला \| हास्योत्पादक निश्चयात्मक विधेयात्मक |

Table 7.7: Aligning the Hindi and Marathi embedding spaces(Procrustes' alignment) using sentence vectors from a parallel corpus of 47000 sentences

## 7.9 Drawbacks

The issues faced by the model can be highlighted by examining areas where the model underperforms: for data which included repetition, the best score was 56%. Among these, the accuracy on query words which are the lemma themselves is 73.03% and the accuracy on test words which are not already in root form is 11%. This percentage drops to 7% if duplicates are removed from the dataset. The disparity between the two subclasses show that the difference vectors used are not good enough for all test words or there is an issue with the embeddings themselves.

The accuracy of the model based on PoS tags of the words are highest for nouns (100%) and the least for verbs(11%). Adjectives and adverbs perform fairly well at 70% and 75% accuracies respectively.

The problems with this method can be categorised into 4 different categories:

- Word embedding quality: even though Hindi is one of the Indian languages with the most resources, the quality of embeddings is still not as good as English's, which is trained on a billion tokens. The issues with the embeddings can be seen in the list of closest words in all of the Tables spread out across this chapter.
- Data sparsity: This is related to the previous problem and is a direct cause of it. If the data is sparse, the model cannot learn the context of each properly and map it to a point in the embedding space which captures its meaning properly
- False positives: Instead of giving the correct root, the model predicts another word as the root. This word can either be morphologically or semantically close to the query word.
- False negatives: If the lemma is not in the corpus, the model cannot predict the correct lemma at any point of time. Even though this is mitigated a bit by adding substrings to the corpus, it does not alleviate the problem completely.

The issue of data sparsity and low quality of word embedding is something that cannot be changed easily without adding more data to the corpus. Even then, there is no guarantee that at which point the model will start giving good results.

To solve the problem of false negatives, we also tried to get the word corresponding to the final embedding via a brute force search. Since we are now working at the character level, whose numbers are fixed, we can join any string of characters to form a word closest to the embedding. This method worked out well for some words but failed in others.

# Chapter 8

# Morphology using seq2seq models

The best result from using the embeddings stand at 53%, which is still behind the results from a rule based lemmatiser[10]. To get better results, we move to more powerful techniques. Although to make them work within the confines of our restraint of low resources, we need to make them lightweight or augment the training data so that the training remains effective.

Seq2seq models have become popular for lemmatization and other morphological tasks, along with neural machine translation[46]. However, seq2seq is a predominantly supervised learning method and requires substantial amount of data to learn effective patterns relevant to the task, which is a constraint for low-resource languages.

Recent work on data augmentation in NLP by Anastasopoulos and Neubig, 2019 [3] showed that augmenting the data was possible and effective in a low-resource setting. The paper was intended for a morphological re-inflection task, which is related to the lemmatization task. It also helps address the multiple issues present in the embeddings-based approach: since we are augmenting data, the problem of data scarcity is mitigated. False negatives and false positive are also handled because the model is now learning about the correct output based on annotated data.

## 8.1 Data preparation

We use the data from UD Treebank[45] and from the SIGMORPHON 2019 shared task[39] to create two different datasets of Indian languages from the Indo-Aryan and Dravidian family of languages. We collect datasets for five languages from the UD treebank: Hindi, Marathi, Sanskrit, and Tamil. From the SIGMORPHON 2019 shared task, we collect language data from the cross-lingual morphological inflection task for Bengali, Hindi, Kannada, Sanskrit, Telugu, and Urdu. Out of these, Telugu is the only one that does not have a high resource dataset.

For each language we create two datasets - one in a high resource setting (with around 5,000+ lines), and one in a low resource setting (with only 100 lines). The data for the low resource setting is then augmented using 8.1.1.

| Language | Total | High | Low |
|---|---|---|---|
| Hindi | 255894 | 15000 | 100 |
| Marathi | 1895 | 1895 | 100 |
| Sanskrit | 1301 | 1301 | 100 |
| Tamil | 5483 | 5483 | 100 |
| Urdu | 10089 | 15000 | 100 |

(a) UD treebank

| Language | Total | High | Low |
|---|---|---|---|
| Bengali | 3394 | 3394 | 100 |
| Hindi | 10000 | 10000 | 100 |
| Kannada | 3506 | 3506 | 100 |
| Sanskrit | 10000 | 10000 | 100 |
| Telugu | 61 | - | 61 |
| Urdu | 10000 | 10000 | 100 |

(b) SIGMORPHON 2019 task 1

Table 8.1: Data available for different languages. The tables show the number of inflected-word lemma pair for each language in each dataset. The *Total* column shows the original number of such pairs and the *High* and *Low* columns show the curated training dataset size in a high and low resource setting respectively.

### 8.1.1 Data augmentation

We use the alignment method to generate additional artificial data for augmenting our original dataset[21]. The reason for choosing this method is that Indian languages show rich and diverse morphology and that other ideas like a longest common substring match might not work properly in these environments.

In the alignment method, any sequence of at least two characters in the inflected word that are aligned to the same characters in the lemma is assumed to be a part of the stem. For each of these sequences, we randomly sample alphabets from that language and substitute it with the identified sequence.

For each language, the training data is augmented so that the total training set size is equal to 10,000, including the original training data.

## 8.2 Vanilla seq2seq model

Let $\boldsymbol{X} = x_1...x_N$ be a character of the inflected word, $\boldsymbol{T} = t_1...t_M$ the set of morphological tags associated with it and let $\boldsymbol{Y} = y_1...y_K$ be the lemma target character sequence. The goal is to model $P(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{T})$.

`<START>` and `<END>` tokens are added to the training input and outputs. The inputs are tokenised at the character level and are one-hot encoded. These vectors are then embedded in a latent space via a projection matrix before being fed to the encoder. The encoder for the input is single layer LSTM. The encoder's output is an intermediate representation $\boldsymbol{h}_1^x \dots \boldsymbol{h}_N^x$ and $\boldsymbol{h}_1^t \dots \boldsymbol{h}_M^t$ for the lemma and tag respectively, which are then fed to the decoder. The decoder is initialised with the final states of the encoder and the first input to the decoder is the `<START>` token.

### Results

The simple LSTM models achieve 26% and 76% accuracy for Bengali (transliterated to Hindi) and Marathi respectively. After including a callback for reducing the learning

rate to improve learning, the accuracy for Bengali increases to 29% while the accuracy for Marathi remains almost the same at 75%. On including masking in the LSTM layer, the accuracy further increases to 34% and 78% respectively. The performance of the models deteriorated to 22% and 54% for Bengali and Marathi respectively when GRUs[17] were used in place of LSTMs.

We also train models using an attention model[7], which is described in more detail in Section 8.3. Using the attention mechanism and masking increases the accuracy for Bengali further by 3 percentage points to 37%. The test accuracy for Marathi still remains at 76%. On using the Bengali dataset in its original script (Bangla), the accuracy further shot up to **57%**.

A note of interest is that the accuracy for Marathi has remained constant throughout this evaluation. This might be because of the difference in amounts of data present for both languages. In the case of Bengali, smaller fluctuations in training methodology can improve or deteriorate a model's performance easily. This is mitigated in the high resource setting because of the sheer amount of data which makes it better positioned to handle fluctuations in training methodology.

## 8.3 Attention based models

We use an input-feeding attention network in the model[37]. Specifically, the output at the previous time-step $k$ ($\boldsymbol{s'_{k-1}}^{x}$) is used to attend over tags and then used to create a tag-attentive state $\boldsymbol{s}'_k$ . This is then attended over the input embeddings and fed as the context vector $\boldsymbol{c}^x_k$ to the encoder at the next timestep.

$$\boldsymbol{s}_k = \boldsymbol{s}'_{k-1} + \boldsymbol{c}^t_k \qquad \boldsymbol{c}^x_k = \left[\sum_n \alpha^x_{kn} \boldsymbol{h}^x_n\right]$$

$$\boldsymbol{s}_k = \text{dec}(\boldsymbol{s}'_{k-1}, \boldsymbol{c}^x_k, y_{k-1}) \quad \boldsymbol{c}^t_k = \left[\sum_m \alpha^t_{km} \boldsymbol{h}^t_m\right]$$

$$\alpha^x_{kn} = \text{softmax}(\boldsymbol{v}^t \text{tanh}(\left[\boldsymbol{W}^{s'}_{\alpha^x} \boldsymbol{s}_k; \boldsymbol{W}^h_{\alpha^x} \boldsymbol{h}^x_n\right])$$

$$\alpha^t_{km} = \text{softmax}(\boldsymbol{v}^t \text{tanh}(\left[\boldsymbol{W}^s_{\alpha^t} \boldsymbol{s}'_{k-1}; \boldsymbol{W}^h_{\alpha^t} \boldsymbol{h}^t_m\right])$$

We also incorporate structural bias into the attention model so that it encourages Markov assumption over alignments[19]. This means that if the $i$-th source character is aligned to the $j$-th target one, then the model favors alignments from the $(i+1)$-th to $(j+1)$-th character or from the $i$th to $(j+1)$-th characters.

To do this, we experiment with several scoring functions:

- General attention: $\boldsymbol{h}^T(W_H + W_m + W_p)$
- Concatenate attention: $\boldsymbol{v}^T \text{tanh}(W_{hh} + W_{H_{hh}} + W_m + W_p)$
- We modify the scoring function on our own and add an offset by modifying terms in the scoring functions

We train the models in the language's own script and choose the hyperparamters which gave the best results in the previous section.

We also mask the output layers from the embedding modules along with the attention model. This variant of the model is used in conjunction with a recurrent bidirectional LSTM layer instead of a unidirectional LSTM.

### Results

In the unidirectional LSTM case, the accuracy for the modified structural bias is the greatest for Bengali at 55%. For Marathi, the best attention model is the concatenating attention with 84% accuracy. In both cases, general attention performs the worst at 46% and 80% respectively.

The output of all the biLSTM models are shown in Table 8.2.

| | Concatenate Attention | Custom attention | Concatenate Attention (with mask on encoder) |
|---|---|---|---|
| Bengali | **59** | 56 | 57 |
| Transliterated Bengali | **59** | 52 | 58 |
| Marathi | **83** | 79 | 84 |
| Hindi | **100** | 100 | 100 |
| Tamil | **89** | - | 85 |
| Telugu | **50** | - | 29 |
| Urdu | 85 | - | **87** |
| Kannada | **80** | - | 41 |
| Sanskrit | **100** | - | 60 |

Table 8.2: Test accuracy for multiple languages trained under different scoring functions for structured bias in attention. The models trained here are BiLSTM models with input feeding attention.

From the table, it is clear that our custom scoring function did not perform as well as the concatenate attention scoring function. Additionally, the accuracies of models have increased significantly when a BiLSTM is used.

## 8.4 Cross lingual training

We make a few changes to the monolingual training phase to adapt it for cross-lingual training. For the remainder of this section, let $L_1$ be the source language(high resource) and $L_2$ be the target language(low resource). We use a modified transfer learning method that transfers learning from a model learnt on $L_1$ to another language $L_2$. This approach[5] has been shown to give competitive results on multiple tasks.

Concretely, we share several modules across the source and target language and train the models in multiple phases on different combinations of these modules. For example, the sequence-to-sequence model, along with the attention model, is shared

across the two languages. Depending on whether we are training $L_1$ or $L_2$, the embedding module is changed.

The training phase of the model can be split into four parts:

- Copying phase for $L_1$(P1): Like in the monolingual setting, the model is allowed to learn to copy. The copying phase is stopped when the accuracy reaches 75%. At this point, the attention model has learnt the required bias in the weight matrices.

- Copying phase for $L_2$(P2): This is a similar step as before, but this for the low resource language. The embedding layer for $L_1$ is swapped out for a new embedding layer. This phase is also stopped when the copying accuracy crosses 75%. By this point, the embedding model would have learnt the projection matrix for the one-hot encodings of character tokens in $L_2$.

- Training phase for $L_1$(P3): The embedding module for $L_1$ is swapped back into the model and the learning period starts. TAt the end of this phase, the model obtained is nearly the same as that obtained in the monolingual setting.

- Training phase for $L_2$(P4): We again swap out the embedding module with the embedding module for $L_2$ and fine-tune it for the lemmatisation task. We observe that the model converges quickly in this phase compared to the training phase for $L_1$, although the time to convergence varies with different language pairs. At the end of the previous step, we observed that zero-shot transfer learning did not good results. In contrast, training the model further on $L_2$ improved the accuracy on the validation set increases by percentage points.

In different variations of the models, we freeze different parts of the model. Table 8.3 shows the validation accuracies for Hindi-Bengali models when different sets of weights are frozen. Based on the results from these models, we choose the training procedure for the models.

### 8.4.1 Results

Table 8.3 shows the accuracy on the validation set when different parts of the model are allowed to be trainable in different phases. Note that we do not change the training phases of the high resource language (P1 and P3) at any point of time. It is clear from the table that the approach described by Artetxe et al. 2019[5] does not work here verbatim. We need to allow the encoders, decoders and attention models to be trained in the last phase for the model to give good results. The best result is obtained when the model is trained end-end whenever the low resource language is being trained,

Using this training process, we create cross-lingual models for different pairs of languages. Hi-Mr model reports an accuracy of 50% and Hi-Bn models show an accuracy of 61%. For Telugu, we use Hindi and Kannada as high resource languages. The acccuracy is 84% in both cases.

| Full | Embedding | Encoder,Decoder, Attention | FCN/ Dense | Best accuracy |
|---|---|---|---|---|
| P1,P2,P4 | - | P3 | P3 | **61** |
| P1,P4 | P2 | P3 | P2,P3 | 58 |
| P1 | P2,P4 | P3 | P2,P3,P4 | 34 |
| P1,P2 | - | P3,P4 | P3,P4 | 57 |
| P1 | P2 | P3,P4 | P2,P3,P4 | 59 |

Table 8.3: Validation accuracy on Hindi-Bengali cross-lingual models when different different parts of the model are frozen in different phrases(Sec 8.4). P3 and P1 (training phase for higher resource language) remain unchanged in all rows. Each column represents the trainable part of the model

## 8.5 Other Methods

We also experimented with a few methods of our own design. The first one is word embedding data to help in the prediction task. We take the Fasttext embeddings of the words and concatenate them at the output of the decoder to help in the prediction.

We hoped that this would work because the inherent structure that is learnt in for embeddings during trainings could have helped guide the models to perform better. Additionally, the Fasttext embeddings also include subword level information and since we work at character level data, it might have boosted the training.

We also check how the performance is affected when we reduce the set of morphological data provided to the model. Instead of the complete set of morphological features, we provide only the PoS tag to the model. This approach is coupled with the embeddings based appraoach: we vary the size of Fasttext embeddings which are concatenated to the decoder output while providing only the basic PoS tag information.

Another experiment that we carried out was creating custom BiLSTM models. That is, we rearranged the inputs to the LSTMs in forward and backward directions so that the paddings in both directions aligned with each other. This would have helped the model to learn to ignore some parts of the data and focus more on a specific part, thereby boosting accuracy.

### 8.5.1 Results

The custom BiLSTM models give 0% accuracy on Bengali and only 15% and 16% accuracy with masks applied on attention model and attention + seq2seq models respectively.

Table 8.4 shows the variation of accuracy with only basic PoS tag information and different embedding sizes. It should be noted that although we reduce the information provided to the model, the accuracy does not drop as sharply as one might think. An extreme case is Marathi where the accuracy is not affected by reducing the PoS tag data.

Table 8.5 shows the variaiton of accuracy with the dimensionality of embeddings

| | 0 | 100 | 300 | Previous best |
|---|---|---|---|---|
| Bengali(transliterated) | 56 | 53 | 54 | 59 |
| Marathi | 84 | 84 | 84 | 84 |

Table 8.4: Test accuracy for Bengali and Marathi when only basic PoS tag information is provided. The numbers on different columns represent the dimensions of the Fasttext embeddings that are concatenated at the end to the decoder

used in the model as well as the type of masking used, It can be inferred that adding the embedding data does not improve the accuracy in most cases. A possibility might be that the network might need to contain deeper neural nets to extract more meaningful data from the embedding. Another possibility is that the model can learn the essential part of the embeddings on its own and thus the embeddings are just spurious. This might be justified with the fact that there is no clear correlation between the dimensionality of the embedding and the accuracy. Further ablation studies might need to be carried out to test these claims concretely.

| | Mask 1 Dim 100 | Mask 1 Dim 300 | Mask 2 Dim 100 | Mask 2 Dim 300 | Previous best |
|---|---|---|---|---|---|
| Bengali | 53 | 57 | 52 | 54 | **59** |
| Bengali(transliterated) | 8 | 57 | 53 | 57 | **59** |
| Marathi | **85** | 83 | **85** | 84 | 84 |
| Hindi | 99 | 99 | **100** | **100** | **100** |

Table 8.5: Test accuracy for languages when Fasttext embedding data is concatenated to the decoder output. Mask 1 means that a mask is applied to the attention model to ignore the contribution of padded data. Mask 2 also masks the seq2seq model. Values in bold indicate the best models and underlined values represent the best performing models when embeddings are concatenated

# Chapter 9

# Future Work and Conclusion

## 9.1 Future Work

We focus on lemmatisation as our primary task, but there are several other tasks which can also be explored. Several competitions like the SIGMORPHON shared tasks have started focusing more on low resource languages as well. Although, these tests contain only a handful of Indian languages. Creating test sets or even creating new tests as well as good baselines would go a long way in promoting research on low languages in the community.

We have covered less than 10 Indian languages as part of our BTP, but there are many more languages for which research has not been done. In some cases, the datasets are also not available. For this, we released a workshop paper in LREC 2020 titled "A Passage to India": Pre-trained Word Embeddings for Indian languages". The work released pretrained contextual as well as non contextual models for 14 Indian languages in the public domain for use by the research community.

As part of the second stage of our BTP, we have attempted to contribute towards this research by analysing the performance of models in extremely low resource settings as well as analysing the effect of different source languages in the transfer learning tasks. More concretely we do the following:

- We explore the lower limits of data required for training monolingual models which still achieve a reasonable accuracy
- In the cross-lingual setting, we analyse each source-target language pair's performance and identify the best language pair
- We evaluate the *reflexiveness* of cross-lingual learning, that is if a source-target language pair performs well, does the reverse hold true or not

All this work has been submitted for a peer reviewed conference and will hopefully will be of use to researchers looking to work in this field.

It would be helpful if concurrent research in this area on other Indian languages takes place. With digitisation,the rate of language loss has also increased sharply. To

prevent the languages from going extinct, we need to digitise the languages so that it is more accessible for people worldwide.

Alternatively, even better data augmentation techniques or more lightweight models would be beneficial for Indian languages. Any research in this area would go a long way in preserving the languages.

## 9.2 Conclusion

There has been tremendous progress in NLP in recent years with both development in both technology and amount of data that has become available, with machines performing closer to the level of humans in multiple tasks. Till now, these advancements have been limited to extremely high resource languages like English. Slowly, the research community has started to take note of low resource languages and is now actively contributing towards its research as a whole. We hope that our work as part of our BTP has contributed to this research and inspires others to take up research in the context of Indian languages, or any low resource languages for that matter.

# Bibliography


[1] A.K. Farmer R.M. Harnish A. Akmajian, R.A. Demers. *Linguistics: an Introduction to Language and Communication*. Prentice Hall, 1995.

[2] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pages 1638–1649, 2018.

[3] Antonios Anastasopoulos and Graham Neubig. Pushing the limits of low-resource morphological inflection. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 984–996, Hong Kong, China, November 2019. Association for Computational Linguistics.

[4] David F Armstrong. The gestural theory of language origins. *Sign Language Studies*, 8(3):289–314, 2008.

[5] Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. On the cross-lingual transferability of monolingual representations. *arXiv preprint arXiv:1910.11856*, 2019.

[6] Vinayak Athavale, Shreenivas Bharadwaj, Monik Pamecha, Ameya Prabhu, and Manish Shrivastava. Towards deep learning in hindi ner: An approach to tackle the labelled data scarcity. *arXiv preprint arXiv:1610.09756*, 2016.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[8] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[9] Pushpak Bhattacharyya. Natural language processing: A perspective from computation in presence of ambiguity, resource constraint and multilinguality. *CSI journal of computing*, 1(2):1–13, 2012.

[10] Pushpak Bhattacharyya, Ankit Bahuguna, Lavita Talukdar, and Bornali Phukan. Facilitating multi-lingual sense annotation: Human mediated lemmatizer. In *Proceedings of the Seventh Global Wordnet Conference*, pages 224–231, 2014.

[11] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

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

[13] Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

[14] William Bright. Ethnologue: Languages of the world ed. by barbara f. grimes, and: Index to the tenth edition of ethnologue: Languages of the world ed. by barbara f. grimes. *Language*, 62(3):698–698, 1986.

[15] Qian Chen, Xiaodan Zhu, Zhenhua Ling, Si Wei, Hui Jiang, and Diana Inkpen. Enhanced lstm for natural language inference. *arXiv preprint arXiv:1609.06038*, 2016.

[16] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

[17] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[18] Christopher Olah. Understanding lstm networks, 2015. [Online; accessed 16-November-2019].

[19] Trevor Cohn, Cong Duy Vu Hoang, Ekaterina Vymolova, Kaisheng Yao, Chris Dyer, and Gholamreza Haffari. Incorporating structural alignment biases into an attentional neural translation model. *arXiv preprint arXiv:1601.01085*, 2016.

[20] Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word translation without parallel data. *arXiv preprint arXiv:1710.04087*, 2017.

[21] Ryan Cotterell, Christo Kirov, John Sylak-Glassman, David Yarowsky, Jason Eisner, and Mans Hulden. The sigmorphon 2016 shared task—morphological reinflection. In *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 10–22, 2016.

[22] Oksana Dereza. Lemmatization for ancient languages: Rules or neural networks? In *Conference on Artificial Intelligence and Natural Language*, pages 35–47. Springer, 2018.

[23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[24] Susan T Dumais. Latent semantic analysis. *Annual review of information science and technology*, 38(1):188–230, 2004.

[25] Asif Ekbal and Sivaji Bandyopadhyay. Development of bengali named entity tagged corpus and its use in ner systems. In *Proceedings of the 6th Workshop on Asian Language Resources*, 2008.

[26] William Empson. Seven types of ambiguity (new york: New directions). *EmpsonSeven Types of Ambiguity1947*, 1947.

[27] Dean Falk. The" putting the baby down" hypothesis: bipedalism, babbling, and baby slings. *Behavioral and Brain Sciences*, 27(4):526, 2004.

[28] John R Firth. A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*, 1957.

[29] ZS Harris. Distributional structure. word, 10 (2-3), 1954.

[30] Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. Deep semantic role labeling: What works and what's next. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 473–483, 2017.

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

[32] James R Hurford, Brendan Heasley, and Michael B Smith. *Semantics: a coursebook*. Cambridge University Press, 2007.

[33] Jakob Uszkoreit. Transformer: A novel neural network architecture for language understanding, 2017. [Online; accessed 16-November-2019].

[34] Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. End-to-end neural coreference resolution. *arXiv preprint arXiv:1707.07045*, 2017.

[35] Google LLC. *SentencePiece embeddings*, 2020 (accessed June 28, 2020). `https://github.com/google/sentencepiece`.

[36] Google LLC. *Transformer: A Novel Neural Network Architecture for Language Understanding*, 2020 (accessed June 28, 2020). `https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html`.

[37] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[38] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pages 6294–6305, 2017.

[39] Arya D McCarthy, Ekaterina Vylomova, Shijie Wu, Chaitanya Malaviya, Lawrence Wolf-Sonkin, Garrett Nicolai, Christo Kirov, Miikka Silfverberg, Sebastian J Mielke, Jeffrey Heinz, et al. The sigmorphon 2019 shared task: Morphological analysis in context and cross-lingual transfer for inflection. *arXiv preprint arXiv:1910.11493*, 2019.

[40] James L McClelland, David E Rumelhart, PDP Research Group, et al. *Parallel distributed processing*, volume 2. MIT press Cambridge, MA:, 1987.

[41] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[42] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[43] Sewon Min, Minjoon Seo, and Hannaneh Hajishirzi. Question answering through transfer learning from large fine-grained supervision data. *arXiv preprint arXiv:1702.02171*, 2017.

[44] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer, 2005.

[45] Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Jan Hajič, Christopher D Manning, Sampo Pyysalo, Sebastian Schuster, Francis Tyers, and Daniel Zeman. Universal dependencies v2: An evergrowing multilingual treebank collection. *arXiv preprint arXiv:2004.10643*, 2020.

[46] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

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

[48] Fernando Pereira, Naftali Tishby, and Lillian Lee. Distributional clustering of english words. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 183–190. Association for Computational Linguistics, 1993.

[49] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[50] Oren Poliva. From where to what: a neuroanatomically based evolutionary model of the emergence of speech in humans. *F1000Research*, 4, 2015.

[51] Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Hwee Tou Ng, Anders Björkelund, Olga Uryupina, Yuchen Zhang, and Zhi Zhong. Towards robust linguistic analysis using ontonotes. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 143–152, 2013.

[52] Jonathan K Pritchard, Matthew Stephens, and Peter Donnelly. Inference of population structure using multilocus genotype data. *Genetics*, 155(2):945–959, 2000.

[53] Tobias Pütz, Daniël De Kok, Sebastian Pütz, and Erhard Hinrichs. Seq2seq or perceptrons for robust lemmatization. an empirical examination. In *Proceedings of the 17th International Workshop on Treebanks and Linguistic Theories (TLT 2018), December 13–14, 2018, Oslo University, Norway*, number 155, pages 193–207. Linköping University Electronic Press, 2018.

[54] H. Levesque R. J. Brachman. *Readings in Knowledge Representation*. Morgan Kaufmann, 1985.

[55] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[56] Ananthakrishnan Ramanathan, Jayprasad Hegde, Ritesh Shah, Pushpak Bhattacharyya, and M Sasikumar. Simple syntactic and morphological processing can help english-hindi statistical machine translation. In *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-I*, 2008.

[57] Siva Reddy and Serge Sharoff. Cross language pos taggers (and other tools) for indian languages: An experiment with kannada using telugu resources. In *Proceedings of the Fifth International Workshop On Cross Lingual Information Access*, pages 11–19, 2011.

[58] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[59] Erik F Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050*, 2003.

[60] Roger C Schank and Robert P Abelson. *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures*. Psychology Press, 2013.

[61] Hinal Shah, Prachi Bhandari, Krunal Mistry, Shivani Thakor, Mishika Patel, and Kamini Ahir. Study of named entity recognition for indian languages. *Int. J. Inf*, 6(1):11–25, 2016.

[62] Manish Shrivastava, Nitin Agrawal, Bibhuti Mohapatra, Smriti Singh, and Pushpak Bhattacharya. Morphology based natural language processing tools for indian languages. In *Proceedings of the 4th Annual Inter Research Institute Student Seminar in Computer Science, IIT, Kanpur, India, April*, 2005.

[63] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[64] Carl Vikner and Sten Vikner. Hierarchical morphological structure and ambiguity. *Merete Birkelund, Maj-Britt Mosegaard Hansen and Coco Norén, eds*, pages 541–560, 2008.

[65] Paul J Werbos et al. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[66] Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE PROJECT MAC, 1971.

[67] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.

[68] Yin-Lai Yeong, Tien-Ping Tan, and Siti Khaotijah Mohammad. Using dictionary and lemmatizer to improve low resource english-malay statistical machine translation system. *Procedia Computer Science*, 81:243–249, 2016.