Word Embeddings and Morphological Tools for Indian Languages

Kumar Saunack
160050056

Computer Science and Engineering

Indian Institute of Technology Bombay

B. Tech. Thesis Project

Contents

Chapter 1

Introduction

1.1 Language and NLP

Natural Language Processing lies at the intersection of linguistics, computer science and artificial intelligence and is concerned with interactions between the computer and humans using natural language.
The ultimate objective of NLP is read, decipher, understand, and make sense of the human languages in a manner that is valuable.

1.2 Stages and ambiguities in NLP

The following section describes the different stages in any end-to-end NLP task. The bullet points in each stage describe the problems that crop in that stage.

1.2.1 Phonology and Phonetics

Conversion of utterances (audio dialogs) into words

- Detecting word boundaries
  Audio files are ”continuous”, in the sense that it is a single file and it is up to the computer to figure out where one word ends and the next one begins. Which leads us to the next problem

- Homophones
  Sometimes words or a collection of words sound the same. The difficulty lies in choosing the correct word(s). This task can be more difficult in some languages than the others. Since Hindi has a 'static' pronunciation for all it's characters, ambiguity arises when a word can be split in different ways.
  On the other, English for example, has multiple same-sounding words like *lead,led* and *read,red*
  Issues can also arise due to ”close”-sounding words, especially in informal talks

which are fast paced. Humans can easily differentiate between different possibilities due to context, semantics and experience granted to them. But for computers, it is a challenging task to choose the correct one repeatedly.

1.2.2 Morphology

Choosing the correct stem from the root

- This is similar to detecting word boundaries. Often words can be split into different stems and the mistake is difficult to recognise on just a glance.
  The difficulty can vary depending on the language since there is a lot of morphological diversity among different languages.

1.2.3 Lexicon

Storage of words and associated knowledge. It includes the semantic, morphology and Part-of-Speech(PoS) tags.

- Choosing between phrases and individual words
  Words when combine to form phrases, the meaning can change unpredictability. An example would be *over the moon*. Taken literally, it would mean that someone has passed the moon in outer space. But in day-to-day usage, it means extremely happy.

- Polysemy
  Each word can have multiple meanings, each used in different contexts and with their own nuances. Looking at a word out of context may not offer any insight as to what it means in a sentence.
  This also extends to cases where a word can be used with different PoS in different cases.
  Polysemy is also extendable to phrases which have multiple meanings. *Heads up* can be used as a warning or it can be used to describe alertness of something.

- NER
  Sometimes, nouns can refer to specific entities. This may be clear in some cases where the first letter of the proper noun is capitalised. But in cases where the proper noun is at the start of a sentence, it may be difficult to differentiate between the two (proper noun and common noun). To give an example, consider the sentence "*Apple is the first company to reach the trillion dollar mark*". The word *Apple* can refer to both the fruit apple or the company Apple which manufactures electronic devices. In this case, the latter meaning has been used. Sometimes proper nouns creep into daily usage and can be confused for proper nouns when, in fact, they stand for a general object. *John Doe* is a popular example. In the United States, it is used by law enforcement agencies to refer to an unknown male. Care needs to be taken to identify these special cases.
  Another case where named entities may be recognised incorrectly is when there

is a confusion between specific people, places and organisations. This ambiguity can occur in our daily speech, but care needs to be taken to resolve this properly. *Florence* is an example where a person-place ambiguity occurs. It can refer to a female or the city in Italy.

- Abbreviations
  An abbreviation can stand for multiple things. *DC*, for example, can refer to both direct current in electricity or Washington DC, the capital of the United States.

1.2.4 Parsing

Processing of hierarchical structure and finding relations between words

- Scope ambiguity
  Adjectives can attach to a noun phrase following it or the noun just after it. In cases where a noun phrase is comprised of multiple nouns, it can be difficult to determine if the adjective is distributed over all the nouns or just the noun following it. An example to demonstrate this would be *The dog brings me the newspaper every morning.* This can be taken to mean that the dog brings the daily newspaper each morning or that the dog brings the same old specific newspaper everyday.

- Attachment ambiguity
  Due to the structure of sentences in different languages, the problem of the assignment of prepositional phrases might give rise to an ambiguity. In English, the sentence structure usually followed is $V - NP_1 - P - NP_2$. The problem is choosing to decide whether to assign the prepositional phrase to the first noun phrase or to the second. To illustrate this, consider the example *The boy plays games with friends.* This can be taken to mean that the boy plays games with his friends, with his friends as his co-players. The other sense would be that he plays games *on* his friends, with his friends being the subject of the games.
  In some cases, this problem can be solved with schema, ie, knowledge about the topic. In NLP parlance, this is known as *selectional preference*. This can be learnt from annotated corpora via Machine Learning based methods. The other alternative is creating manual rules but it does not ensure completeness or correctness.
  Languages with different sentential structures, like Hindi where post-positions are used instead of prepositions, may be unaffected by this problem.
  A note to keep in mind is that attachment ambiguity can occur with phrases or clauses too since they effectively act as a substitute for different parts of speech. It is not too hard to think of examples where this is the case.

1.2.5 Semantics

Processing of meaning

- Semantic role ambiguity : agent vs beneficiary
  A noun can be interpreted as either the object on which the action/quality is being conferred(beneficiary) or the one on who carries out the action(agent). This ambiguity arises partly due to the multiple meanings of a word. An example which illustrates this would be *Helping people are can be too much sometimes.* In this case people who are helpful can be considered as a nuisance, i.e., they are considered as beneficiaries here. On the other hand, it can also be taken to mean that helping others can be too much for a person in some cases (agent).

- Representative vs specific object[12]
  This happens in cases where a noun is considered as a representative of the whole case. In general usage, the noun may be considered as referring to a specific entity. *The dog is going to sleep* and *The dog has been a companion for humans since long* differ in the way they refer to dogs. The first one refers to a specific dog. But in the second sentence, the dog is taken to represent the domesticated canine race.

1.2.6 Pragmatics

Processing of user intention, modeling etc.

- Deriving user intention is a very complex task which depends on a multitude of things, some of which include context, situation specificity, tone, usage etc.

1.2.7 Discourse

Processing of connected text

- Standalone, individual sentences can mean multiple things until a context is provided. Once enough context is provided, information can be gleaned from the text. But the problem remains of forming the context from sentences. Sentences are spoken one after the other and the order in which they are said is of importance, that is, chronology plays an important part. So depending on how sentences proceed, the meaning can change. Each new sentence provides potentially new information through which the earlier assumptions can be updated. This is why processing connected text in a proper manner is an important topic.

  1: *Today was not a productive day*
  2: *Are you sure?*
  1: *On further thought, you're correct.*

  In this case the conclusion that one would draw is that the day was productive for the people talking. But if lines 1 and 2 are interchanged, the outcome changes in the opposite direction. Depending on the order in which these statements are heard, two speakers can reach completely different conclusions.

1.3 Language Models

A language model is a probability distribution over sequences of words. Given a sequence, it assigns a probability value to the whole sequence. Mathematically,

$$P(w_1, w_2, .., w_n) = \Pi_i p(w_i|w_1, w_2, .., w_{i-1})$$

Now, based on some assumptions, this can be simplified further. If we assume a unigram distribution, ie, each word appears independently of it's context, the probability reduces to

$$P(w_1, w_2, .., w_n) = \Pi_i p(w_i)$$

For a bigram distribution, a word is assumed to depend only on the word preceding it:

$$P(w_1, w_2, .., w_n) = \Pi_i p(w_i|w_{i-1})$$

In general, for an n-gram distribution, the probability value turns out to be

$$P(w_1, w_2, .., w_n) = \Pi_i p(w_i|w_{i-(n-1)}, .., w_{i-1})$$

Any good language model, qualitatively, should assign higher probability values to more frequent sentences than the ones which are wrong or are rare. A popular metric for this is given by perplexity, defined on any test sentence as

$$PP(W) = P(w_1, w_2, .., w_N)^{-1/N}$$

Minimising perplexity is the same as maximising the probability of the sentence.
To account for unseen words in the test set, smoothing is often done before calculating the probability values.

1.4 Word embeddings

Word embeddings are numerical representations of words. Since computers take numbers as input, it is necessary to find some sort of representation for words. A naive idea would be to assign a single number to each word. But this idea is not good because of 2 reasons: (i) the number of words in any language is very huge (ii) this representation does not allow for any relationships between any two words except for lexicographic ordering. A better way is to embed the words in a vector space, with each word being referred to by a vector.
This raises the question of the desired properties in any word embedding

1.4.1 Characteristics of word embeddings

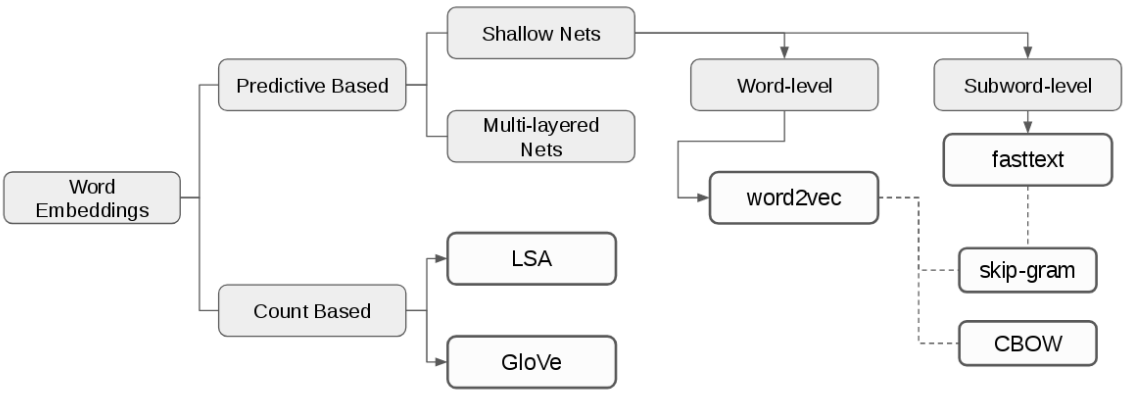
- The word embeddings should model the semantic similarity between words. For example, man and men have the same meaning except for the count. One would expect their embeddings to remain close in the embedded space too.

- Denseness of the vector space: The embeddings of all words should be dense, ie, the vector space should not be sparsely filled. The average distance between words should not be arbitrarily large.
- Relations between words should be maintained by the embeddings. A famous example is the king:man::queen:woman. In this case, if the vector representations also follow this rule, it means that the embeddings follow semantic relations between words too.
- Dimensions of the embedding should capture different properties of the word. If the word 'man' has some embedding, it is desirable that a dimension capture the gender of the word so that the semantic relation between man and woman is captured
- It should perform well on NLP tasks. Without qualitative results, the embeddings cannot be called a good embedding.

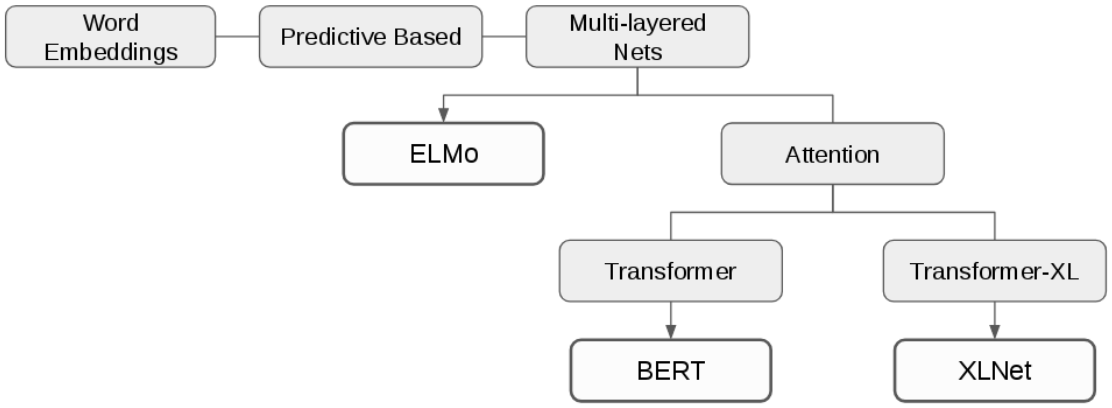
1.5 Word embedding models

Word embedding models have evolved from simple statistical, SVD based methods to neural network based methods since the 2000s. The majority of word embedding models these days are based on neural networks.
Figure 1-1 highlight the major word embedding models.

All the models except Word2Vec (Mikolov et al. 2013) and ELMo (Peters et al. 2018) have been explained later in the report.



(a) Non-contextual word embedding models



(b) Contextual Word embedding models (embeddings can change depending on context)

Figure 1-1: Word embedding models

Chapter 2

Word embedding models

2.1 GloVe

In the paper by Pennington et al. 2014 [24], the authors start with the idea that the global co-occurence data is an important information which can be used to improve the quality of word embeddings. The seminal paper by Mikolov et al. 2014 [22] utilised only local contexts. The idea in GloVE embeddings is to get embeddings of words in vector spaces such that words that appear close together frequently are close in the embedding space too. The key difference from earlier methods like LSA is that the earlier global matrix methods did not learn dimensions of meaning. For example, some dimensions of an embedding can represent the gender. This allows word2vec to perform well on analog tasks using vector arithmetic.

First, highlighting the notations used : $P_{ij} = P(i|j) = X_{ij}/X_i$, ie, P_{ij} represents the probability that word j appears in the context of word i. For example, consider the co-occurrence table presented in the paper in Figure 2-1. The probability values in themselves are not useful indicators about the co-occurence of the words. When we take the ratio of the co-occurrence of two different words in the same context or vice versa, we can get an idea of which word is more likely to appear in the context. From the figure, it is clear that the word 'solid' is more likely to appear in the context of the word ice rather than in the context of steam, which makes sense. Similarly, gas is more likely to appear in the context of steam rather than in the context of ice. 'fashion' on the other hand, is equally likely to appear in the context of either ice or steam because it is unrelated to either of them.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Figure 2-1: The co-occurrence probabilities for different words in the corpus

This suggests that the model work on the ratio of the co-occurrence probailities:

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

where w_i is the word embedding for word i and $\tilde{w}$ is the context vector. F is a function which takes the embeddings as input and gives the ratio as output.
Since it is desired to give meaning to simple arithmetic between vectors, it is best if the input to F is also a simple arithmetic between the vectors. The simplest way is to make it the difference between the vectors for words i and j.

$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

Again, since a simple relation is desired between the context vectors and word vectors, the best way is to use a dot product. This allows a relation to maintain a linear structure:

$$F((w_i - w_j)^T \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

To determine F, the equation can be simplified using two tricks:

1. If the log of ratio is used instead of the ratio itself, the division is converted to a subtraction which matches the input structure. Also, since log is a monotone increasing function, the relative structure in the embedding space is maintained.

2. The model currently does not capture the fact that some words occur more than others. To account for this fact, a bias term is added for each word in the input

After making these modifications, the resultant equation turns out to be

$$(w_i - w_j)^T \tilde{w}_k + b_i + b_j = \log(P_{ik}) - \log(P_{jk})$$

Matching the structures on the LHS and RHS, the equation can be simplified to

$$w_i^T \tilde{w}_k + b_i = \log(P_{ik}) = \log(X_{ik} - \log(X_i))$$

Consuming the $log(X_i)$ into the bias term and adding a bias term for w_k for symmetry, we get

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$$

A main drawback of this model is that it assumes that all co-occurrences are of the same quality. In reality, words that appear more frequently together carry more information compared to other, rare co-occurring words. This is because the rare co-occurrences are noisy and do not add much value. To take this into account, a modified loss function was proposed that introduces a weighting function into the

model:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_k + b_i + \tilde{b}_k - \log(X_{ik}))^2$$

where V is the size of the vocabulary. The weighting function should satisfy the following properties:

1. The function should approach 0 as x tends to 0 such that the $f(x)(\log x)^2$ is finite in the limit 0
2. It should be non-decreasing
3. It should not be too high for frequently co-occurring words so that they are not overweighted

The authors found $f = min(1, (x/x_{max})^{3/4})$ to be a suitable function for their purposes

2.1.1 Similarity to Word2Vec

The authors state that the GloVE model is similar to the Word2Vec model, the only difference being the type of loss function being considered.

$$J = -\sum_{i \in corpus, j \in context_i} \log Q_{ij}$$

where Q_{ij} is the softmax over the dot product of vector representations of w_i and w_j. Now, $\log Q_{ij}$ is added when i, j appear toegether in the corpus, ie, $\log Q_{ij}$ is added X_{ij} times. Using the relation between P_{ij} and QX_{ij},

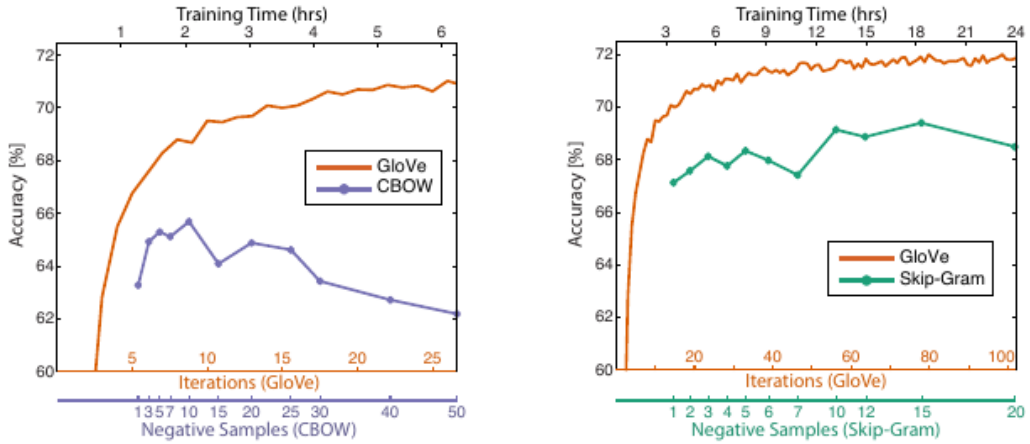
$$J = -\sum_{i=1}^{V}\sum_{j=1}^{V} X_{ij} \log Q_{ij}$$

$$J = -\sum_{i=1}^{V} X_i \sum_{j=1}^{V} P_{ij} \log Q_{ij}$$

So the loss term in word2vec is a frequency weighted cross-entropy loss between the predicted and actual word distribution. Again, here the term X_i represents that each data has equal weightage. However, the paper does suggest that using negative sampling and filtering the data improves the performance. This means that the weight function can be an arbitrary function of X_{ij}

$$J = -\sum_{i,j} f(X_{ij}) P_{ij} \log Q_{ij}$$

The GloVe paper argues that log mean squared error is better than cross-entropy because cross entropy tends to put too much weight on the long tails. Also, a disadvantage of using cross-entropy loss is that the distribution Q needs to normalised which increases the computation requirements.



(a) GloVe vs CBOW (b) GloVe vs Skip-Gram

Figure 2-2: Comparison of GLoVE with word2vec (a) skipgram and (b) CBOW models on the word analogy task[24]

2.1.2 Results

The authors compare the results of their embedding model with word2vec on the word analogy task of Mikolov et al. 2013 and on the Co-NLL-2002 shared benchmark dataset for NER (Tjong Kim Sang and De Meulder, 2003).
The authors found that GLoVE consistently out-performed both word2vec and the baseline SVD models on word-similarity tasks on all datasets. A comparison with word2vec is shown in Figure 2-2, keeping the same training corpora, embedding dimension and window size in both of them.

2.2 Fasttext

Popular word model representations before FastText by Bojanowski et al. 2017 [6], did not use subword information while learning word representations. This was a limitation, especially for languages with large vocaulary and many rare words. Fasttext introduced a new approach based on the skipgram model where each word is represented as a a bag of character n-grams.
The loss function in the skipgram model introduced by Mikolov et al. 2013 is given by:

$$\sum_{t=1}^{T} \sum_{c \in C_t} \log p(w_c | w_t)$$

where w_i is the i^{th} word in the vocabulary W and C_t is the set of indices of words surrounding w_t and the training corpus is represented as a sequence of words $w_1, w_2, ..., w_T$.
One choice of the probability of a context word is given by the softmax, but it predicts only one context word. So the authors frame the prediction of context words as a set of independent binary classifications. The goal is then modified into predicting the

presence or absence of words in a context. Formally, given a context position c for a word at position t, the loss function is given as:

$$\log(1 + e^{-s(w_t,w_c)}) + \sum_{n \in N_{t,c}} \log(1 + e^{s(w_t,n)})$$

where $N_{t,c}$ is the set of negative samples sampled form the vocabulary. Denoting the logistic loss function by l, the objcetive function turns out to be

$$\sum_{t=1}^{T} [l(s(w_t, w_c)) + \sum_{n \in N_{t,c}} l(-s(w_t, n))]$$

The scoring function used by Mikolov et al. 2013 takes the dot product of the vector representations of the words. This is where Bojanowski et al. 2017 deviate from word2vec. They argue that by incorporating the internal structure of words, the model can perform better.
To do this, each word is represented as a bag of $n-$gram. Special boundary symbols $<$ and $>$ are added at the beginning and end of words respectively to distinguish prefixes and suffixes from other sequences. The word w itself is also included in the bag of $n-$grams so that the model can also learn a representation for the word. Taking the word *human* as an example with $n = 3$, the representation will be

<hu,hum,uma,man,an>

along with the special sequence

<human>

This example shows why $<$ and $>$ are important: `man` and <man> are different. The first one is a subword while the second one is the word representation for the word man itself.
In practice, the authors extract all $n-$grams for n greater than 2 and less than 7.
Now, each word is represented by the sum of the vector representations of all it's $n-$grams. Formally, if G represents the size of the dictionary of all $n-$grams, and $G_w \subset 1, 2, ..., G$ represents the set of $n-$grams appearing in word w, the word representation for w is

$$v_w = \sum_{g \in G_w} z_g$$

This gives us th scoring function

$$s(w, c) = \sum_{g \in G_w} z_g^T v_c$$

where v_c is the similar representation for the word c. An advantage of this model is that it allows sharing of representations across words. This means that reliable representations of rare words can be learnt. For example, `human` and `mango` share

		sg	cbow	sisg
CS	Semantic	25.7	27.6	27.5
	Syntactic	52.8	55.0	77.8
DE	Semantic	66.5	66.8	62.3
	Syntactic	44.5	45.0	56.4
EN	Semantic	78.5	78.2	77.8
	Syntactic	70.1	69.9	74.9
IT	Semantic	52.3	54.7	52.3
	Syntactic	51.5	51.8	62.7

(a) Word analogy task

		sg	cbow	sisg-	sisg
AR	WS353	51	52	54	**55**
DE	GUR350	61	62	64	**70**
	GUR65	78	78	**81**	**81**
	ZG222	35	38	41	**44**
EN	RW	43	43	46	**47**
	WS353	72	**73**	71	71
ES	WS353	57	58	58	**59**
FR	RG65	70	69	**75**	**75**
RO	WS353	48	52	51	**54**
RU	HJ	59	60	60	**66**

(b) Word similarity task

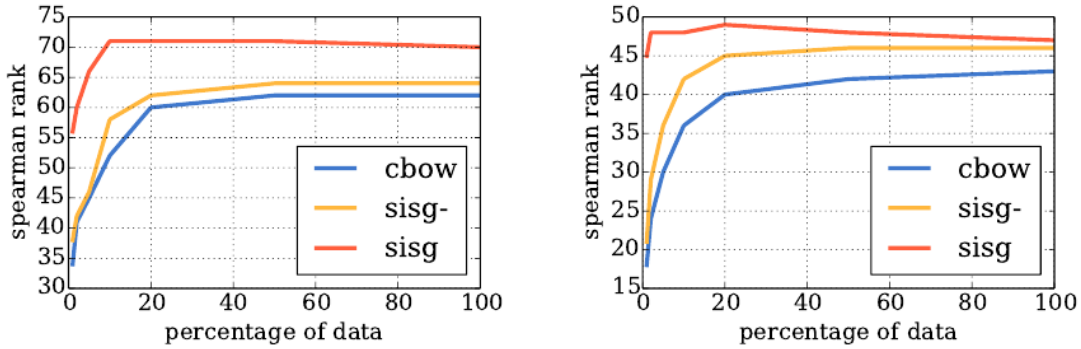
Figure 2-3: Comparison of fasttext with word2vec (a) skipgram and (b) CBOW models on different tasks [6]

the subword `man` and during training both of them will update the representation of the subword `man`. Now, suppose if `manatee` is a rare word, the subword information learnt from the training corpora can be used to create a representation for the target word. This is better than randomly selecting any vector representation for the word because the morphology of the word contributes some meaning to the representation of the word.

2.2.1 Results

The authors compare their models to word2vec skipgram and CBOW models. The vector dimension used in all cases is 300 with a window size of 5 and 5 negative samples per positive sample. The testing is carried out on a variety of languages including English, Russian, German amongst others.

For the human similarity/relatedness task, the fasttext model performs as well as other models. The improvement is especially noticeable in the rare english words dataset (RW) and languages which are morphologically rich like Arabic, German and



(a) DE-GUR350 (b) EN-RW

Figure 2-4: Comparison of fasttext with word2vec (a) skipgram and (b) CBOW models on the word analogy task with different training corpora size[6]

Russian. For example, table tennis is German is written as 'tischtennis'. By using the representation for tennis, the model can identify the similarity between table tennis and tennis in German. However, the fasttext model does not offer any advantage over the English dataset containing normal words (WS 353). Since the words are common, good vectors can be obtained for them without using subword information. The results are displayed in Figure 2-3b. `sisg-` refers to fasttext where a null vector is assigned for OOV words. In `sisg`, subword representations are used to create a representation for OOV words.
For the word analogy task, fasttext improves on the syntactic tasks but does not outperform the baselines in semantic tasks. In some cases like German and Italian, it instead degrades the performance on semantic tasks. Again, morphologically rich languages like German and Czech benefit more from fasttext than less rich languages. Another benefit of fasttext is that it requires substantially less training data than word2vec to perform well on the tasks, This is because fasttext utilises subword information so it can learn representations for a larger range of words more easily. This also means that the model's performance saturates more quickly. But the main advantage remains that even with a restricted training corpora, fasttext can learn effective representations.

2.3 XLNet

The BERT model introduced by Devlin et al. [11] improved the performance on many NLP tasks, including question-answering, next sentence prediction and semantic similarity. They did this by including bi-directional context in their training. Earlier models like ELMo trained two separate models, but they were trained separately.
Before understanding the improvememtns of XLNet over BERT, it is necessary to understand the drawbacks of BERT:

1. Pretrain - finetune discrepancy: the [MASK] tokens used in BERT to learn deep

bidirectional contexts during prertraining does not occur during the finetuning phase. The authors try to resolve this by masking only some tokens with a certain probability and introducing noise in the input.
To understand this more concretely, we need to understand what does for non-[MASK] tokens. The model can easily learn to just copy the input into the output. Even after introducing noise, since noise represents only a small percentage of the actual data, the actual percentage of noisy tokens that BERT learns over is small.

2. Independent predictions: each [MASK] token is predicted independently. This prevents BERT from learning dependencies between predicted tokens. An example would be predicting the output for

   `[MASK][MASK] was the president of [MASK]`

   This can be filled in multiple ways:

   `Barack Obama was the president of USA`

   Another way would be

   `Pranab Mukherjee was the president of India`

   However, the following is not a valid output:

   `Barack Obama was the president of India`

   BERT is unable to handle this precisely because it learns the predictions for tokens in parallel. Although [MASK] tokens are not present during fine tuning, this is a problem with BERT because it reduces the number of dependencies that BERT can learn at once. A point to note however: this problem is not present in traditional language models. This is because they generate words in a specified order so they can learn dependencies between all words in a sentence.

XLNet[33] aims to improve upon these drawbacks while keeping the advantage of BERT : deep bidirectional context. The bidirectional context was a crucial point in favour of BERT which helped it to perform so well on so many tasks. The way XLNet does this is by introducing permutational language modelling.
Mathematically, traditional language models performs training by maximising likelihood under auto-regressive factorisation:

$$\max_{\theta} \sum_{t=1}^{T} \log p_\theta(x_t | x_{<t})$$

where the probability is softmax over $h_\theta(x_{1:t-1})$ (the contextual representation generated by the model) and the word embedding of x_t. BERT, on the other hand,

is based on denoising auto-encoding. The training objective is to reconstruct the original sequence given the corrupted sequence (obtained by masking tokens)

$$\max_{\theta} \sum_{t=1}^{T} m_t \log p_\theta(\tilde{x}|x`)$$

where $x`$ is the corrupted version generated by replacing tokens with [MASK], $\tilde{x}$ is the original sequence, $m_t = 1$ indicates that $x_t$ is masked. The softmax over here is over $H_\theta(x`)$ (the hidden vectors generated by a transformer) and the embedding of x_t.
XLNet instead optimises the objective

$$\max_{\theta} \mathbb{E}_{z \sim Z_T} \left[\sum_{t=1}^{T} \log p_\theta(x_{z_t}|x_{z<t}) \right]$$

where Z_T is the set of all possible permutations of length T index sequence [1,2,..,T]. z_t represents the t^{th} element in the permutation.
This fits into both the AR and AE models. The AR model is incorporated because it looks only at elements before it in the permutation to predict the next token. For the AE model, since we share the model parameters across all factorisations, in expectation, x_t has access to all elements except itself. In this way, it can learn deep bidirectional context.
The permutation here does not modify the actual sequence order but the order in which the tokens are predicted. To better understand this, consider Figure 2-5. It is clear that the sequence remains the same, the only thing that changes is the order in which we predict the tokens. In the first case, the first word to be predicted is 3, so the model predicts it based just on the initial state. In the second case, the factorisation order is 2,4,3,1. So the second token is predicted first, based on the initial state of the model. Given the new state, the fourth token is predicted. Now based on the state obtained after predicting the fourth token, the third token is predicted.
Note that if we average over all the 4 examples, the prediction for x_3 includes context from both directions which means that the model is able to learn bidirectional context.

2.3.1 Implementation details

The permutation language model cannot work with the naive Transformer implementation. This is because Transformers, when predicting a token at position i, masks the embedding for the word completely, including it's positional embedding. This can present a problem when using XLNet. To see where the problem, consider permutations $z^{(1)}$ and $z^{(2)}$ such that

$$z^{(1)}_{<t} = z^{(2)}_{<t} \quad \text{but} \quad z^{(1)}_t = i \neq j = z^{(2)}_t \tag{2.1}$$

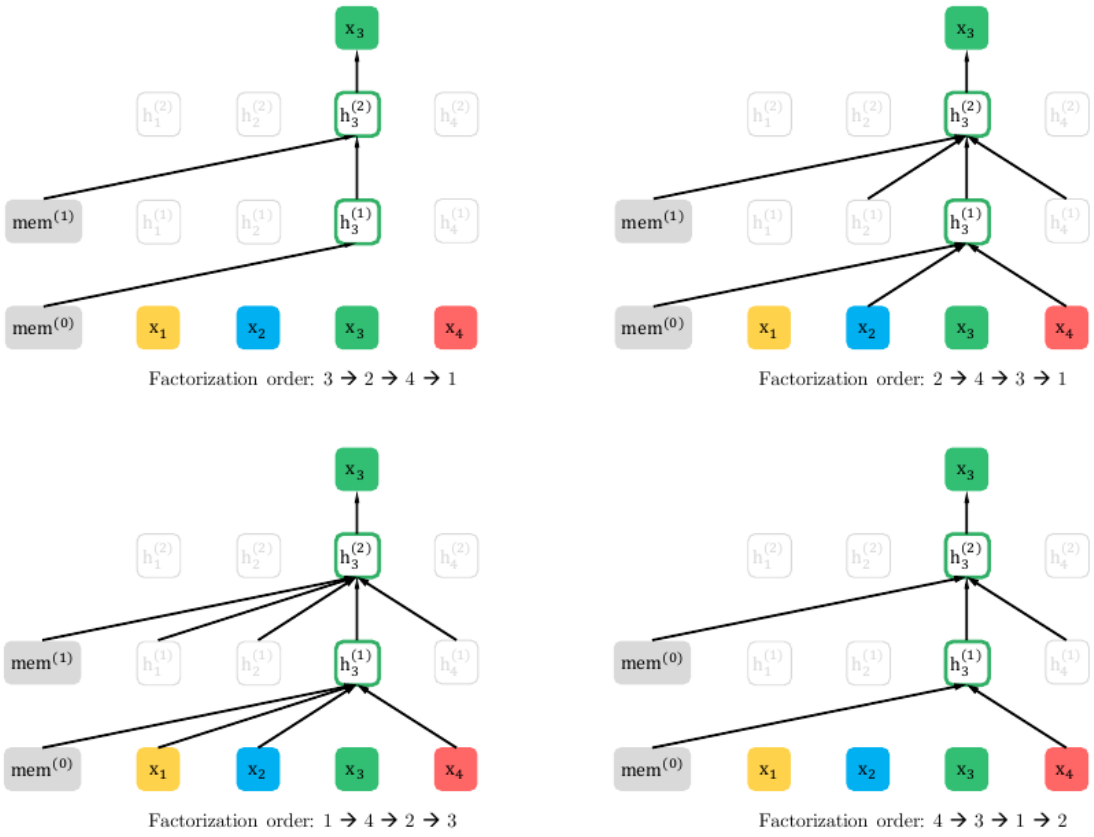


Figure 2-5: Permutation language modelling objective for predicting x_3 for different permutation orders[33]

If we consider the standard softmax formulation $p_\theta(X_{z_t} = x|x_{z<t})$ using the embedding of x and $h_\theta(x_{z<t})$ we can see that the output will be the same regardless of i and j. This is clearly not desired since different parts of a sentence have different structures, for example, the beginning of the sentence has a different structure than the middle or end.
To solve this problem, the next-token distribution is made position-aware by using $g_\theta(x_{z<t}, z_t)$ instead of $h_{theta}(x_{z<t})$, ie,

$$p_\theta(X_{z_t} = x|x_{z<t}) = \frac{exp(e(x)^T g_\theta(x_{z<t}, z_t))}{\sum_{x'} exp(e(x')^T g_\theta(x_{z<t}, z_t))}$$

The problem remains of how to incorporate this into the Transformer architecture. The problems faced are

1. to predict token x_{z_t}, $g_\theta(x_{z<t}, z_t)$ should use only the position z_t and not the context x_t otherwise the problem becomes trivial

2. to predict other tokens x_{z_j} with $j > t$, $g_\theta(x_{z<t}, z_t)$, the context needs should also be provided to provide complete contextual information.

To resolve these issues, the authors propose using two sets of hidden representations instead of the usual one:

- content representation $h_\theta(x_{z\ leqt})$ which is the same as the hidden states in a Transformer: it includes both the context and x_{z_t} itself
- query representation $g_\theta(x_{z<t}, z_t)$, which has access to contextual information of words before itself but not x_{z_t} itself. It also includes the position z_t

The initialisation is done as follows : $g_i^{(0)} = w$ where w is a trainable vector and $h_i^{(0)} = e(x_i)$. The update rules are as follows:

$$g_{z_t}^{(m)} \leftarrow \text{Attention}(\text{Q} = g_{z_t}^{(m-1)}, \text{KV} = h_{z<t}^{(m-1)}; \theta)$$

$$h_{z_t}^{(m)} \leftarrow \text{Attention}(\text{Q} = h_{z_t}^{(m-1)}, \text{KV} = h_{z\leq t}^{(m-1)}; \theta)$$

where Q,K and V represent query, key and value in an attention operation. Note that the update rule for the content stream remains the same so during finetuning, the query stream can be dropped and use the content stream as a normal Transformer XL. Also, $g_{z_t}^{(M)}$ can be used to compute the probabiltiy values .
As an example, consider the sentence

`Dogs and cats are mammals`

and we are supposed to predict `are` and the previous words in the permutation are `dogs` and `mammals`. The content stream would encode information for `dogs` and `mammals`. The query stream would encode the positional information of `are` and the information from the content stream.
Since the model is more complicated, the convergence rate is slow. To speed up the process, only the last n tokens in the permutation are predicted. Figure 2-6 illustrates partial prediction more clearly. Mathematically, we change the objective to

$$\max_\theta \mathbb{E}_{z\sim Z_T} \left[\sum_{t=c+1}^{T} \log p_\theta(x_{z_t} | x_{z<t}) \right]$$

Many downstream tasks in NLP require multiple segments of text as input like the question answering task. To handle this, BERT learns segment embeddings. XLNet also uses the [CLS] and [SEP] tokens like BERT to demarcate different segments. But instead of assigning absolute embeddings, XLNet learns relative embeddings. Given a pair of positions i and j in the sequence, if they are from the same sequence, $s_{ij} = s_+$ otherwise $s_{ij} = s_-$ where s_+ and s_- are learnable model parameters. Now, during attention calculation, the attention weight between i and j is calculated as $a_{ij} = (q_i + b)^T s_{ij}$ where q_i is the query vector and b is a learnable bias vector. a_{ij} is then added to the attention weight. The benefit of using relative segment embeddings is that XLNet can be extended to handle any number of segments, instead of just 2 like in BERT.

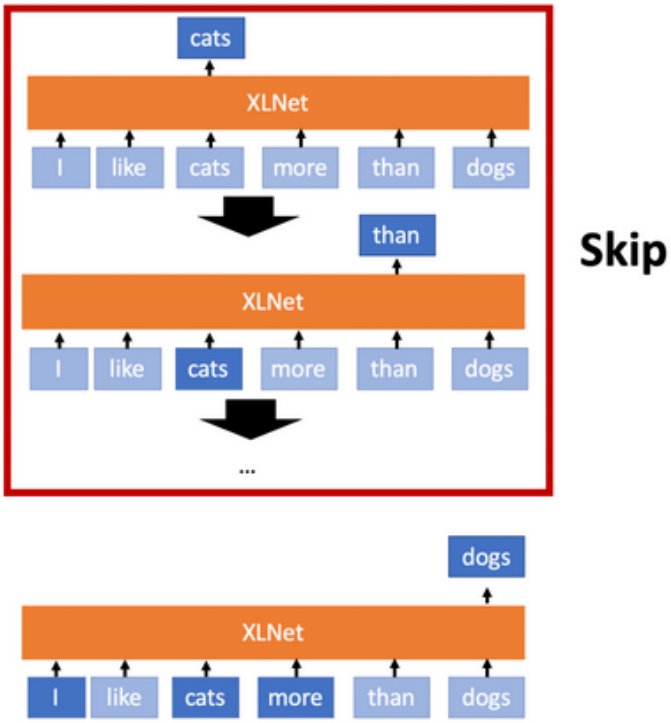


Figure 2-6: Illustrating partial prediction in XL-Net. If the permutation order was "cats", "than", "more", "I" and "dogs" in that order, the prediction for the first 4 permutation elements are skipped and "dogs" is directly predicted[19]

2.3.2 Comparison with BERT

As stated earlier, one disadvantage that BERT has is that it predicts the output for [MASK] tokens in a parallel fashion. This is a problem, because it allows for lesser dependencies to be learnt. To see how XLNet improves on this, consider the sentence

`New Delhi is the capital of India`

Suppose both BERT and XLNet use the tokens [New, Delhi] as prediction targets and maximises $\log p$(New Delhi|is the capital of India). Additionaly, suppose that the factorisation order is [is, the, capital, of, India, New, Delhi]. The training objective then becomes

$$J_{BERT} = \log p(\text{New}|\text{is the capital of India}) + \log p(\text{Delhi}|\text{is the capital of})$$

$$J_{BERT} = \log p(\text{New}|\text{is the capital of India}) + \log p(\text{New}|\text{Delhi is the capital of India})$$

So in this case, XLNet is able to capture the dependency between (New, Delhi), which is omitted by BERT.
In a more general setting, given a sequence $x = [x_1, x_2, ..., x_T]$, define a set of target context pairs $I = \{(x, U)\}$, wher U is the set of tokens that form the context of x. In the sentence given above, some target-context pairs could be (Delhi, {New}), (Delhi, {capital}), (Delhi, {New, capital}).
Now given a set of target tokens $\mathbb{T}$ and a set of non-target tokens $\mathbb{N} = x$

$\mathbb{T}$, the log probability maximised by BERT and XLNet are

$$J_{BERT} = \sum_{x \in \mathbb{T}} \log p(x|\mathbb{N}) \quad \text{and} \quad J_{XLNet} = \sum_{x \in \mathbb{T}} \log p(x|\mathbb{N} \cup \mathbb{T}_{<x}) \tag{2.2}$$

where $\mathbb{T}_{<x}$ are the tokens in $\mathbb{T}$ that have a factorization order prior to x. Consider the following cases:

1. If $U \subset \mathbb{N}$, the term $p(x|U)$ occurs in both BERT and XLNet
2. If $U \subset \mathbb{N} \cup \mathbb{T}$ and $U \cap \mathbb{T} \neq \phi$, the dependency is included only in XLNet. As a result, XLNet can learn more dependencies than BERT, which in turn means better training signals.

2.3.3 Results

XLNet achieves state of the art results across 18 tasks including text classification, natural language inference, coherence resolution, document ranking, question answering etc.
Highlighting the improvements made by XLNet in different areas:

- Long text understanding (Figure 2-7a: for testing this, the RACE dataset [20] was used. It contains questions for middle and high school students for Chinese students. The average length of the passage is more than 300 which makes it a challenging benchmark for long text understanding
- Text classification : XLNet performs better than previous models by a large margin on the IMDb, Yelp, Amazon and other databes
- GLUE dataset: XLNet outperforms BERT on 7 out of 9 tasks. The tasks on which BERT performs better than XLNet are checking the semantic equivalence of two questions (QQP) and checking the linguistic accepting of a sentence (CoLA). On all other inference and similarity tasks, XLNet gives the best results.
- Question answering : XLNet outperforms previous state-of-the-art models on the SQuAD 1.1 dataset. The dataset contains questions that always have a corresponding answer in the given passages. Here also, XLNet performed signifcantly better than multiple BERT-based models.

RACE	Accuracy	Middle	High
GPT [25]	59.0	62.9	57.4
BERT [22]	72.0	76.6	70.1
BERT+OCN* [28]	73.5	78.4	71.5
BERT+DCMN* [39]	74.1	79.5	71.8
XLNet	**81.75**	**85.45**	**80.21**

(a) Results of XLNet on the RACE dataset

Model	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	WNLI
Single-task single models on dev									
BERT [2]	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-
XLNet	**89.8/-**	**93.9**	**91.8**	**83.8**	**95.6**	**89.2**	**63.6**	**91.8**	-
Single-task single models on test									
BERT [10]	86.7/85.9	91.1	89.3	70.1	94.9	89.3	60.5	87.6	65.1
Multi-task ensembles on test (from leaderboard as of June 19, 2019)									
Snorkel* [29]	87.6/87.2	93.9	89.9	80.9	96.2	91.5	63.8	90.1	65.1
ALICE*	88.2/87.9	95.7	**90.7**	83.5	95.2	92.6	**68.6**	91.1	80.8
MT-DNN* [18]	87.9/87.4	96.0	89.9	**86.3**	96.5	92.7	68.4	91.1	89.0
XLNet*	**90.2/89.7**†	**98.6**†	90.3†	**86.3**	**96.8**†	**93.0**	67.8	**91.6**	**90.4**

Table 4: Results on GLUE. ∗ indicates using ensembles, and † denotes single-task results in a multi-task row. All results are based on a 24-layer architecture with similar model sizes (aka BERT-Large). See the upper-most rows for direct comparison with BERT and the lower-most rows for comparison with state-of-the-art results on the public leaderboard.

(b) Result of XLNet on the GLUE benchmark tests

Figure 2-7: Performance of XLNet on various tasks[33]

Chapter 3

RNN/LSTM, Attention, Transformers, Self-attention

3.1 RNNs and LSTMs

Recurrent neural networks[7][17][23] were introduced to be able handle temporal relations in data. Feedforward networks managed well on classification and pattern recognition tasks. But since there was no temporal dependency learnt in the model, it failed to perform as well in temporal tasks like word prediction, recommendation tasks etc.
At the heart of recurrent neural networks, there is a capability for the network to 'memorise' parts of the input which allows it to learn from sequential data. A simple illustration for an RNN is given in Figure 3-1. At each timestep, the RNN is fed an input x_t and it produces an output h_t which is also fed as a state input to the next timestep. This looping of output back into the input at the next timestep is what enables RNNs to learn temporal dependencies. The update rules are given by:

$$\begin{aligned} h_t &= f(W^{(hh)}h_{t-1} + W^{(hx)}x_t) \\ y_t &= softmax(W^{(S)}h_t) \\ J^{(t)}(\theta) &= \sum_{i=1}^{|V|} y_{t_i} \log y_{t_i} \end{aligned} \tag{3.1}$$

In theory, RNNs are capable of handling "long-term dependencies." For example, consider the example "He is from Germany so he must know how to speak _____". A human can understand that since the sentence is talking about a language and that the native country is Germany, the answer must be German. Note that in this case, we need to use data from the previous word as well as data present in the sixth words in the 'past'. An RNN could possibly learn this dependency too and give the correct output. But as the dependencies grow farther apart, in practice, RNNs do not seem

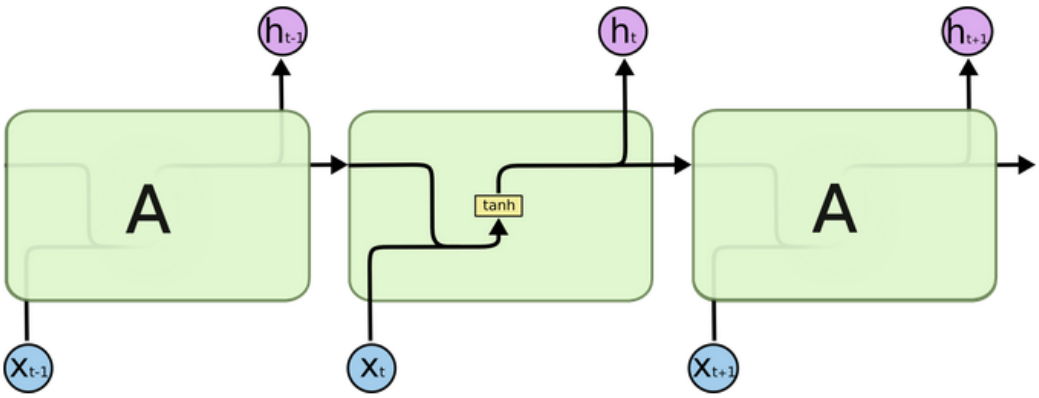


Figure 3-1: The repeating module in an RNN contains a single layer which contributes to the vanishing gradient problem. source : Colah's blog

to be able to learn them. This problem was explored in depth by Hochreiter (1991) [German] and Bengio, et al. (1994), who found some fundamental reasons why it might be difficult : the vanishing/exploding gradient problem. At each time step, the same weight is used to evaluate y_t. During backpropagation, the same weight is multiplied for calculating the gradient. So the further the network moves back in time, the bigger or smaller the gradient becomes, depending on the value.
To solve this, LSTMs[15] were introduced by Sepp et al. (1997). They introduce 'forget valves' and 'memory valves' to control the flow of information from one cell to another. Consider the structure of an LSTM cell given in Figure 3-2. Each cell takes 3 inputs : x_t is the input at timestep t, h_{t-1} is the output from the previous LSTM, C_{t-1} is the cell state of the previous LSTM cell. There are 2 outputs from each cell : C_t is the cell state at timestep t and h_t is the output at timestep t.

Obtaining the new cell state has 2 parts : the forget valve (represented by the cross in the top line) and the memory valve (represented by the plus sign in the top line). The forget valve is controlled by a simple neural network that takes as input C_{t-1}, h_{t-1}, x_t and a bias b_0. These are concatenated and passed through a sigmoid activation function. The output from the neural network is then multiplied element wise with the previous state. Since the sigmoid has outputs in the range 0 to 1, it acts as a filter for choosing retention of elements of the previous state. If the value is 1, the previous state element is retained as is; if the value is 0, the old value is completely erased.
The memory valve adds new information based on the current inputs provided. Again it is a simple neural network. The inputs to this nueral network are the same as in the previous case. The difference is that the biases change and the activation function changes to tanh. The element wise product of the tanh activation and sigmoid activation function is then added to the old memory value to get the new cell state C_t.
The output of the cell is derived from the new cell state and the output of the sigmoid activation. The new cell state is passed through a tanh activation and then multiplied

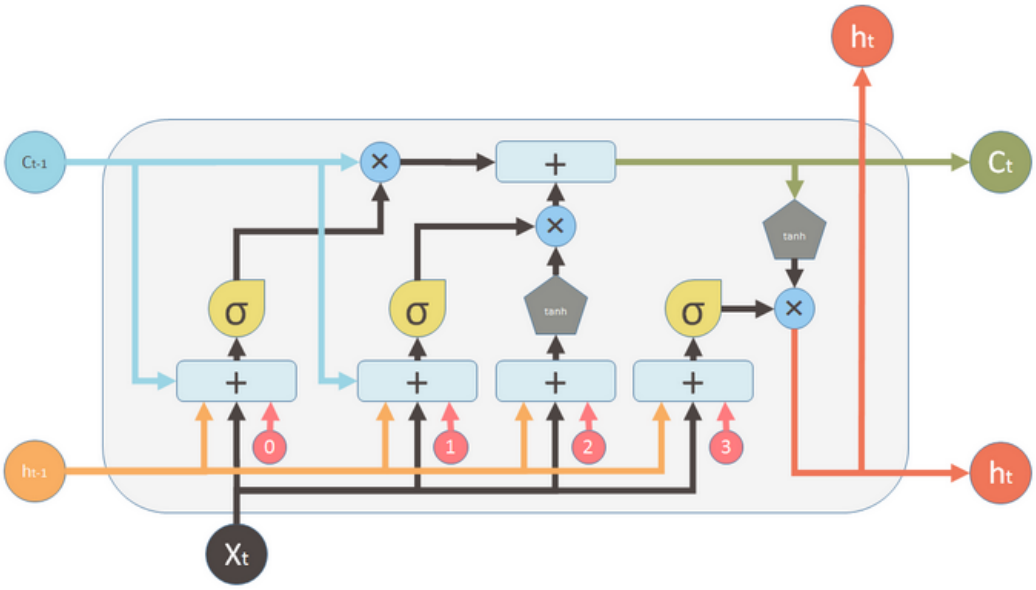


Figure 3-2: Each cell in an LSTM network (each cross represents element-wise product). source : Shi Yan's blog on Understanding LSTM and it's diagrams

element-wise to the output of the sigmoid activatoin. The result is the output of the cell at timestep t h_t.

$$
\begin{aligned}
f_t &= \sigma(W_f[C_{t-1}, h_{t-1}, x_t] + b_0) \\
i_t &= \sigma(W_i[C_{t-1}, h_{t-1}, x_t] + b_1) \\
\tilde{C}_t &= tanh(W_C[C_{t-1}, h_{t-1}, x_t] + b_2) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
\\
o_t &= \sigma(W_o[C_{t-1}, h_{t-1}, x_t] + b_2) \\
h_t &= o_t * tanh(C_t)
\end{aligned}
\tag{3.2}
$$

3.2 Attention

Even though LSTMs were able to fix the vanishing gradient problems in RNNs, they still had a major dependency. They could not effectively model long range dependencies. The reason is because the context information is a vector of fixed length. Regardless of the length of the sentence, the vector size is unchanged. To alleviate this problem, the attention mechanism[3] was proposed by Bahdanau et al. 2014.

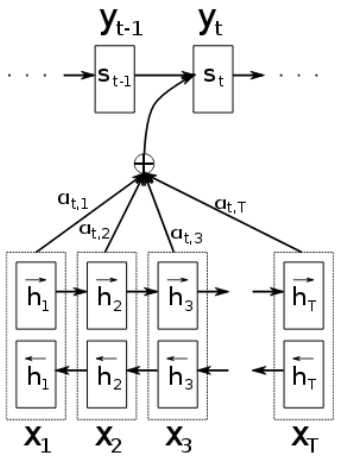


Figure 3-3: Generating the t-th target word y_t given the input sequence $(x_1, x_2, ..., x_T)$[3]

The authors propose soft-searching for a set of positions in a source sentence where the most relevant information is present. This is presented as an extension to the encoder-decoder model which learns to align and translate jointly.
The departure from earlier works is that it does not create a fixed length context vector. Instead, it encodes the input sequence into a sequence of vectors and chooses a subset of the vectors appropriately.
Mathematically, in encoders in normal recurrent networks with input $(x_1, x_2, .., x_T)$

$$h_t = LSTM(x_t, h_{t-1})$$

$$c = h_T$$

For decoders, the product of ordered conditionals is maximised

$$p = \Pi_{t=1}^{T} p(y_t|\{y_1, y_2, .., y_{t-1}\}, c) = \Pi_{t=1}^{T} g(y_{t-1}, h_t, c)$$

The attention mechanism changes the conditional probability to

$$p(y_t|\{y_1, y_2, .., y_{t-1}\}, x) = g(y_{t-1}, h_t, c_t)$$

Note that the conditional probability now depends on the initial sequence and the context for each word is now separate.

The context vector c_i depends on the sequence of outputs of the encoder at each position, ie, on $(h_1, h_2, .., h_T)$. Each h_i contains information about the context upto position i in the input sequence with a focus on the i^{th} word.
The context vector is calculated as a weighted sum of these annotations:

$$c_i = \sum_{j=1}^{T} \alpha_{ij} hj$$

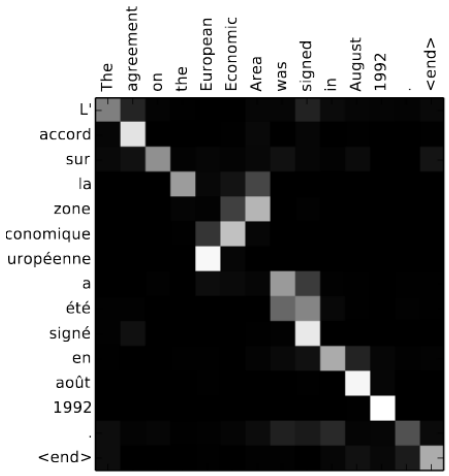


Figure 3-4: Attention weights for French-English translation. Pure white represents a weight of 1, while black represents 0[4]

where

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})}$$

Each e_{ij} is calculated by the alignment function $e_{ij} = a(s_{i-1}, h_j)$. The alignment function scores how well the inputs around position j match the output at position i. The authors parametrise the alignment model a as a feedforward neural network. The model outputs a soft alignment which allows for gradient propagation and learning a. α_{ij} represents the importance of annotation h_j with respect to the previous hidden state h_{i-1}. Thus the decoder decides which parts of the source sentence to pay attention to. In the traditional encoder-decoder model, all information was encoded into a fixed length vector so the decoder did not have any choice but to use that vector only. With the approach proposed by Bahdanau et al. 2014, the information can be spread out over the input sequence and the model can still perform well.
For getting the annotations h_j, the authors use a bidirectional RNN. The representations for left-to-right and right-to-left sequences are evaluated and the expressions are concatenated to get the final representation at position j in the input.
Figure 3-3 explains the decoder model at output position t. The outputs from the two LSTMs are concatenated and the weights α_{ij} are evaluated. The weighted sum of the annotations are then ued to find the output at position t

Figure 3-4 shows how the attention model is useful. In French, it is common for the noun order to be inverted in proper noun. In the given example, `zone economique europenne` is translated to `European Economic Zone`. The literal, word-by-word translation however would have given `Area Economic Europe`. By using attention, the model is successfully able to put more emphasis on `europenne` while predicting `Europe`. Note that this is possible also because of the fact that bidirectional LSTMs were used. Otherwise, the model would still have been able to model long range dependencies, but only in the backward direction.

3.3 Transformers and Self-attention

Vaswani et al. 2017[32] introduced a new architecture wherein they completely disposed of RNNs and used only attention mechanisms. This was partly due to the following reasons:

- RNNs are not parallelisable because they are autoregressive models, ie, they need to see inputs one at a time in sequential order.
- Long range dependencies are difficult to model in RNNs. Although this was solved by sequence-aligned states using attention mechanism, the problem of parallelisability still persists.
- Convolutional networks, on the other hand, are easy to parallelise and perform decently because they are able to model local dependencies.[18] However, path length between dependencies are logarithmic in distance, plus padding is required for text.
- By replacing RNNs with attention mechanism and encoding the position in the sequence, Transformers can be parallelised.
- Since attention is computed for every position pair, dependencies can be learnt in O(1) time.

To solve the problem of parallelization, Transformers use Convolutional Neural Networks together with attention models. Attention boosts the speed of how fast the model can translate from one sequence to another.

3.3.1 Architecture

Transformers are based on the encoder-decoder model and the autoregressive model. This means that the words are first encoded using some mechanism and as the words are decoded, they are also fed as inputs for further decoding. The actual model is given in Figure 3-5.
6 encoders are stacked to make the encoder part of the transformer. Similarly, 6 stacked decoders make up the decoder. The encoder has 2 sublayers, while the decoder introduces an additional sublayer. Both the encoder and decoder are made up of two components:

- Multi Head attention
- Position encoding and Position wise feedforward network

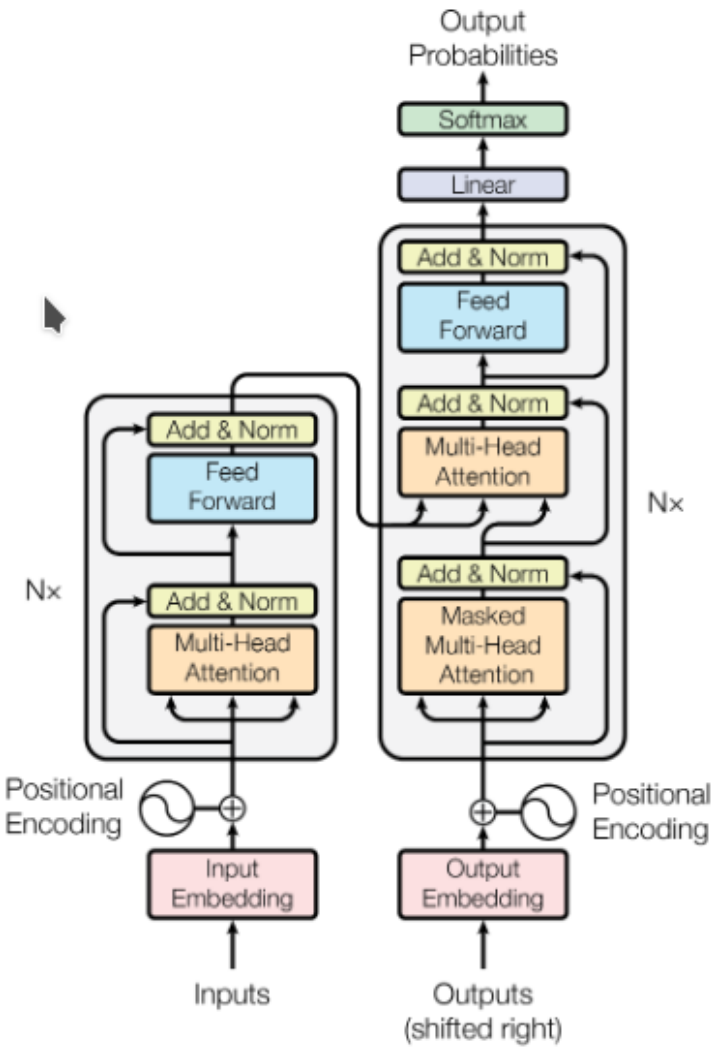


Figure 3-5: The transformer architecture consisting of the encoder and decoder[32]

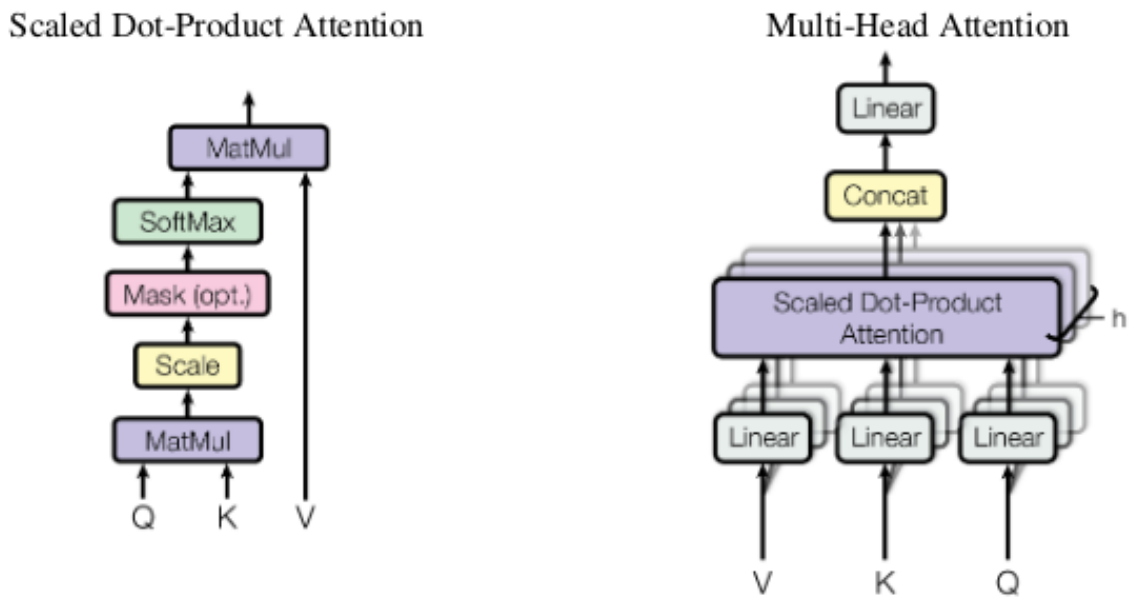


Figure 3-6: (left) Scaled dot product attention and (right) Multi head attention[32]

Multi Head attention

Firstly, the paper defines attention to be a function of 3 inputs : Q (query), K(key) and V(value). It takes a weighted sum of the values based on the weights calculated by a compatibility function between the query and key. Note that this definition fits into the attention introduced by Bahdanau et al. 2014. There, the key and value were the hidden states of the LSTM and the query was the input state of the decoder. Mathematically, attention can be defined as

$$S = \sum_{t=1}^{T} f(Q_t, K_t)V_t$$

where f is the compatibility function.
The paper uses a slightly different compatibility function than the one introduced in the original attention paper. They propose the use of a scaled dot product attention, which is a modification over the normal dot product attention:

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

where d_k is the dimension of Q and K.
Since in practice the queries and keys can be packed together into matrices, dot product attention can directly use matrix multiplication which makes it more efficient and open to optimisation.
The reasoning for scaling the dot poduct is as follows: for large number of dimensions, the dot product can grow arbitrarily large making the gradient almost 0. To counteract this, a scaling by $\sqrt{d_k}$ is done.

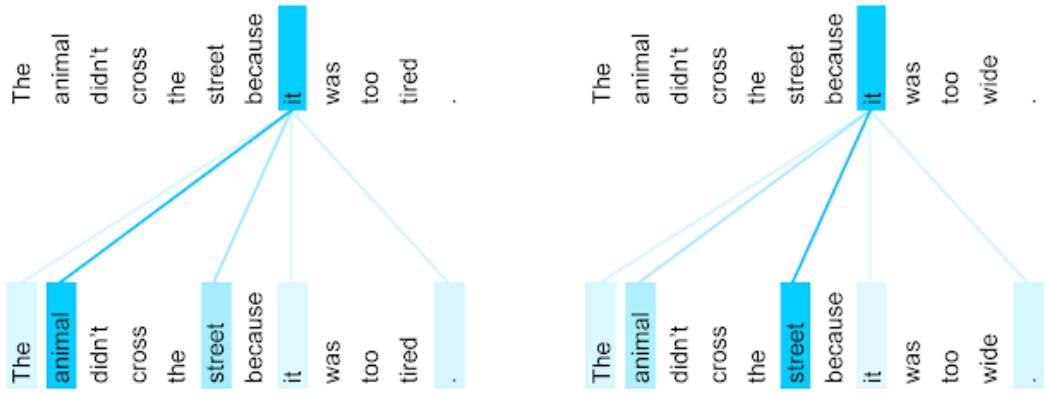


Figure 3-7: The encoder self-attention distribution for the word "it" from the 5th to the 6th layer of a Transformer trained on English to French translation (one of eight attention heads). Source : Google AI blog

Figure 3-6 highlights the Multi-head mechanism introduced in the Transformer architecture. Each input is transformed to a key, query and vector via projection to dimensions d_k, d_k and d_v. The scaled dot product attention is then evaluated. This is done across multiple attention networks. The outputs from these are concatenated and then fed through a linear feedforward layer.
The multi head attention essentially carries out self-attention, ie, attention of each input token with respect all other input tokens. This is done because of the following:

- Regular encoder-decoder architectures face the problem of long term dependencies.

- To rectify this, for every input word, the attention distribution with every word is learnt and a new representation for the input is calculated. This also mimics the standard encoder-decoder architecture where every position in the decoder can access information about every position in the input sequence.

- Using self-attention, at each level, each input representation to the decoder and in the encoder too has global information about every other token. This allows the representation at every position of the input to be modified according to the global information.

Figure 3-7 illustrates how self attention can bring a change in the input. Since the word `it` has high attention weight with the words `animal` and `street`, the word vector at the position of `it` will be dominated by the actual noun it represents. This is an example of how the transformer encoder architecture allowed the model to learn global dependencies and modify the input accordingly. Mathematically,

$$\text{MultiHead(Q,V,K)} = \text{Concat}(head_1, head_2, .., head_h)W^O$$
$$\text{where } head_i = Attention(QW_i^Q, VW_i^V, KW_i^K) \tag{3.3}$$

One thing to note, however, is that to maintain the autoregressive property of transformers while training, during decoding, some values need to be masked. This is presented in Figure 3-6 where an optional mask layer is present in the self-attention layer.

Positional embeddings

Since no recurrence or convolution is present, the model needs to use some other way to keep track of the relative ordering of words in the sequence. The authors add positional embeddings to each input word embedding. The positional embeddings created have the same dimension as the embedding dimension d_{model}. The primary embedding used in the paper is

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

where pos is the position and $2i$ is the dimension. The primary hypothesis for choosing this embedding was because it was thought that it would enable easier attention between relative positions since for any pos, PE_{pos+k} is a linear function of PE_{pos}. The authors also tried using learnt positional embeddings instead of fixed embeddings. The results in both cases were almost the same.

Residuals

Each sublayer in the encoder as well as the decoder has a residual connection before layer normalisatoin ie $LayerNorm(x + sublayer(x))$. This helps preserve the positional embeddings from the input to the output. The authors tried running the model without the residual connections and the performance of the model degraded by a large margin.

3.3.2 Position wise feed forward neural networks

Each layer of encoder and decoder contains a fully connected feed forward network, which is applied to each position separately and identically. The feedforward network is made up of 2 linear activation layers with a ReLU between them.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

It is worthwhile to note that the linear transformations are the same for different positions given an encoder/decoder stack. The input and output dimensions are the

same, d_{model}.

3.3.3 Running the model

Encoder:

- Generate initial input from the input words by concatenating the input embedding and positional embedding
- Use self attention head and feedforward networks to create new representation at each position in the input

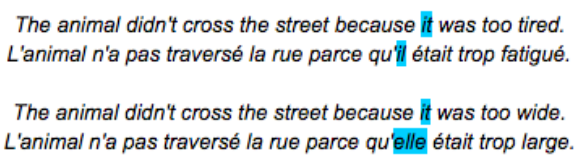
Decoder:

- Decoder input is the output embedding + positional encoding which is offset by 1 to ensure that prediction for position i depends only on words before i
- After passing through the masked multi-head attention, the attention value with the output of the encoder is calculated
- This is passed through a feedforward network for each position of the output
- A linear transformation converts the decoder output into a logits vector
- Softmax over the logits vector is calculated and the word associated with it is decided as the output at that time step

However, when we train the Transformer, we want to process all the sentences at the same time. When we do this, if we give the decoder access to the entire target sentence, the model can just repeat the target sentence. This is where the masked multi-head attention is useful. During testing on real world data, the model does not have access to the output beforehand so no masking is required.

3.3.4 Advantages

The self-attention mechanism allows Transformers to perform better than previous models on machine translation tass. By using it, co-reference resolution can be easily done and translation can be performed more accurately.



Self attention mechanism at work. Source: Google AI blog

Since the attention weights in Figure 3-7 correctly resolve the pronoun, the translation is also correct. French pronouns depend on the gender of the noun it refers to so in earlier works, this used to be a challenge.

3.3.5 Limitations

Despite the improvements that the Transformer architecture introduced, it has some limitations too:

- Attention can only deal with fixed-length text strings. The text has to be split into a certain number of segments or chunks before being fed into the system as input
- This chunking of text causes context fragmentation. For example, if a sentence is split from the middle, then a significant amount of context is lost. In other words, the text is split without respecting the sentence or any other semantic boundary

Chapter 4

Morphological tools

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

4.1 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.

4.2 Linguistics based

Statistical tools to find the lemma were the standard practice until the resurgence of neural networks in NLP since 2013. Most of them were based on rules written by linguistic experts, which required several man hours for careful construction. To alleviate this problem, several methods were imposed which tried to find structure in words with minimal human intervention. Here we outline one such approach which was based on tries[5].

Input words are used to construct tries. A word could be extracted from the trie by going down the root to the leaf and concatenating the characters found in

the way. For finding the lemma, the word and the backtracking levels are specified. Then the trie is traversed either to the leaf or till the first point of mismatch. In the former case, the model outputs all the words after backtracking along the tree for the specified number of levels.

This method attained more than 80% precision on 18 Indian and 5 European languages with minimal human intervention. This result was obtained after applying some heuristics to filter out the backtracking results.

4.3 Neural network based models

An intermediate step towards neural networks was taken when systems started incorporating perceptron networks[16] to predict the inflection pattern used in a word from several different sources. Although this was a first step towards utilising neural networks, the popularity of LSTMs and encoder-decoder architectures became prominent later when it set new state-of-the-art for several morphological tasks. Encoder decoder architectures are, at the heart, LSTM models in which one model is responsible for producing the encoding of the input. The decoder then takes this as input, and using some sort of start token, the model begins giving the output.

Chapter 5

Word Embeddings for Indian Languages

5.1 Prior resources available

India has 22 constitutionally recognised languages with a combined speaker base of over 1 billion people. Although the speaker base is quite large, the quantity of resource is poor. The demand for NLP tools for Indian languages is expected to increase as more and more Indian language users join the Internet. Indian language enabled phones is also increasing the demand and supply for original content in these languages. In this case, naturally, the demand for NLP tools which work on Indian languages also increases. The basic tasks for which they are required include parsing, document matching, document summary, inter-language translation etc.
Monolingual corpora for Indian languages is available, although not as exhaustive as English. Some of the sources are :

- Wikipedia dump
- CIIL corpus
- ILCI corpus
- CommonCrawl corpus

Although there are one or two more sources for the corpus, it is still not enough to build quality word embeddings. For comparison, English word embedding models are trained on corpora which contain a billion words. On the other hand, for Hindi, which is the most prevalent Indian language, the corpus size barely touches 100 million.

5.2 Evaluating embeddings in the Indian context

Given the lack of data, the embeddings can still be created. Their quality however cannot be measured in the same way as English word embedding models are evaluated. Some possible evaluation metrics for Indian language word embedding models 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 $\boldsymbol{vec}(B) - \boldsymbol{vec}(A) + \boldsymbol{vec}(C)$ is closest to $\boldsymbol{vec}(D)$ or not. Again, this is useful to determine if the word embeddings can capture the semantics of the language

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

- Next word prediction: Given the first $(n-1)$ words of a sentence, can the n-th word be predicted?

Indian languages have unique properites which make it different from English:

- Indian languages are morphologically rich. This increases the vocabulary size and reduces the number of observed instances of a given token in the corpus. This can impact the quality of the word embeddings.

- Many Indian languages are unlikely to have large monolingual corpora. However, these low-resource languages are likely derivatives of some high-resource languages. It is worth investigating if the word embeddings of the low resource languages can be improved using the corpora/embeddings of high resource languages.

5.3 Data available

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

As can be seen, compared to resources for English, the training data available for Indian languages is quite low. Even among the Indian languages, there is a sharp disparity in the resources available. Hindi has around 50 times the resources of its nearest competitor. Bengali and Tamil, on the other hand, still have more than a million sentences in the resources. After that, the size of training data available steadily decreases as we go down the table.

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 underrepresented in the research community and which could benefit from research.

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

Chapter 6

Word embedding of Indian Languages applied to POS tagging

6.1 Non-contextual embeddings

Based on the corpus that we have, embeddings were created for Indian languages of different dimensions and based on different models. These embeddings were evaluated over PoS tagging task on the Flair treebank corpus which has datasets for Hindi, Telugu, Marathi and Tamil. Marathi results have been excluded from this section because it has only 4 tags so the accuracy and F1 score was always 1. The Flair treebank corpus uses a CRF to determine the PoS tags using the embeddings provided by the user. So indirectly, it can tell us about the quality of our embeddings.

6.1.1 GloVe

We created GloVe embeddings of dimensions 50, 100, 200, and 300. Words with frequency less than 2 were not included in the library. The symmetric window size for the co-occurrence matrix is 15.

Table 6.1: GloVE models and their result on UPoS tagging over the Flair treebank

Language	Dimension	Accuracy	F1 score
ta	50	0.5786	0.733
ta	100	0.606	0.7547
ta	200	0.5995	0.7496
ta	300	0.5976	0.7481
te	50	0.6905	0.8169
te	100	0.7167	0.835
te	200	0.729	0.8433
te	300	0.7249	0.8405

Table 6.2: GloVE models and their result on XPoS tagging over the Flair treebank

Language	Dimension	Accuracy	F1 score
ta	50	0.4022	0.5737
ta	100	0.4366	0.6078
ta	200	0.4788	0.6476
ta	300	0.4827	0.6511
te	50	0.6709	0.8031
te	100	0.7208	0.8377
te	200	0.7025	0.8252
te	300	0.7269	0.8419

6.1.2 Word2vec and Fasttext

Word2Vec embeddings of dimensions50, 100, 200, 300 for both skip-gram and CBOW architectures were created using the gensim library[29] implementation of Word2Vec. 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.

Figure 6-1 shows the variation of accuracy on XPoS tagging with number of dimensions for different languages and models. It can be seen that the accuracy of the models start saturating at 200 dimensions and that the loss is minimum for fasttext models with embedding dimension 300. In fact, this trend was observed for all languages that we could test on: fasttext models consistently outperformed skipgram and CBOW models for all dimensions.

6.2 Contextual embeddings

6.2.1 ELMo

We train ELMo embeddings [25] 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 8.3 shows the perplexity scores of the ELMo models trained by us.

Table 6.3: skipgram and fasttext models and their result on UPoS tagging over the Flair treebank

Language	Dimension	Model	Accuracy	F1 score
ta	50	sg	0.5742	0.7295
ta	50	fasttext	0.6317	0.7743
ta	100	sg	0.6411	0.7813
ta	100	fasttext	0.7258	0.8411
ta	200	sg	0.6582	0.7939
ta	200	fasttext	0.768	0.8688
ta	300	sg	0.6672	0.8004
ta	300	fasttext	0.7517	0.8582
te	50	sg	0.6671	0.8003
te	50	fasttext	0.7521	0.8585
te	100	sg	0.7437	0.853
te	100	fasttext	0.807	0.8932
te	200	sg	0.7715	0.871
te	200	fasttext	0.8346	0.9098
te	300	sg	0.7802	0.8766
te	300	fasttext	0.8511	0.9196
hi	50	sg	0.9045	0.9499
hi	50	fasttext	0.9173	0.9568
hi	100	sg	0.9318	0.9647
hi	100	fasttext	0.9435	0.9709
hi	200	sg	0.9375	0.9678
hi	200	fasttext	0.9438	0.9711
hi	300	sg	0.9396	0.9688
hi	300	fasttext	0.9524	0.9756

Table 6.4: skipgram and fasttext models and their result on XPoS tagging over the Flair treebank

Language	Dimension	Model	Accuracy	F1 score
ta	50	sg	0.4263	0.5978
ta	50	cbow	0.4518	0.6224
ta	50	fasttext	0.4821	0.6506
ta	100	sg	0.4816	0.6501
ta	100	cbow	0.4955	0.6626
ta	100	fasttext	0.6008	0.7506
ta	200	sg	0.5011	0.6677
ta	200	cbow	0.5265	0.6898
ta	200	fasttext	0.6589	0.7944
ta	300	sg	0.5461	0.7064
ta	300	cbow	0.5359	0.6978
ta	300	fasttext	0.6637	0.7979
te	50	sg	0.6807	0.81
te	50	cbow	0.765	0.8669
te	50	fasttext	0.7437	0.853
te	100	sg	0.7437	0.853
te	100	cbow	0.7543	0.8599
te	100	fasttext	0.8161	0.8988
te	200	sg	0.7693	0.8696
te	200	cbow	0.7693	0.8696
te	200	fasttext	0.8323	0.9085
te	300	sg	0.7891	0.8821
te	300	cbow	0.7935	0.8849
te	300	fasttext	0.8511	0.9196
hi	50	sg	0.8904	0.942
hi	50	cbow	0.8926	0.9432
hi	50	fasttext	0.9124	0.9542
hi	100	sg	0.9142	0.9552
hi	100	cbow	0.913	0.9545
hi	100	fasttext	0.9347	0.9662
hi	200	sg	0.9254	0.9612
hi	200	cbow	0.9226	0.9597
hi	200	fasttext	0.9381	0.9681
hi	300	sg	0.9325	0.9651
hi	300	cbow	0.9235	0.9602
hi	300	fasttext	0.9395	0.9688

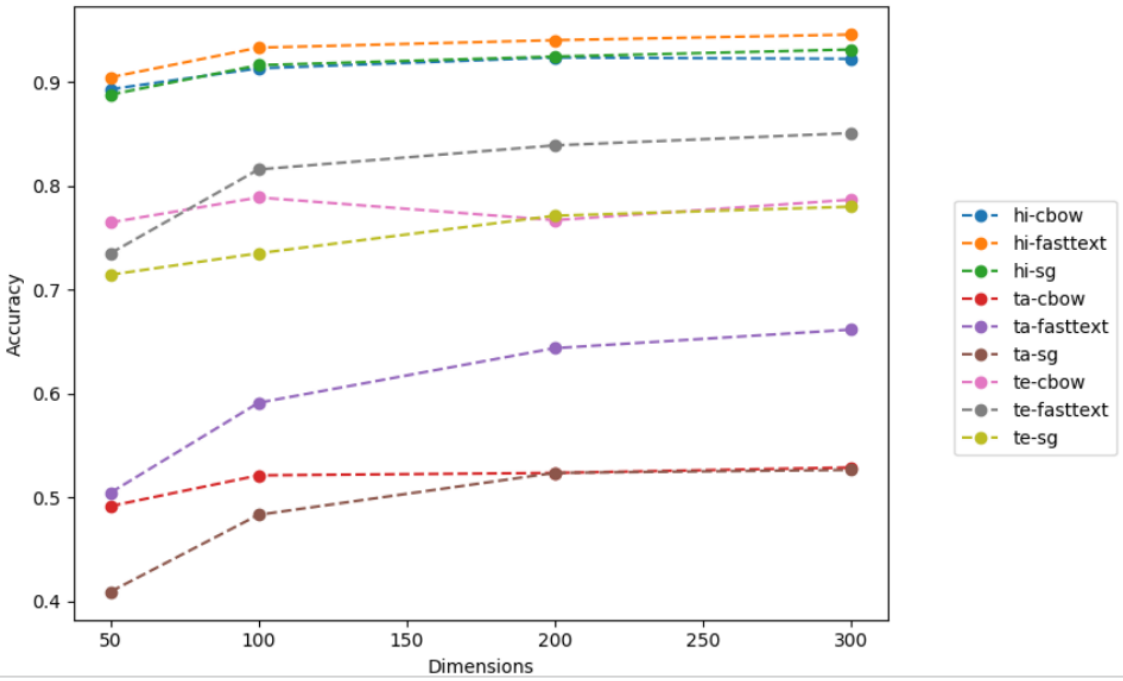


Figure 6-1: Variation of accuracy on PoS tagging with number of dimensions for different languages and models

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

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

Table 6.5: ELMo prerplexity scores

6.2.2 BERT

We train BERT (Bidirectional Encoder Representations from Transformers) [11] embeddings of 300 dimensions. Since BERT can be used to train a single multilingual 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 [13] 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.

Chapter 7

Morphological tasks for Indian languages

Most of the work on morphological analysis in Indian languages have been done using rule based systems [31] [27], which require linguistic experts to model them. Even with the idea of cross training language models [28], people still relied on manually annotated rules for using the knowledge from a high resource language in a low resource setting.

With word2vec [22], 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[5][34][30], 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.[10][26]

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.

7.1 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.

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

Table 7.1: Data available for different languages in UD Treebank. 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.

Chapter 8

Using embeddings for lemmatization

Working in a low resource setting, especially one in which the number of sentences does not exceed 1K in some cases, requires artificial data generation or completely unsupervised methods for any type of training to be possible. Building on the work done in the last chapter, the aim of these experiments was to determine if embeddings could be used as a viable approach to solving the task of lemmatization for Indian languages.

8.1 Vanilla approach

From the embeddings, we chose to start with Word2Vec and Fasttext because of the ease of preparation of embeddings with said models as well as the lower complexity. We choose the models with the best performance, ie, the models which embed the words in 300 dimensions.

As an initial experiment, we tried to find the closest word to each query word and see if the lemma was present there. The logic for that was as follows: embeddings are supposed to capture the semantic and syntactic meaning of the words. This means that the related words are supposed to present close to each other in the embedding space. The relation between embedding vectors of related words was also shown by Mikolov et al. 2013 [22] when he showed that the closest word to $\boldsymbol{vec}(king) - \boldsymbol{vec}(man) + \boldsymbol{vec}(woman)$ was the vector embedding of $queen$($\boldsymbol{vec}$() is a function that maps the word to its embedding in the embedding space). Since an inflected words and its lemma are related to each other semantically, they should be present close to each other. Also, since the words are also related syntactically in most cases, barring irregular inflections, it makes more sense that the inflected word-lemma pair would be close in the embedding space. An example of the top 5 closest words generated from this method is shown in table 8.1 The three rows correspond to the results from 3 different models, namely Skipgram, Fasttext and CBoW. It can be seen that the root word "बदबू" is present in the top 5 list for each model, which signifies that our intuition was somewhat right. It can also be observed that the Word2Vec

Skipgram	Fasttext	CBoW
बदबू	बदबूदा	बदबू
दुर्गन्धपूर्ण	बदबूदारों	बजबजाती
गंदा	हूँबदबूदार	गंदे
गंदगी	बदबूआ	दुर्गन्ध
दुर्गंध	बदबू	गंदा

Table 8.1: Closest word for "बदबू" using Word2Vec and Fasttext embeddings

models tend to give words which are semantically closer and not syntactically. This makes sense because Fasttext incorporates subword level information, which means that the inflected form of the word has a high similarity with the root word. This is also evident in the average string similarity between the results for the 3 models: Fasttext has an average of 0.56 similarity with the root word, followed by CBoW at 0.38 and SkipGram at 0.39. These metrics are evaluated using the complement of standard Levenshtein distance as a proportion of the length of the original word. If we consider the average over the best scores, Fasttext still performs better than other models. though not by a huge margin. The scores are 0.77, 0.71 and 0.72 using the same metric.

Another example given in Table 8.2, which represents the closest words for the query word "आत्मविश्वास" hint at the possible limitations of this model. In this case, Skipgram was not able to get the root word in the list at all, while the other two managed to get it. Also, a more detailed look at the two tables show a pattern in the results given by Fasttext: all the outputs are the various inflections of the query word. This is both an advantage and disadvantage. This pattern could be advantageous because the model is almost guaranteed to put syntactically as well as semantically close words closer to the query word. This means that a possible higher accuracy. On the other hand, for words with too many inflections or variants, this could backfire. This might happen because the model might put all the inflected forms/variations at equal distance from one another.

As predicted, the model did not perform well on all words. 100 words were randomly chosen from the dataset and the top 10 words for each of them were listed. On manual comparison, it was found that less than 10% of the times, the root word actually came in the top 10 closest list for that query word.

Skipgram	Fasttext	CBoW
आत्मबल	त्मविश्वास	आत्मबल
आत्मविष्वास	अत्मविश्वास	उत्साह
आत्मविश्	संबल:आत्मविश्वास	आत्मविश्वाश
कॉन्फिडेंस	आत्मविश्वाश	आत्मविष्वास
उत्साह	आत्मविश्वासदस	कॉन्फिडेंस

Table 8.2: Closest word for "आत्मविश्वास" using Word2Vec and Fasttext embeddings

8.2 Further testing and heuristics

Based on the results, Fasttext was chosen as the preferred model. The average similarit and best similarity show that this model outperformed the others.

To further improve upon the results, several heuristics were applied. Taking inspiration from previous research on lemmatizers[5], we decided to use post processing in the output to get better results. One heuristic that comes to mind is the string similarity between the inflected word and the lemma. Except for the case of irregular verbs, most inflections are formed from the root word via the addition of a suffix and a prefix. For example, "लड़का" and "लड़के" have a majority of letters common in between them. An example of an irregular inflection in Hindi, on the other hand, would be "जा" and "गया" .

To utilise this, we enforced a 70% string similarity between the words before considering it in the output. This did not change the results by much, suggesting that maybe the threshold was too high or that the problem lay elsewhere. Dropping the threshold did not help either.

Another heuristic that was a possibility was filtering based on length of the outputs. In most cases, the lemma is the shortest word among all its inflected forms. This again occurs mostly due to the fact that inflections are formed by the modification and addition of letters to the lemma at the start or end. Most of the exceptions to this rule can again be found in verbs which modify in a non standard way. Based on the tables presented in this section so far, the filtering could weed out some unwanted candidates.

For this condition we imposed that the length of the output should be at most as long as the query word. Although it did filter out many candidates across query words, the accuracy did not go up. To identify the cause behind it, we looked more carefully at the results. The inspection revealed that even though the filter removed

many words, other, semantically close words instead of the lemma replaced the ones removed.

Combining the two methods also managed to raise the accuracy by only a few percentage points, but still hovering around the 10% accuracy mark. An interesting observation, however, was made due to all the detailed inspections: for some words, the closest words were very close and for others, they were far away. The latter occurred mostly in the case of words which did not appear as frequently in the corpus. Foe example, the word "आत्मविश्वास" , along with its inflections are not common enough as "लड़का" and its variants. Due to this reason, any heuristic based on the distance between the root word and query word were discarded.

Although Fasttext did give better results compared to Word2Vec and CBoW, the outputs were still not sufficiently good. The root words were still mostly present in the 5th-10th posiiton of the list of closeset words by the query word. A better and a contextual model might give good better results because of better quality embeddings

8.3 ELMo

ELMo uses BiLSTM networks, which means it encodes more information than word2vec or Fasttext. Additionally, it uses character convolutions, which means it has the ability to process Out of Vocabulary(OOV) words. This is useful in scenario where the resources are limited as there is no guarantee that the lemma might be present in the corpus.

The difference between ELMo and the previous models was clear in the words present close to 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. For example, the average cosine distance in the list of ten closest words was 0.24, whereas in the case of Fasttext it was 0.33. Table 8.3 shows the closest words along with the cosine distance from the query when using ELMo embeddings.

It should be noted that the distance to the inflected forms is lesser than the distance to the root word. To increase the top 3 or even top 1 accuracy, the distances need to be changed, which suggests that some post processing was required.

Additionally, another problem that was faced was that the query word itself could be the lemma. If the word itself were to be included in the list of closest words, it would always rank first because the distance of any word to itself is the minimum possible distance. The following methods considered this as a potential issue and therefore tried to address it.

8.4 Post processing outputs

This section lists out several of the post-processing methods that we tried for improving the accuracy along with the rationale behind each of them. None of the ap-

Corpus word	Predicted root	Cosine similarity
मसूढ़ों	मसूढ़ा	0.28911465406417847
	मसूढ़ो	0.19095295667648315
	मसूढों	0.08356654644012451
	सूढ़ों	0.20856153964996338
	मसूढ़े	0.2144671082496643
	मसूढ़ा	0.35028088092803955
	मसूढॊ	0.4301416277885437
	नसूढ़ों	0.19788700342178345

Table 8.3: Top 9 closest words (we excluded the word itself) for "मसूढ़ों" and their distance from "मसूढ़ों"

proaches mentioned in this section (except the one which uses substrings) improved the accuracy of the model.

The first one was taking the average of the top 10 closest words and then finding the word closest to the resulting embedding. The assumption behind this was that each lemma had a cluster centered around it. For example, if the language was English, the words *played*, *playing*, *plays* would all be equidistant from *play*, the root word. But on visualising the embeddings in a 3D space using t-SNE, it was clear that this was not the case. Additionally, another issue that exacerbated this problem was the fact that all the inflections for a word were not present in the corpus. For example, multiple plural inflections of a word would be present, thereby skewing the cluster towards one side. Also, the quality of embeddings are not always perfect which lend another set of inaccuracies to this experiment.

The second method was based on the assumption that the root word would be present in the list of closest words for all its inflections. The methodology followed is explained via the following example: let Q be the query word and $c_1, c_2, .., c_{10}$ the list of closest words for Q. Now, the list of closest words for all c_i is found out and the list is aggregated. The mode among this list is chosen as the root of the word. One of the issues faced via this method was that the c_i were not all inflections of the root word. This led to many other words being included in the aggregated list. Additionally, the c_i which were inflections of the root word were also present in the list of closest word for each c_i. This meant that a lot of words would have the same frequency in the aggregated list and thus there would be no mode. Heuristics such as filtering based on length could be applied, but it would have minimal effect because most of the inflected words are of the same length.

The last method produced some good results because it handled some of the issues of data sparsity. For each query word, the same steps are performed, but in this case, substrings of the query word are added as possible lemmas for the word. In most of the inflected words, the root words are a substring of the inflected word. If the substrings were taken into consideration too, the model could potentially identify the substrings as lemmas even though they did not appear in the corpus at all or appeared very frequently.

8.5 Vector arithmetic

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:

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

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%. Some of the sample outputs using this method are given in Table 8.4.

Looking at the results of this model, it was observed that the model performs the best for nouns among verbs, adjectives, nouns and adverbs. Adverbs are not included in this result because the total number of adverbs in the corpus was very less. We hypothesise that the model performs the best for nouns because nouns have lesser inflections than verbs. For confirming this hypothesis, the distance of an inflected noun with its root and the distance of an inflected verb with its verb was compared. The difference was of an order of magnitude 10. Nouns tend to be closer to their lemmas with a distance less than 0.5 on average. On the other hand, the difference between an inflected verb and its root lies around 7.00-8.00. This is attributed mostly to the non-presence of the lemma in its root form in the corpus. Verbs are always modified according to the subject or noun before being added in the sentence. This makes it harder for the model to learn good embeddings for the word. Even though the model can handle OOV words, without any context, the quality of embeddings deteriorate a bit.

Corpus word	Top Similar words
चैंबरों	[चैंबरो, चैंबर, चैंबरों] [चैंबरो, चैंबर, चेंबर]
हुक्मनामे	[हुक्मनामा, हुकुमनामा, हुक्मनामे] [हुक्मनामा, हुकुमनामा, वक्फनामा]
बुज़ुर्गों	[बुज़ुर्गों, बुज़ूर्गों, बुज़ुर्गों] [बुज़ुर्ग, बुज़ुर्ग, बुज़ुर्गिन]
रियायतें	[रियायत, रियायत, रियायते] [रियायत, रियायत, रियायते]
सौदे	[सौदा, सौदों, सौदाकारी] [सौदा, सौदें, समझौता]
सिपाहियों	[सिपाहियो, सिपाहीयों, सिपाहियों] [सिपाहियो, सिपाही, सिपाहिय]

Table 8.4: Using geometric methods to find the lemma of a given word. The words in the first and second brackets are the results are using (खजाने, खजाना) and (लोटेंगें, लोटना) as the known word-lemma pair respectively. Green words highlight the lemma of the query corpus word.

This model addresses some of the issues highlighted in the previous, while remaining essentially an unsupervised model. The problem where identifying the lemma when the query was the lemma itself was essentially a difficult task, unless the model could learn to differentiate between lemmas and non-lemmas without any training data. This method on the other hand, does not need to learn anything of the sort. Although there is reason to believe that adding the difference vector would move the resulting vector close to an inflected form, that is not the case. The accuracy remains as high as 73% for the classes where the query and the lemma are the same. On the other hand, the accuracy on the complement set is very low at 7.35%. Although this improves by a few percentage points by using heuristics such as length filtering, adding substrings, it does not address the main drawbacks of this model.

8.6 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 9

Efficacy of LSTM and seq2seq models

Most recently, people have started using seq2seq models for lemmatization and other morphological tasks. However, these models requires a proper dataset before the model can learn patterns effectively. This is why the approach using embeddings was considered first.

Anastasopoulos and Neubig, 2019 [1]showed that data augmentation in NLP, specifically for the lemmatization task, was possible. Implementing this data and utilising it could prove helpful for lemmatization in Indian languages.

This also helps address the multiple issues present in the embeddings-based approach: since we are augmenting data, the problem of data scarcity is almost negligible. It also handles the issue of false negatives and false positive because the model is now directly giving an output instead of learning things from a campus.

The best result from using the embeddings stand at 53%, which has a lot of scope to be improved upon. To work within the confines of our restraint of low resources, we need to make a lightweight model or augment the training data.

9.1 Decoder

ELMo embeddings already encode useful information in the embedding space. To utilise this information, the encoder from the standard encder-decoder model was stripped off. By using the ELMo embeddings as context and state input to the decoder, the intuition was that the decoder could extract out the PoS tag information and use it to output the lemma. Specifically, the 1024 dimensional embeddings were projected into two different latent spaces, each with dimension 100. These were fed to the decoder as the hidden state and cell state respectively. The model was given a `<START>` token as input and the output of the LSTM at time step t was fed into a fully connected network. The loss model used was a logits loss over the softmax of the output of the FCN.

The model failed to learn any useful dependencies, with less than 20% accuracy on the test file provided by the UD treebank. For this model, no data augmentation

was used because the number of training tokens for Hindi is of a respectable size (357K tokens). We suspect that this is due to the high dimensionality of the ELMo embeddings and the mismatch between the expected inputs to the LSTM and the actual inputs received.

9.2 Seq2seq

We moved to the standard encoder-decoder architecture in seq2seq models after experimenting only with the decoder. The encoder and decoder were vanilla LSTMs. The input to the encoder was the inflected form of the lemma with a start token `<START>` prepended to it and `<END>` appended towards the end. The inputs are padded with blank tokens to make the inputs of uniform size in a batch. As in Anastasopoulos and Neubig, 2019 [1], we used 10 as the batch size and ran the model for 100 epochs. The latent dimension used in the embeddings was 100. The loss model used was a cross entropy loss.

Since the models were trained on Hindi only, there was no need for augmenting the data. On a training dataset of size 375K and test dataset of size 20K, the model achieved 64.91% accuracy. There were a few modifications which were tried out, and are mentioned in the following paragraphs.

Since most of the words contain only 3-6 characters, and the input size is 15, a lot of padding takes place in most cases. This reduces the training loss and makes it look like the model is performing well, when it is not. Additionally, the greater size of input means more resources need to provided for the model to learn effectively. To mitigate this issue, we proposed clipping the words at 8 characters. This reduced the padding by half and also reduced the input size. In this case, the training loss was a better indicator of the actual performance of the model. We experimented with different clip lengths and found 8-10 to be the optimum clip length, without any significant drop in accuracy and a significant speedup in time for convergence in training.

Additionally, several methods like including a warm up phase and cooldown were also tested, but the accuracy decreased on using those models by 5-10 percentage points. The idea behind warm up phase was that via the warmup phase, the model would learn to imitate the input in the output. Since some letters between the lemma and inflected word remain invariant, the warmup phase would allow the model to model that part in the output. Several lemmatisation models [1] have benefited from this method.

All the results for seq2seq models are collated in Table 9.1

9.3 Transfer training

For some languages, there is considerable restriction on the amount of data available. To combat this issue, ideas for cross lingual[28] have proposed in several avenues in NLP and have shown to be useful[14][8]. The idea behind this is that languages of the

	Seq2Seq	Seq2Seq with attention	Cross lingual models (Source lang: Hindi)
Hindi	65	**91.94**	-
Bengali	<10	**36**	23
Marathi	60	**85**	63
Tamil	53	**76**	62

Table 9.1: Accuracies(in %) on the test set for different models

same origin share many similarities which can be 'transferred' from one language to another. An example would be the family of Indian languages which have all derived from Sanskrit. The resemblance between Sanskrit and Hindi, for example, is very high and looks like it might benefit from transfer learning across languages.

Several tasks [21], have started coming up which focus on improving the accuracy of models of languages with less resources using high resource langauges. We use the dataset of Bengali from SIGMORPHON 2019 shared task [21] as the low resource dataset. The training and testing set for both of these have only 100 tokens. On training the encoder and decoder directly, the model is prone to overfitting and does not give a good accuracy. Using the data augmentation technique mentioned in Anastasopoulos and Neubig, 2019 [1], we increase the training data from 100 words to 10K words. The basic idea of the technique is that using alignment models, the unchanging part of the word and lemma is identified. They are then randomly replaced with any combination of strings to get artificial training data for the model.

For transfer training, we first train the model on the source language and then fine-tune it on the transfer language. The best models from the source language are used for fine-tuning on the transfer language. The results of our model on different languages can be seen in Table 9.1.

Using transfer learning, the accuracy achieved on the Bengali training set was 23%. This is less compared to the accuracy in monolingual setting with augmented data, which gives an accuracy of 36%. The performance for Tamil and Marathi also drops by 14 and 22 percentage points respectively. On observing the outputs, we found that several inflection patterns present in the low resource language were not present in Hindi, leading to poorer performance in the cross-lingual setting. Additionally, the number of training samples might have not been varied enough to 'override' the weights in the network trained on Hindi.

9.4 Character convolutions

When going from an inflected word to its lemma, most of the transformations take place at the boundary of the affixes and the root word. The remaining part of the affixes are simply removed in the process of obtaining the lemma. This gave rise to the idea that encoding pairs of characters might give good results because the model could then perform better at word boundaries. Thus we decided on using character

convolutions [9]for the model.

Each word was first sent through the keras pre-processor which assigned numbers to each character. Then the numbers were converted to a categorical representation which were then projected into a latent space of 100 dimensions. Further, the embeddings were sent to a set of 6 convolutional filters with pooling layers and ReLU activation in between them. The final result was then flattened, passed through a fully connected network and fed into the encoder. The decoder part remains unchanged.

The models were trained using the same hyperparamters used in other sections. The performance of the model degraded sharply compared to the standard model. The performance hit was more obvious in lower resource languages, most likely because of the data. Convolutional networks are reported to give good results when trained on an abundance of data, which is around more than a million tokens.

9.5 Attention

Attention models [3] became popular because they resolved a long standing problem of LSTMs. Since the decoder was free to attend to any portion of the output from the encoder, the model could, in theory, pick out relevant information from the encoder and use it to give better results. Several technologies and progresses have been made since then, like the transformer [32] and BERT [11], which build upon attention.

We implemented Bahdanau attention in our models for the language. The outputs of the encoder were attended to before being sent as cell states in the model. This allowed the model to learn better cell and hidden state representations, which gave better accuracy. For Hindi, the model was able to achieve 91.9% accuracy overall on the UD treebank dataset, which is a big improvement over the 64.91% achieved by seq2seq models.

9.6 Analysis

Seq2seq models give an overall accuracy of 64.91% percent for the lemmatization task on Hindi. The percentage remains almost the same for the 2 classes of test words: query words which are already in root form and query words which have a non zero edit distance from their lemmas. The accuracies stand at 63.7% and 67.8% respectively. This shows that the seq2seq model is performing much better on the second class of words compared to the embedding based models.

If we look at the accuracies after removing duplicates, the data tells a different story. The overall accuracy drops to 39% and the accuracies in the 2 classes stand at 34% and 51%. This suggests that the seq2seq model learns how to change the endings very well. However. it fails to distinguish between the lemmas and non-lemmas in the query words.

On focusing on accuracies classified according to PoS tags, the model improves its performance on several PoS tags compared to the embedding based model. Nouns, verbs, adjectives have accuracies 96%, 84% and 96% respectively. Most notably, the

performance for verbs has increased, which is also reflected in the 51% accuracy stated in the previous paragraph. For adverbs, however, the accuracy dips to 39.8%. Apart from sentence conjunctions, adverbs and modifiers, the model consistently scores 60% or higher.

Attention models improve upon the seq2seq model in almost all aspects. The overall accuracy jumps to 91.9% as well as in the 2 categories (lemmas and inflected words as query words). Even if duplicates are removed, the model gives accuracies upwards of 83%, with the exception of one category (inflected words in the deduplicated case) where it scores 80%.

Even in the case of classification by PoS tags, the worst performance is for pronouns at 85%. For all the parts of speech, the accuracy is above 89%, rounded off to the nearest integer.

Chapter 10

Conclusion

As part of our BTP stage 1, we worked on getting a good understanding of the developments in word embedding models. After that, to get an idea of the scenario of word embeddings in the context of Indian languages, we started exploring the work done in the field for Indian languages. What we found was as of now according to the exploration done by us, there is no established baseline on NLP tasks for word embeddings on Indian languages. Most of the work present in academia focuses on a single task like PoS tagging in some cases and NER detection in others, but those too are scattered.
The first stage of our BTP culminated in a comprehensive database of word embedding models for over 14 Indian languages. The corpora for the 14 Indian languages were collected from Wikipedia dumps, ILCI corpus and manually annotated datasets. They were cleaned and processed before being used for training. The embedding models generated till now include word2vec (skipgram and CBOW), GloVE, fasttext, ELMo, BERT and XLM (cross-lingual models). The non-contextual models have been evaluated on PoS tagging tasks and their results presented in Chapter 6. These results were compiled across all models and languages and selected as a workshop paper in LREC 2020 titled "Pre-trained Word Embeddings for Indian Languages". This was the first collection of word embedding models along with datasets for low resource Indian languages.

The second stage of our work focused on establishing a baseline for the morpholog-

		Geometric	seq2seq	attention
Including Repetition	All	56.96	64.91	**91.94**
	Q==R	73.03	63.71	**92.87**
	Q!=R	11.2	67.89	**89.3**
Excluding Repetition	All	55.51	39.33	**90.38**
	Q==R	73.12	34.93	**93.96**
	Q!=R	7.35	51.63	**80.58**

Table 10.1: Comparison of all models on lemmatisation task

ical tasks in the Indian setting, especially lemmatization. The approaches considered were deciding keeping in mind the scarcity of resources for Indian languages. We started out with completely unsupervised methods, which built on the work done in the previous semester, using embeddings. After several iterations, we moved to specific models which are generally considered useful for these problems, namely seq2seq models. Again, after performing various experiments and improving the results up to 64%, we decided to implement attention based models. Although these models require annotated data, we modified the general procedure used to train these models so that they could become viable in our setting. We used transfer learning from a high resource language to a low resource one and also augmented the data. This way, even a set of sufficiently good 100 samples could be augmented to 50K samples and the results would still work.

Table 9.1 lists the different sequence to sequence models, with and without attention, that were trained by us. We can see that the attention models score better than all other models, even in the case of transfer learning. We empirically show that attention based monolingual learning is the most effective for all languages, despite data scarcity

	With repetition			Without repetition		
	Geometric	Seq2seq	Attention	Geometric	Seq2seq	Attention
PART	**90.91**	40.73	87.36	80	35.08	**91.49**
CCONJ	42.86	92.11	**94.78**	25	54.55	**91.29**
AUX	59.32	72.16	**92.86**	20	57.14	**91.67**
ADJ	70	**96.94**	95.35	69.57	80	**88.24**
SCONJ	66.67	28.85	**91.3**	50	20.78	**89.13**
VERB	11.11	84.21	**100**	7.41	70.42	**100**
ADV	75	39.79	**100**	75	31.72	**100**
ADP	50	78.25	**98.62**	72.73	63.16	**91.67**
DET	26.67	**93.53**	89.09	44.44	82.86	**88.74**
PROPN	66.2	97.3	**100**	66.67	85.51	**100**
NOUN	70.49	96.17	**100**	68.75	83.95	**100**
PRON	38.89	**96.62**	83.63	26.67	70	**85.11**

Table 10.2: Comparison of all models on lemmatisation task based on PoS tags

Tables 10.1 and 10.2 show the comparisons between the best performing models in each case. There are 3 types of models that we have considered: 1) models which use only embeddings and vector arithmetic to get the lemma (referred to as geomtric in the tables), 2) seq2seq models where the input is the inflected word and output is the lemma, and 3) attention based seq2seq models (referred to as attention in the tables).The first table shows the accuracy of all 3 models on the same dataset. The 'All' column shows results without any filtering. 'Q==R' displays the accuracy for those query words which are the lemma themselves. 'Q!=R' displays the accuracy for words where the lemma and the input query are not the same. The 'including repe-

tition' and 'excluding repetition' signify whether the (query, lemma) pair duplicates have been kept or removed respectively (the corpus contains a lot of repeated words so a lot of repetitions are present).

We perform a more exhaustive analysis using the seq2seq based methods which utilise attention. The work is still going on and we have submitted our paper for peer review. Briefly, the paper covers the following things:

- 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

Chapter 11

Future work

The work that we have done focuses on lemmatisation specifically, but several other avenues still remain unexplored. As an example, extracting morph tags from a given text is something that could be useful for the research community. Several online tasks, like SIGMORPHON already have these tests, but they cover only a handful of Indian languages. Devising a test, as well as solutions to the problem could go a long way in promoting research in the community. Since lemmatisation, in particular, is useful in downstream tasks and several NMT machines use lemmatizers to modify the sentence as a part of pre-processing, work in a direction similar to ours might benefit several other research works.

For our models, the concept of transfer learning has not been explored fully. Transfer learning is routinely used in computer vision tasks and it gives good results. Similarly, better methods of transfer learning might improve the performance for many low resource languages.

Alternatively, if the amount of annotated data required for training could be reduced further, several languages will benefit from it. Data augmentation and transfer learning are the first steps in this direction, but more work can be done in terms of lighter and less data intensive architectures.

Another method that has been shown to work is multilingual learning[2]. Instead of using only one language for learning purposes, if models use multiple languages as a source of information, the accuracy of the models can increase. The choice of language families would matter in this case too.

Bibliography

[1] 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.

[2] Antonios Anastasopoulos and Graham Neubig. Pushing the limits of low-resource morphological inflection. *arXiv preprint arXiv:1908.05838*, 2019.

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

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

[5] 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.

[6] 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.

[7] Axel Cleeremans, David Servan-Schreiber, and James L McClelland. Finite state automata and simple recurrent networks. *Neural computation*, 1(3):372–381, 1989.

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

[9] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781*, 2016.

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

[11] 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.

[12] David Ernest. Ambiguity in nlp, david ernest. `https://cs.nyu.edu/faculty/davise/ai/ambiguity.html`. Accessed: 2019-08-05.

[13] Google. Sentence piece embeddings. `https://github.com/google/sentencepiece`, 2018.

[14] Liane Guillou, Christian Hardmeier, Preslav Nakov, Sara Stymne, Jörg Tiedemann, Yannick Versley, Mauro Cettolo, Bonnie Webber, and Andrei Popescu-Belis. Findings of the 2016 wmt shared task on cross-lingual pronoun prediction. *arXiv preprint arXiv:1911.12091*, 2019.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Lstm can solve hard long time lag problems. In *Advances in neural information processing systems*, pages 473–479, 1997.

[16] Piotr Jędrzejowicz and Jakub Strychowski. A neural network based morphological analyser of the natural language. In *Intelligent Information Processing and Web Mining*, pages 199–208. Springer, 2005.

[17] Michael I Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Artificial neural networks: concept learning*, pages 112–127. acm, 1990.

[18] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.

[19] keitakurita. Xlnet explained. https://mlexplained.com/2019/06/30/paper-dissected-xlnet-generalized-autoregressive-pretraining-for-language-understanding-explained/. Accessed: 2019-11-11.

[20] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.

[21] 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.

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

[23] Barak A Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269, 1989.

[24] Jeffrey Pennington, Richard Socher, and Christopher D 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.

[25] 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.

[26] 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*, pages 193–207. Linköping University Electronic Press, 2018.

[27] 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.

[28] 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.

[29] 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`.

[30] Ricardo Rodrigues, Hugo Gonçalo Oliveira, and Paulo Gomes. Lemport: a high-accuracy cross-platform lemmatizer for portuguese. In *3rd Symposium on Languages, Applications and Technologies*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[31] 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.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[33] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763, 2019.

[34] 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.