Literature Survey: Machine Translation

Shreya Alva
Shubham Dewangan
Nitish Joshi
Pushpak Bhattacharyya
CFILT, Indian Institute of Technology Bombay, India
{shreya, nitishjoshi, shubhamd, pb}@cse.iitb.ac.in

Abstract

Machine Translation lies at the heart of Natural Language Processing, which in turn is one of the of the core components of the Artificial Intelligence discipline. In this report, we have a brief look at the legacy approaches to machine translation, while also examining the path that Machine Translation is headed on, with the newest approach to this classic problem - namely neural machine translation This report is a brief introduction to the major works in the field of neural machine translation.

1 Introduction

Artificial Intelligence is transforming the way we interact with the world as it bridges the gap between humans and machines. Machine translation (MT) is the task wherein computers transform input from one natural language into another natural language, while preserving the meaning of the original input. It also serves the purpose of bridging the language divide between humans with the help of automation. The input to an MT system can be a sentence, a block of text or whole documents. This problem is fraught with all kinds of ambiguities that make this problem very challenging for a computer that has no world knowledge or intuitive understanding of human language; to name a few of the ambiguities that exist in this field are morphological (what words to use and how to combine sub-word units), syntactic (relating to grammar), semantic (to infer the meaning of the words, as same word can mean different things like river or financial bank) and so on.

Traditionally there existed 3 approaches to machine translation (Bhattacharyya, 2015): direct MT, Rule Based MT (abbreviated to RBMT) and Data Driven MT. Under Data Driven MT we have 2 approaches, Example Based and Statistical MT (abbreviated to EBMT and SMT respectively). We will discuss the legacy approaches of Rule Based MT and Example Based MT in Chapter 2 and Statistical MT in section 3.

However, in the past decade Neural Network based approaches have gained momentum, as they have in several other fields. This seminar report is an exploration of the journey of machine translation, and where it is headed. The main challenge is to make systems that deliver adequate and fluent translations. Researchers keep endeavoring to improve the translation quality, which is most commonly measured by the BLEU (bilingual evaluation understudy) score, a metric that captures how good the translation done by the machine is by comparing the precision and recall of n-grams (n=1,2,3,4) with some reference translations. Although there may be several translations that need not be among the reference ones, BLEU (and its interactive form, iBLEU) remain the most popular metrics due to the simplicity of calculation and its good correlation with human judgment.

2 Legacy Approaches: RBMT and EBMT

2.1 Rule Based Machine Translation (RBMT)

The original approach to machine translation required a critical involvement by humans and linguistic experts, as training data was not widely available nor were the computational resources of the time up to the mark to exploit the language transfer information could be provided by data. In this approach, humans explicitly stated rules that specified how an input sentence is to be analysed, transferred into an intermediate representation which is then used to generate the target language sentence.

These kinds of systems suffer from a high precision, low recall condition such that when the rules can be correctly applied, they generate the right output, but most of the time these rules cannot be aptly applied. When more than one rule is applicable to a sentence, there can be a conflict between rules, so careful ordering is necessary.

There are two approaches to RBMT - Interlingua and Transfer Based MT.

2.1.1 Interlingua based MT

Interlingua is an intermediate language that represents meaning without any ambiguity. To perform translation, the system needs to convert input from the source language to the form of the interlingua, and then from the interlingua to the target language.

Universal Networking Language (UNL) is one such interlingua. Information is represented sentence-wise. Each sentence can be represented by a hypergraph wherein the nodes are concepts and directed edges depict relations. Concepts are known as universal words (UWs) and are gathered from the lexicon (lexicon entries have syntactic and semantic attributes). They are a language-independent representation of word knowledge. For example, food(icl ¿ salad) denotes the noun salad. The 'icl' notation captures inclusion phenomenon and forms an is-a structure like in semantic nets. A standard set of relation labels (RLs) relate UWs and thus capture conceptual knowledge. Speech acts such as speaker's time, aspect, view, of an event, are depicted via attribute labels.
The reasons that establish the authority of UNL are that it is the only interlingua that does these together:

(1) uses semantic relations to connect words with the main predicate of the sentence.

(2) gives an unambiguous word representation

(3) lucidly expresses properties of speakers' and referents worldview

However creating a UW dictionary that can link different languages lexemes is challenging - the two main obstacles being:

(1) granularity of conceptual space is not the same in all languages and

(2) the existence of multiwords. Multiwords are combinations of words characterized by noncompositionality (the meaning of the combination word cannot be derived by the words that make it up) and fixed collocation of words, in structure and order. Examples: the prepositional verb *took off* which means to run away, phrasal-prepositional verb *catch up with* which could either mean reaching someone person ahead of you, or just getting to talk with someone you haven't met in a long time. To perform translation, first in the Analysis stage, the source sentence is converted to the interlingua, and then the interlingua is converted to the target sentence (generation or G-stage). Since the interlingual representation requires full disambiguation, the A-stage is a complete natural language analysis problem. From the interlingua representation it can be converted into the target language in these 3 stages: selecting the appropriate lexemes and inserting the function words, identifying the case and generating the correct morphology (verb form and so on) and planning the syntax i.e. deciding the word order for these words.

2.1.2 Transfer based MT

Since complete disambiguation of a sentence is a very difficult task, and not even necessary for closely related languages wherein ambiguity can be preserved without affecting the meaning, Transfer based MT gained traction. The process involves taking the source sentence up to an intermediate stage in the Vauquois triangle and then performing transfer to an intermediate representation. This does not demand complete disambiguation of the sentence. The defining characteristic of transfer-based MT is the application of well-defined transfer rules that work with the generalizations of lexical objects to perform structural transformations (JJ: adjectives, NP: noun phrases, V: verb) . Transfer rule T: REPS -> REPT is a mapping from the source language representation REPS to target language sentence representation REPT. To capture the common operation in translating from an SVO language like English to an SOV language like Hindi, a rule of this form < V NP -> NP V >, that reverses the constituents of a verb phrase applies.

2.2 Example Based Machine Translation (EBMT)

The core idea is *translation by analogy*. To perform a translation, we compare the input to the system to the parallel corpora database that the system possesses, which is done by computing text similarity. The fundamental requirement of text similarity are: (1) a valid measure of similarity; it should measure similar texts as indeed similar and dissimilar ones as dissimilar, and (2) resources like large lexical knowledge networks that enable one to capture the notion of 'similarity'. Transfer takes place through templates that are learned from data. EBMT shares stages of processing with RBMT and SMT. The processing that they perform on the source sentences is common to all three; but their outputs differ.

2.2.1 How EBMT is done

The broad steps that are to be followed while doing EBMT are:
1) Compute similarity with templates available
2) Stitch the matching fragments together
3) Smooth boundary friction
The drawback of EBMT is that it doesn't scale up well.

3 Statistical Machine Translation (SMT)

Pure SMT relies wholly on data, the model is built without any human intervention, using corpuses.

It has been the ruling paradigm for until the early 2010s. To teach a machine how to translate, it needs examples which give it this information: Translation of the words (the bilingual word mappings) i.e. what word maps to what word(s) and the correct positions for the translated words in the target sentence (alignment). The underlying principle of SMT is word alignment.

3.1 Word Alignment

When a sentence in one language is translated to another, there exists a mapping between the words. A word from either side of the translation can map to zero, one or more words on the other side. The phenomenon of mapping one word to multiple words on the other side is called fertility. A word can map to multiple words due to:
1. Synonymy on target side, (e.g., 'water' in English translating to jal, paani, neer, etc., in Hindi),
2. Polysemy on source side (e.g., 'brightened' translating to 'khil uthna', as in her face brightened with joy –> usakaa cheharaa khushi se khil uthaa)
3. Syncretism ('to run' translating to 'bhaaga', 'bhaagi', 'bhaage') These alignments are the key to translation.

To learn these word mappings the system requires several parallel sentences, that will introduce the possibility of a particular alignment and others to establish their certitude. Given a sentence pair, the other pair must be of one of the two types: one-same-rest-changed (only one word is common to both sentences) and one-changed-rest-same (the sentences only differ by one word). To compute these mappings, the Expectation Maximization algorithm is used.

The basic equation of SMT is:
$$\arg\max_e \hat{e} = \arg\max_e P(e|f) = \arg\max_e P(e) \cdot P(f|e)$$

where e is a sentence from the sentence bank E of language 1 (say, English), and f is from the sentence bank F of language 2 (the foreign language, like French), P(e) is the language model P(e) and P(f | e) is the translation model. P(e) is computed from the product of N-grams obtained from L1's corpora. On the other hand, P(f | e) would be the product of word translations given the input words, subject to the effect of alignment.

IBM proposed several approaches to build a translation model and alignment model, of which the 3 most important and basic approaches were:

3.1.1 IBM Model 1

It is the most basic model and builds on the following assumptions:

1. Length of the input *m* and length of the output *l* is same, i.e. $P(m + l) = \epsilon$

2. The probabilities of all alignments are equal i.e. any word can map to any word with equal probability. Therefore, $\prod_{j=1}^{m} P(a_j | f_1^{j-1}, a_1^{j-1}, e, m) = \frac{1}{(l+1)^m}$

3. The word f_j depends on the jth word in the alignment *a* in *e* which is e_{aj}, i.e., $\prod_{j=1}^{m} P(f_j | f_1^{j-1}, a_1^{j-1}, e, m) = \prod_{j=1}^{m} P(f_j | e_{a_j})$

Combining all the three assumptions and applying optimization stels, the IBM model-1 becomes:

$$P(f|e) = \frac{\epsilon}{(l+1)^m} \prod_{j=1}^{m} \sum_{i=0}^{l} P(f_j | e_{a_j}) \quad (1)$$

Drawbacks of IBM model-1:

- The condition of input and output sentence having the same length is too rare and limiting.
- Due to language phenomenon, words usually occur in phrases, and it is unlikely that they would have equal alignment probability across languages.
- There could be zero or more alignments possible for a single input word.

3.1.2 IBM Model 2

Assumptions of this model are as follows:

1. It takes alignment into consideration and models it separately i.e., $\prod_{j=1}^{m} P(a_j|f_1^{j-1}, a_1^{j-1}, e, m) = \prod_{j=1}^{m} P(a_j|j, l, m)$
2. As before, f_j depends on e_{a_j}.

The resulting translation model is:

$$P(f|e) = \epsilon . \prod_{j=1}^{m} \sum_{i=0}^{l} \Big(P(i|j, l, m).P(f_j|e_i) \Big) \tag{2}$$

It suffers from similar drawbacks as model 1, only having addressed the issue of accounting for alignment.

3.1.3 IBM Model 3

Works with an alignment model that considers fertility of words. Fertility $\eta(\phi|f)$ can be defined as for each word in foreign sentence, number of words $\phi = 0, 1, 2, \ldots$ are generated. Note that fertility can also take care of *NULL* mapping.

Translation by IBM model-3 is a four step process:.

1. Fertility: For each source sentence word s_i, fertility ϕ_i is chosen with probability $P(\phi_i|e_i)$.
2. Null Insertion: Number of target words to be generated from *NULL* is chosen with probability $\eta(\phi|NULL)$.
3. Lexical Translation: Translation of each word, like IBM model-1, is done by $P(f|e)$.
4. Distortion: Probability of translating f_i (foreign word in i$^{\text{th}}$ position) to s_j (source word in jth position) is modeled by $d(j|i, l, m)$.

$$\begin{aligned} p(f|e) &= \sum_{a_1=0}^{l} \sum_{a_2=0}^{l} \cdots \sum_{a_m=0}^{l} \big(P(f, a|e)\big) \\ &= \sum_{a_1=0}^{l} \sum_{a_2=0}^{l} \cdots \sum_{a_m=0}^{l} \Bigg[\binom{m-\phi_0}{\phi_0} p_0^{m-w\phi_0} p_1^{\phi_0} \\ &\prod_{i=1}^{l} \Big(\phi_i! \eta(\phi_i|e_i) \Big) \times \prod_{j=1}^{m} \Big(t(f_j|e_{e_j}) d(j|a_j, m, l) \Big) \Bigg] \end{aligned} \tag{3}$$

IBM Model 4 and IBM Model 5 are more powerful, but they are too complex. Process of translating a new input sentence, say f^{new} is called *Decoding*. We apply the same Noisy channel model:

$$\begin{aligned} e^{best} &= argmax_{e^{new}} \Big(P(e^{new}) P(f^{new}|e^{new}) \Big) \\ &= argmax_{e^{new}} \Big(P(e^{new}) \frac{\epsilon}{(l+1)^m} \sum_{a} \prod_{j=1}^{m} P(f_j^{new}|e_{a_j}^{new}) \Big) \\ &= argmax_{e^{new}} \Big(P(e^{new}) \frac{\epsilon}{(l+1)^m} \prod_{j=1}^{m} \sum_{i=0}^{l} P(f_j^{new}|e_{a_j}^{new}) \Big) \\ &= argmax_{e^{new}} \Big(\frac{\epsilon}{(l+1)^m} \Big) \Big(P(e_0^{new}) \Big) \Big(\prod_{i=1}^{l} P(e_i^{new}|e_{i-1}^{new}) \Big) \\ &\qquad \Big(\prod_{j=1}^{m} \sum_{i=0}^{l} P(f_j^{new}|e_{a_j}^{new}) \Big) \end{aligned} \tag{4}$$

When languages have different fertilities, word based translation suffers. Thus phrase based translation is the most seminal way of doing SMT, with phrases (not necessarily linguistic) instead of words being the translation units.

One major drawback of SMT is that it requires individual tuning of various components. Thus ushered in the era of Neural Machine Translation - which comprises of an end to end system that can be jointly tuned instead of having to adjust each of the individual components.

4 Sata-Anuvadak

As the name Sata-Anuvadak (Kunchukuttan et al., 2014) suggests, this literature is about 100 translators. This work focuses on multiway translation of Indian Languages. In this work, Indian languages from various families are considered and following objectives were investigated:

1. To understand translation patterns involved when translating between same language family and other.
2. To leverage use of shared characteristics between Indian language. The shared characteristics in brief are as follows:
   (a) Free Word Order with SOV(Subject-Object-Verb) form being canonical.
   (b) Most of the Indian languages have scripts derived from the Brahmi Script.
   (c) Vocabulary and Grammatical rules are majorly derived from Sanskrit language.
   (d) Most of the Indian languages are morphologically rich in nature.
3. To investigate the effect of pre-processing and post-editing in SMT systems for Indian languages.

In general, this literature worked on four systems, they were:

1. Baseline Phrase-Based System: This system is used to investigate relationship between translation models while taking Language families and corpus size into consideration.

2. English-Indian Language with reordering : This system focuses on English to Indian Language translation by applying reordering rules on English side so that both the sides conform to the same word order.

3. English-Indian Language with Hindi-tuned source side reordering: This system uses better reordering rules and concludes that when creating rules by considering the English-Hindi reordering rules benefits the overall system.

4. Indian Language- Indian Language with Transliteration: For the unknown words translation, that don't share the same script, Transliteration can be done by exploiting the Unicode ranges that has been allotted to Indian languages.

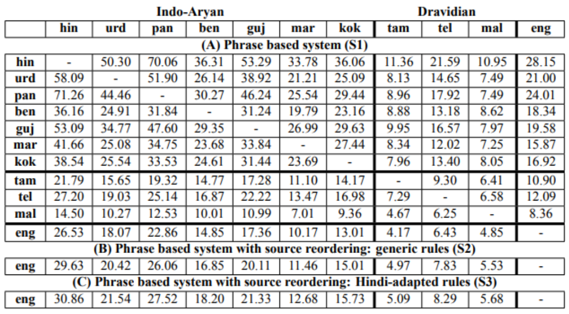
But, the current focus of the project is Indian-Indian language and that too very specifically, Hindi-Marathi and for that we have taken the results of System1 described above as the baseline for all our experiments. Since, Hindi & Marathi share the same Devanagari script, thus, Transliteration technique needn't be applied as a post-editing step.

The following important conclusions came out of the work:

1. Translation accuracy between Indo-Aryan language families show best results because of the same word order and similar case marking whereas the Dravidian languages being more morphologically rich show less accuracies.

2. The effect of increasing corpus size benefits the Indo-Aryan languages to a great extent. For morphologically poor languages, the effect of increase in corpus size helps the translation accuracies to improve whereas for morphologically richer languages, increasing the corpus size is not that worth it.

3. Morphologicially richer to poorer languages translation gives poor results because of absence of source side phrases because of morphological richness, solution to which, morphological segmentation can help solve this issue.



	Indo-Aryan							Dravidian			
	hin	urd	pan	ben	guj	mar	kok	tam	tel	mal	eng
(A) Phrase based system (S1)											
hin	-	50.30	70.06	36.31	53.29	33.78	36.06	11.36	21.59	10.95	28.15
urd	58.09	-	51.90	26.14	38.92	21.21	25.09	8.13	14.65	7.49	21.00
pan	71.26	44.46	-	30.27	46.24	25.54	29.44	8.96	17.92	7.49	24.01
ben	36.16	24.91	31.84	-	31.24	19.79	23.16	8.88	13.18	8.62	18.34
guj	53.09	34.77	47.60	29.35	-	26.99	29.63	9.95	16.57	7.97	19.58
mar	41.66	25.08	34.75	23.68	33.84	-	27.44	8.34	12.02	7.25	15.87
kok	38.54	25.54	33.53	24.61	31.44	23.69	-	7.96	13.40	8.05	16.92
tam	21.79	15.65	19.32	14.77	17.28	11.10	14.17	-	9.30	6.41	10.90
tel	27.20	19.03	25.14	16.87	22.22	13.47	16.98	7.29	-	6.58	12.09
mal	14.50	10.27	12.53	10.01	10.99	7.01	9.36	4.67	6.25	-	8.36
eng	26.53	18.07	22.86	14.85	17.36	10.17	13.01	4.17	6.43	4.85	-
(B) Phrase based system with source reordering: generic rules (S2)											
eng	29.63	20.42	26.06	16.85	20.11	11.46	15.01	4.97	7.83	5.53	-
(C) Phrase based system with source reordering: Hindi-adapted rules (S3)											
eng	30.86	21.54	27.52	18.20	21.33	12.68	15.73	5.09	8.29	5.68	-

Figure 1: Sata-Anuvadak System1 Results(Kunchukuttan et al., 2014)

5 Morfessor

5.1 Introduction

5.1.1 What is Morpheme

Morpheme is a indivisible morphological unit of a language that carries some meaning. For example: un, break, able together forms 'unbreakable'. Here 'un' , 'break' & 'able' constitutes the morpheme set of English language.

5.1.2 Why Learning Morpheme Important?

When considering language with large vocabulary, learning morphemes helps. In support of this statement, consider a task of estimation of n-gram model, which means finding the probabilities of all the words in the n-gram sequence. When the vocabulary size is large enough, the n-gram estimation faces two major problems, they are:

1. Since the vocabulary size considered is very high and when the word itself is considered as the basic unit, then there exists large set of word representations and thus increasing data sparsity. The n-gram estimation because of data sparsity is poor as a result.

2. When considering morphologically rich languages, they have large number of word forms which are not even seen at the time of training. The model fails to deal with new word forms. Thus, learning sub-word units like morphemes may help dealing with this issue.

5.2 Approaches

In the work(Creutz and Lagus, 2002), there are 2 unsupervised approaches to learn morphemes are discussed. They are discussed below with minor details:

5.2.1 Method1: Recursive Segmentation & MDL Cost

1. **Model Cost Using MDL**: To define the model cost, few terminology we must be aware of are:

    - Tokens: The total number of words in a text or corpus regardless of how often they are repeated.
    - Types: The total number of distinct tokens in a text or corpus is called types. For example: In the sentence "I am Indian. I love India.", there are 6 tokens and 5 types in this example.
    - Codebook: The vocabulary of all the morph types is termed as morph codebook or simply codebook.

    The total cost is divided into two parts: (1) Cost of Source Text and (2) Cost of Codebook. Refer to the equation 5

$$
\begin{aligned}
C &= Cost(Sourcetext) + Cost(Codebook) \\
&= \sum_{tokens} -logp(m_i) + \sum_{types} k * l(m_j)
\end{aligned} \tag{5}
$$

    The MDL cost function described in the equation 5 is explained below:

    (a) **Cost(Source Text)**: The cost of source text is the sum of negative likelihood of the morph, summed over all the tokens in the source text.

    (b) **Cost(Codebook)**: The cost of codebook is the total length in bits required to represent all the morph types in the codebook. The term $l(m_j$ represents the length of the morph token and 'k' represents the number of bits required to represent a character. Say, for English lowercase alphabets, 26 different representation is required which would be sufficed by 5 bits i.e. 32 different representations.
    The probability of m_i is calculated by maximum-likelihood estimate as follows in equation 6:

$$
p(m_i) = \frac{count(m_i)\, in\, source\, text}{total\, count\, of\, morph\, tokens} \tag{6}
$$

    (c) **Recursive Segmentation**: A recursive approach is applied for the search of best morph segmentation. The below described points tell us in brief about the recursive segmentation technique:

    - Initially, all the tokens in the source text is considered as valid morphs and are added to the codebook.
    - These tokens are tried for all different splits into two and the one with minimum total cost is chosen. The total cost is calculated by recursively performing this step on the two segments we have after segmentation.
    - The segmentation can be seen as a binary tree and an example of it is shown in the figure 2. The figure takes two words into consideration from Finnish language with some overlap in the morphs. There are numbers associated with each boxes in format a:b where a represents the position where segmentation is happening and b represents the count of that token in source text. The count of chunk must be equal to the sum of occurences of its parents i.e. in the example the morph 'auton' is having count as 9 which is sum of occurences of 'linja-auton' (5) & 'autonkulijettajallakaan (2)'.
    - To find a split, we search for the exact match of the word in this hierarchical structured tree and recursively trace the splits until the leaf nodes are reached.

    (d) **Adding & Removing Morphs**: When we encounter a new word in the source text, we just don't use the already seen morphs to split the word as that may lead to local optima. For, each new word seen:

    - if the word has been observed already, remove its instance from the tree and decrease the count of all the

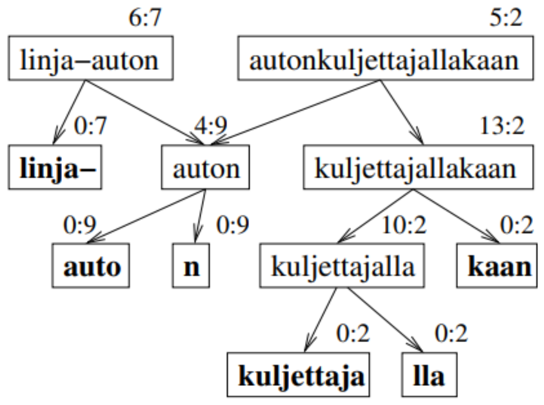


Figure 2: Hierarchical Segmentation of 2 words from Finnish Language (Creutz and Lagus, 2002)

children associated with that chunk. Do the segmentation of the word afresh. This may result in better segmentation and will avoid local optima issue.

- if the word is observed for the first time, segment it by looking at all the possible split it can hold which has the minimum cost associated as described in the previous section.

(e) **Dreaming**: Since, the cost is updated only with every next token processed, the quality of segmentation of the words that occurred at the very beginning and didn't occur again in the training stage might be poor as the model learns new morphs gradually. To resolve the quality issue of poor segmentation of the words that occurred in the beginning, we do a step called *Dreaming*, where no new words are processed, rather process the words already seen and that too in random order. Dreaming continues for some specific amount of time or until there is reasonable decrease in the cost. The figure 3 shows how dreaming lead to decrease in the cost.

5.2.2 Method2: Sequential Segmentation & ML Cost

1. **Model Cost Using Maximum Likelihood**: The cost function used is the likelihood of the data given the model. The ML estimate is described in the equation 7. The ML estimate for the morph tokens is as same as the one described in the previous method 6. In this method, we are just considering the probabilty of data given the model and ignoring the cost of the model.

$$Cost(Sourcetext) = \sum_{morphtokens} -logp(m_i) \tag{7}$$

2. **Search Algorithm**: This section describes how the splitting is done. The following steps are followed to learn the splittings.

   (a) For the initialization, using a Poisson distribution with $\lambda = 5.5$, the length of the split is sampled and using this length of the interval, the splitting of words is carried out. If the length of the interval sampled is greater than the chunk left after segmenting, then segmentation for that word stops.

   (b) The following steps are repeated some fixed number of times which is a hyperparameter to the training process:

      i. The morph probabilities are estimated for the splitting we have.

      ii. With the splitting we have, i.e. the codebook, re-segment the source text using the Viterbi algorithm to find the best segmenatation of the words into morphs with minimum cost.

      iii. If not last iteration, test the segmenatation using the *Rejection criteria*. If the segmentation doesn't fit according to the Rejection Criteria, make the split randomly as done in the ini-

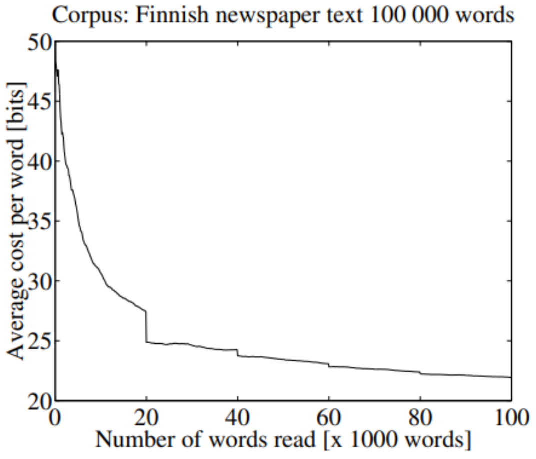


Figure 3: Effect of Dreaming on Cost (Creutz and Lagus, 2002)

tialization step. This random splitting leads to better learning of the model as it leads to learning of new morphs. Without this random splitting, the model would not achieve optimal solution.

   (c) **Rejection criteria**: The following rules are followed under the Rejection criteria defined:

      i. Removal of rare morphs: The morphs which occurs just once till previous iteration are removed as it is known fact that the morphs whose occurrence is extremely low are most probably wrong morphs.

      ii. Reject the segmentation if it contains multiple single letter morphs. Multiple single-letter morphs in the segmentation is often the sign of bad performance of the model meaning that model is stuck in local optimum.

5.2.3 Method3: MAP Estimate

This section is closely related to Mathematical Formulation decribed in Morfessor 1.0(Creutz and Lagus, 2002). According to the work (**?**), the Minimum Description Length (MDL) principle & Maximum a Priori (MAP) Estimate are equivalent & produce the same result.

1. **MAP of the overall probability**: The aim is to induce a model of the language from given source-text in an unsupervised manner. The model of the language (M) consists of two elements:

   (a) **Lexicon**: The lexicon is the set of distinct morphs in the source text. Basically, it is the vocabulary of the segmented corpus.

   (b) **Grammar**: The grammar contains the information about how the segmented text can be recombined.

   The aim is to create a model of the language in such a way that the set of morphs is concise & also gives the concise representation of the source text or the corpus. The MAP estimate for the parameters that is to be maximized is describes in the following equation 8 and 9

$$\underset{M}{argmax} P(M|corpus) = \underset{M}{argmax} P(corpus|M) * P(M) \tag{8}$$

$$P(M) = P(lexicon, grammar) \tag{9}$$

   The equation 8 shows that the MAP estimate is composed of two parts i.e. The Model probability & ML estimate of the corpus given the Model. The equation 9 describes that the model of the language is calculated as the joint probability of the lexicon & the grammar which incorporates our assumption that the morphology learning task must be affected by some or the other features.

2. **Lexicon**: The lexicon is the vocabulary of the segmented text or simply the collection of all the morph units in the corpus. Each morph carries some properties and in the Morfessor Baseline model, the only properties considered are *frequency* of morphs in the source-text or the corpus & the *string* that morph is composed of which also implicitly carries the information of the length of string i.e. the number of characters in the sequence.

3. **Grammar**: The grammar contains the information about how the segmented text can be recombined. But in the baseline model of Morfessor, the grammar is not considered and is taken into consideration in later models. So, the P(M) reduces to P(lexicon) only. In absence of grammar, the model could not decide whether morph should be placed in starting, ending or the middle of the word formed. The probability of morph μ_i is ML estimate and is calculated as described in the equation 10.

$$PP(\mu_i) = \frac{f_{\mu_i}}{N} = \frac{f_{\mu_i}}{\sum_{j=1}^{M} f_{\mu_j}} \tag{10}$$

4. **Corpus**: The words present in the corpus are represented as a sequence of morphs. There might be multiple segmentation of a word and is chosen based on the MAP estimate. The probability of Corpus given the model is estimated as described in equation 11 where in

the corpus W words have been considered & each word is split into n_j morphs. The k^{th} morph of the j^{th} word is termed as μ_{jk}.

$$P(corpus|M) = \prod_{j=1}^{W} \prod_{k=1}^{n_j} P(\mu_{jk}) \quad (11)$$

5. **Search Algorithm**: A greedy search algorithm is utilized for finding the optimal lexicon and the segmentation. Different segmentation is tried and the segmentation that produces the highest probability is selected. This continues until there is no significant improvement seen.
The probabilities are replaced by log probabilities & the product is replaced by sum. *The negative log probability is considered as code lengths as in MDL framework.*

6. **Recursive Segmentation**: A recursive approach is applied for the search of best morph segmentation. The below described points tell us about the recursive segmentation technique:

   - Initially, all the tokens in the source text is considered as valid morphs and are added to the codebook.
   - These tokens are tried for all different splits into two and the one with minimum code length is chosen. The total code length is calculated by recursively performing this step on the two segments we have after segmentation.
   - The segmentation can be seen as a binary tree and an example of it is shown in the figure 4. The figure takes two words into consideration from language with some overlap in the morphs. There are numbers associated with each boxes in format a:b where a represents the position where segmentation is happening and b represents the count of that token in source text. The count of chunk must be equal to the sum of occurences of its parents i.e. in the example the morph 'auton' is having count as 9 which is sum of occurences of 'linja-auton' (5) & 'autonkulijettajallakaan (2)'.
   - To find a split, we search for the exact match of the word in this hierarchical structured tree and recursively trace the splits until the leaf nodes are reached.
   - The above steps are repeated for multiple epochs and in between 2 epochs the words are randomly shuffled to avoid any inherent bias because of the words appearing in the corpus.

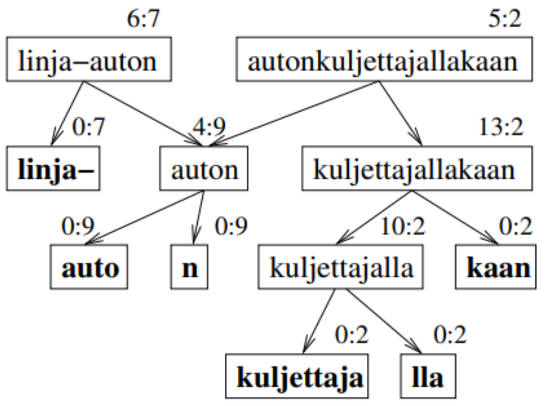


Figure 4: Hierarchical Segmentation of 2 words from Finnish Language(Creutz and Lagus, 2002)

6 BiRNN encoder, RNN decoder architecture

Bahdanau et al. (2015) proposed the first system that jointly learnt how to align and translate input and output sentences with a novel architecture.
The probability of the output sequence is given by the following equation:

$$p((y) = \prod_{t=1}^{T} p(\{y_t|y_1, \dots, y_{t-1}\})$$

The conditional probability for each output word is given by:

$$p(y_i|y_1, \dots, y_{i-1}, \mathbf{x}) = g(y_i, s_i, c_i)$$

where, $\mathbf{x}$ is the input sentence having words denoted by vectors $\mathbf{x_1} \dots \mathbf{x_{T_x}}$ and s_i is the hidden state of the RNN at time i.
The words of a sentence are encoded as one-hot vectors on a limited vocabulary of the words of a language. The input is encoded by a Bi-directional RNN (BiRNN). The forward RNN reads the input sequence, in the given order and calculates a sequence of forward hidden states; the backward RNN does the reverse operation. The encoder maps the input sentence to a sequence of annotations. Each word has an annotation associated

(h_j), whose value is the concatenation of the 2 hidden states.
The model generates a context vector for each word in the output sentence; this was different from previous models of NMT. c_i is the context for target word i. $c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$. α_{ij} is the weight of each annotation, given by:

$$\alpha_{ij} = \frac{exp\,(e_{ij})}{\sum_{k=1}^{T_x} exp\,(e_{ik})} \tag{12}$$

e_{ij} scores how well the input words around position j and output words at position i match.

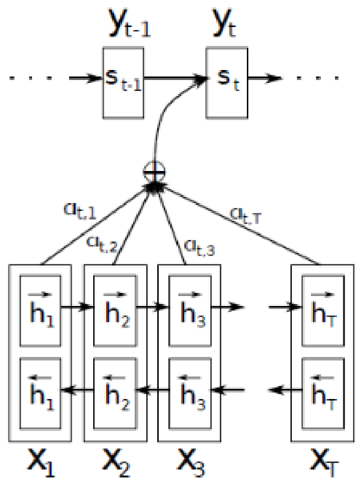


Figure 5: Proposed Model generating target word y_t Bahdanau et al. (2015)

Whenever the proposed model generates an output word, it searches for a set of positions in a source sentence where the most relevant information can be found. The output word is predicted based on the context vectors associated with those pertinent source sentence words as well as the previously generated output words. The decoder uses softmax at the final layer and finds the associated probability for each word in the target vocabulary. In earlier approaches to NMT, the entire input sentence was encoded to a fixed size vector. Here instead a subset of context vectors is adaptively chosen on the fly while decoding.

In this paradigm, alignment is not considered to be a hidden variable, it is computed by a feedforward neural network; model directly finds a soft alignment which means each word need not map to with certainty to a word/words (known as *hard alignment*) but we can assign different probabilities of its mapping to all words in the sentence. The default option for gated units in OpenNMT-py is LSTM.

7 Word Embeddings

Word Embeddings are the n-dimensional vector representation of the word. The general accepted length of the word embedding vector is 300. The whole vocabulary is represented as Word Embedding Matrix and is pre-learned using the algorithm like Word2Vec, Glove, Bert, and many others. The following subsection describes in brief the most common and basic algorithms for obtaining the Word Embeddings from a mono-lingual corpora.

7.1 Word2Vec

Word2Vec ((Mikolov et al., 2013a), (Mikolov et al., 2013b)) was a breakthrough technique for finding word embeddings as opposed to earlier methods that used global counts for obtaining vector representation of the words. The Word2Vec model gave a representation that was found successful in word analogy tasks. For instance, the very famous word analogy example is the vector arithmetic operation as ”king - man + woman” gave a vector representation close to that of ”queen” which was surprisingly great result and made Word2Vec the state-of-the-art technique of finding word embeddings.

The Word2Vec model chooses a window size say 5 (2 words before and 2 words after the word in focus), and applies either of the two approaches to learn word embeddings. They are as follows:

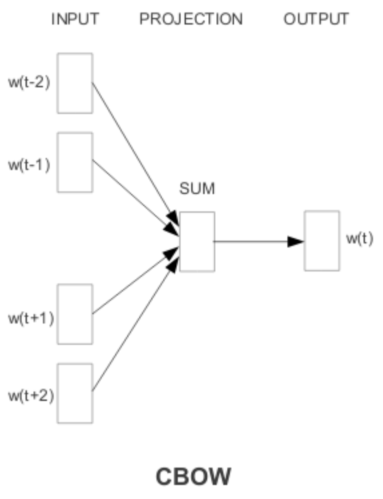


Figure 6: CBOW Visualization ((Mikolov et al., 2013a))

- **Continuous Bag-of-Words model**: The CBOW model aims to predict the word in

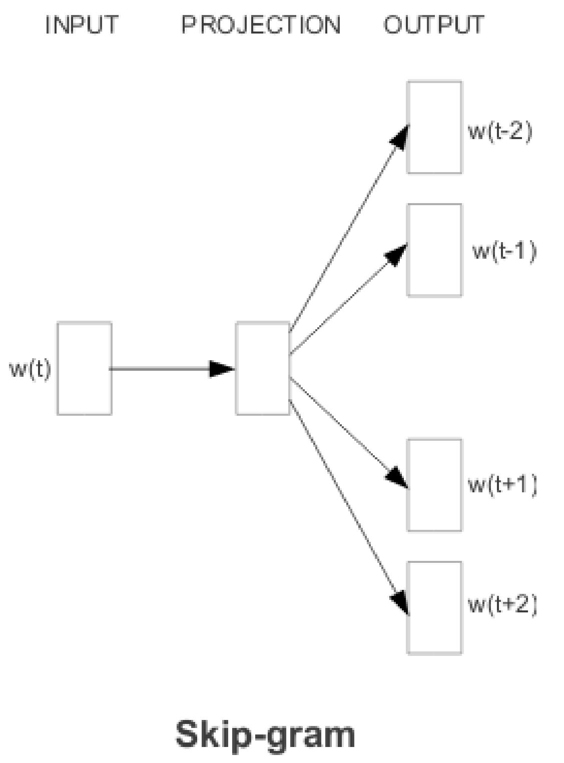


Figure 7: Skip-gram visualization ((Mikolov et al., 2013a))

focus by looking at the words surrounding it i.e. which lies in context window range. Example: The dog —- at strangers. {using The, dog, at & strangers, the CBOW model tries to predict the word 'barks'. }

- **Skip-gram model**: The Skip-gram model aims to predict the context words by looking at the word in focus. {using the word 'barks' the Skip-gram model tries to predict the words The, dog, at & strangers } The training procedure of Word2Vec using Skip-gram model is described in Appendix.

7.1.1 Word2Vec Training

The Word2Vec training is explained below using Skip-gram model with negative sampling

- **Creating the Training Data**: A window size is chosen say 5(2 words before and 2 words after the focus word), and the entire mono-lingual corpus is scanned. To understand exactly how training data set is chosen, take an example sentence as follows: *The cat **sat** on the* mat.

The *cat sat **on** the mat*.

Table 1: Training Data

focus	context	target
sat	the	1
sat	cat	1
sat	on	1
sat	the	1
on	cat	1
on	sat	1
on	the	1
on	mat	1

But, all our examples are with target '1' (i.e for ex: 'the' lies in context of 'sat', so output label is 1, otherwise '0') which makes the embeddings learn nothing from training. However, the model will give 100% accuracy in training set, the model learns only the Garbage embeddings.

Negative Sampling: Adding some 'k' negative samples for each training example to the training set, i.e. word pairs that are not actually neighbours and giving target value '0'.

Table 2: Negative Sampling

focus	context	target
sat	the	1
sat	kite	0
sat	table	0
sat	cat	1
sat	mango	0
sat	bike	0
sat	on	1
sat	the	1
on	cat	1
on	sat	1
on	the	1
on	mat	1

- Initialize two matrices called Embedding matrix and Context matrix of size (vocab_size * chosen dimension)
- Take focus word (sat), context word (the) and negative samples (kite, table) for first step of training.
- Get word embedding of focus word & context word using Embedding matrix and Context

matrix respectively. (v1, v2, v3, v4 represents word embedding in table 3)

Table 3: Error Calculation

e1	e2	target	e1.e2	sigmoid	error
Sat [v1]	The [v2]	1	0.7	0.66	0.34
Sat [v1]	Kite [v3]	0	0.2	0.5	-0.5
Sat [v1]	Table [v4]	0	-1.6	0.16	-0.84

The objective function ((Goldberg and Levy, 2014)) to be maximized is as follows:

$$\underset{\Theta}{argmax} \sum_{(w,c)\varepsilon D} log\sigma(v_c.v_w) + \sum_{(w,c)\varepsilon D'} log\sigma(-v_c.v_w) \tag{13}$$

where, v_c : embedding of the context word and v_w : embedding of the word in focus

7.2 GloVe

GloVe ((Pennington et al., 2014)) stands for Global Vectors for word represntation. GloVe embedding uses global counts knowledge, unlike word2vec. The learning of the embedding is based on co-occurrence matrix formed using the mono-lingual corpus and trains the word vectors (embeddings) such that their difference predict the co-occurrence ratios. Some pair of words occur more often than others and hence the method ensures that such pairs do not contribute much towards the objective loss function by introducing a weight factor for each word-pair based on its frequency of occurrence.

7.2.1 GloVe Training

The GloVe training is purely based on co-occurrence matrix. The principle under which GloVe method works is: The ratio of co-occurrence between two words in context tells about the meaning of the word in focus. The loss function being minimized is as follows:

$$\sum_{i,j} weight(X_{ij})(dot(w_i.\tilde{w_j})+b_i+\tilde{b_k}-log(X_{ij}))^2 \tag{14}$$

where,

- X_{ij} : co-occurrence value of i_{th} and j_{th} word in co-occurrence matrix X.
- w_i : word-embedding of input word
- $\tilde{w_j}$: word-embedding of output word
- b_i and $\tilde{b_k}$: bias terms

8 Bilingual Word Embeddings

There are different ways of representing words in neural models. One particular method is **One Hot Encoding**. In this scheme, we have a very high dimensional vector for every input word. The dimension of this vector is equal to the number of words in our vocabulary. Every word in this vector has a position and this position is set to 1 for the current word and all the other positions are set to 0.For instance:

- *man* = $(0, 0, 1, 0)^T$
- *tan* = $(0, 0, 0, 1)^T$
- *crib* = $(1, 0, 0, 0)^T$

This one-hot vector is very sparse since most of the entries are 0. We map these one hot vectors to lower dimensional space where every input word represents a vector in that space. This is generally referred to as **Word Embedding**. We typically include a word embedding layer between our one-hot vector and the hidden layer of the network. Word embeddings give us *clustering* i.e. generalizing between words and *backoff* i.e. robust predictions under unseen contexts. A complete set of word embeddings can identify similar words and naturally capture relationships between words. For example:

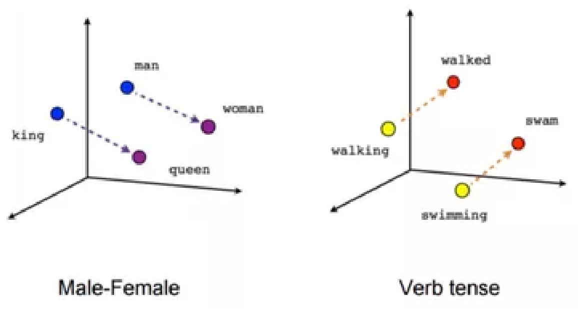


Figure 8: Visualizing the relationship representing capabilities of word vectors

In Mikolov et al. (2013c), the aim is to learn a linear mapping $W \in R_{dd}$ between the word vectors of a seed dictionary.

$$\min_{W \in R_{dxd}} \frac{1}{n} \sum_{i=1}^{n} l(Wx_i, y_i)$$

Typically the square loss is used as the learning objective. W is constrained to be orthogonal since it preserves distances between word vectors and likewise word similarities. Experimentally, it had been observed to improve quality of inferred lexicon.

Once a mapping W is learned, one can infer word correspondences for words that are not in the initial lexicon. Translation $t(i)$ of source word i is obtained as: $t(i) \in argmin_{j \in 1 \dots n} W x_i, y_i$

Word embeddings can improve the quality of language translations when embeddings between two or more languages are aligned using a transformation matrix. Many approaches have been suggested to align word embeddings, and we will be discussing a few of them.

This paper Joulin et al. (2018) presents a different approach to Bilingual Word Mapping. They minimize a convex relaxation of the cross-domain similarity local scaling (CSLS) loss, which significantly improves the quality of bilingual word vector alignment. The CSLS criterion is a similarity measure between the vectors x and y defined as:

$$\text{CSLS}(x, y) = -2cos(x, y) + \frac{1}{k} \sum_{y' \in N_y(x)} cos(x, y') + \frac{1}{k} \sum_{x' \in N_x(y)} cos(x', y) \tag{15}$$

where $N_y(x)$ is the set of k nearest neighbors of the point x in the set of target word vectors $Y = y_1, \dots, y_N$, and cos is the cosine similarity. They removed the orthogonality constraint and found that it does not degrade the quality of aligned vectors. Finding k nearest neighbors of $W x_i$ among elements of Y is equivalent to finding elements of Y having the largest dot product with $W x_i$. Employing this idea leads to the convex formulation when relaxing orthogonality constraint on W. Their approach can be generalized to other loss functions by replacing the term $x_i^T W^T y_j^T$ by any function convex in W. They apply Extended Normalization: instead of computing the k-nearest neighbors amongst the annotated words, use the whole dictionary use unpaired words in dictionaries as 'negatives'. Dataset used was Wikipedia fastText vectors and for supervision a lexicon composed of 5k words and their translations.

The success of orthogonal mapping methods relies on the assumption that embedding spaces are isomorphic; i.e., they have the same inner-product structures across languages, but this does not hold for all languages, especially distant languages. Hoshen and Wolf (2018) align fastText embeddings with orthogonal mappings and reported 81% English–Spanish word translation accuracy but only 2% for English–Japanese.

To combat this, Zhang et al. (2019) puts forth a preprocessing technique called 'Iterative Normalization' that makes orthogonal alignment easier by transforming monolingual embeddings and simultaneously enforcing that: (1) individual word vectors are unit length, and (2) each language's average vector is zero. Iterative Normalization technique: For every word i, we iteratively transform each word vector xi by making the vectors unit length, Then making their mean zero.

This is what sets them apart from previous approaches:

- Earlier, the zero-mean condition was motivated based on the heuristic argument that two randomly selected word types should not be semantically similar (or dissimilar) in expectation.
- But word types might not be evenly distributed in the semantic space, some words may have more synonyms.
- Single round of length and center normalization doesn't suffice.

9 Transformer

While RNNs and LSTM based Neural Networks had been established as the state of the art approaches in sequence modeling tasks like MT, they are constrained by sequential computation. Typically, the generated sequence of hidden states h_t is a function of the previous hidden states h_{t-1} and the input at time t. Attention mechanisms have become a vital part of such models recently, allowing modeling of dependencies regardless of their distance in the input or output sequences. The Transformer, proposed by Vaswani et al. (2017) is model architecture forgoes recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output. The Transformer allows for significantly more parallelization, faster training time and reached a new state of the art in translation quality.

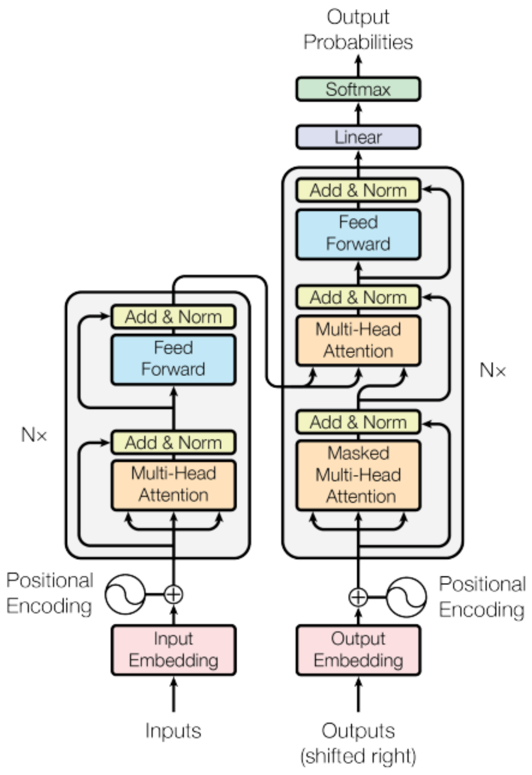


Figure 9: The Transformer model architecture.

Both the encoder and the decoder are composed of a stack of N = 6 identical layers. Each encoder layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. That is, the output of each sub-layer is $LayerNorm(x + Sublayer(x))$. As for the decoder, it inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. A residual connection is employed around each of the two sub-layers, followed by layer normalization. The self-attention mechanism is also modified to prevent positions from attending to subsequent positions. This masking, combined with the fact that the output embeddings are offset by one position, ensure that the predictions for position i can depend only on the known outputs at positions less than i.

The authors have chosen to describe the attention function as mapping a query and a set of key-value pairs to an output, wherein they are all vectors.

9.1 Attention

There are two kinds of attention described in the paper. The input consists of queries and keys of dimension d_k, and values of dimension d_v.

- **Scaled Dot-Product Attention**: is computed via matrix multiplication on a set of queries, using highly optimized code for the same. Q, K, V are the query, key, and value matrices where the respective vectors are packed in. It is much faster and space-efficient in practice compared to additive attention.

$$Attention(Q, K, V) = softmax\frac{QK^TV}{\sqrt{d_k}} \tag{16}$$

The product is divided by $\sqrt{d_k}$ to counteract when softmax function is pushed into extremely low gradient regions when the dot products grow too large in magnitude for large values of d_k.

- **Multi-Head Attention**: Instead of a single attention function, the authors discovered that projecting the queries, keys, and values h (fixed as 8) times, in parallel, with different, learned linear projections into dimension d_v, was beneficial. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

where,

$$\text{head}_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{18}$$

The multi-head is utilized in 3 ways. In encoder-decoder attention layers, queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder, allowing every position in the decoder to attend over all positions in the input sequence (reminiscent of sequence to sequence models). In the encoders' self-attention layers, the queries, keys, and values come from the output of the previous layer, allowing it to attend to all positions in the previous encoder layer. In the decoder, however, as constrained by the auto-regressive property, each position is only allowed to attend to positions up to and including that position.

9.2 Position-wise Feed-Forward Networks

Each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. While the linear transformations are identical across different positions, there is no weight sharing between layers.

$$FFN(N) = max(0, xW_1 + b_1)W_2 + b_2$$

9.3 Positional Encoding

Since the model contains no recurrence (or convolution), to impart the model with knowledge about the order of the sequence or the positions in the sequence, positional encodings are added to the input embeddings at the bottoms of the encoder and decoder stacks. In this work, they chose sine and cosine functions of different frequencies:

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

where pos is the position and i is the dimension.

9.4 Advantages of Self-Attention

The authors mention 3 key benefits of using self-attention. These are:

- Total computational complexity per layer: Complexity per Layer is $O(n^2d)$ operations for self-attention layers and $O(nd^2)$ for recurrent layers, making the former faster when the sequence length n is smaller than the representation dimensionality d, which is most often the case.
- A self-attention layer connects all positions with a constant number of sequentially executed operations, whereas a recurrent layer requires O(n) sequential operations. This means parallelizing computation is easier in the former case.
- Path length between long-range dependencies in the network is also constant with self-attention layers. Learning long-range dependencies is a key challenge in many sequence transduction tasks, and shorter paths lend to easier learning.

10 Byte Pair Encoding

Closed vocabulary systems are those that have a finite, predefined vocabulary; the test set will contain words from the vocabulary itself, i.e., there will be no unknown words. But, translation is an open vocabulary problem, it might happen that in real-world situations, a word in the test set was never seen during training. Such words are called Out of Vocabulary or OOV words. In an open vocabulary system, such OOV words are replaced by a special token, ¡UNK¿ while the output is generated. Neural machine translation (NMT) models typically operate with a fixed vocabulary on the source and target side (due to memory and computational constraints). Thus OOV words pose a problem that impacts the output's fluency and adequacy. Instead of using words as input and output tokens during translation, Sennrich et al. (2016) found that in addition to making the translation process simpler, subword models achieve better accuracy for the translation of rare words than large-vocabulary models and back-off dictionaries, They are also able to productively generate new words that were not seen at training time i.e. they could learn compounding and transliteration from subword representations. The intuition behind this approach comes from the fact that some words are translatable, even if they are new to a competent translator based on a translation of known subword units such as morphemes or phonemes. Such words can be of these types:

- Named entities: can often be copied from source to target text via transcription or transliteration. Example: Barack Obama (English) as (Hindi).
- Cognates and loanwords: with a common origin can differ in regular ways between languages so that character-level translation rules suffice. Example: Kettle (English) as (ketlii, Hindi)
- Morphologically complex words: words containing multiple morphemes, for instance, formed via compounding, affixation, or inflection, maybe translatable by translating the morphemes separately. Example: Solar System (English) as Sonnensystem (Sonne + System, German).

In an analysis of a sample of 100 rare tokens in their German training data, they found that the majority of tokens are potentially translatable from English through smaller units. They identified 56 compounds, 21 names, 6 loanwords with a common origin (emancipate $\rightarrow$ emanzipieren), 5 cases of transparent affixation, 1 number, and 1 computer language identifier.

Byte Pair Encoding (BPE) (Gage, 1994) is a simple data compression technique that iteratively replaces the most frequent pair of bytes in a sequence with a single, unused byte. The authors adapted this algorithm for word segmentation by merging characters or character sequences instead.

Their algorithm is as follows:

- Initialize the symbol vocabulary with the character vocabulary. Represent each word as a sequence of characters, plus a special end-of word symbol '·' to allow restoration to the original tokenization after translation.
- Iteratively count all symbol pairs and replace each occurrence of the most frequent pair ('X', 'Y') with a new symbol 'XY'.
- Each merge operation produces a new symbol that represents a character n-gram. Frequent character n-grams (or whole words) are eventually merged into a single symbol, thus BPE requires no shortlist.

They evaluated two methods of applying BPE:-

- Learning two independent encodings, one for the source, one for the target vocabulary. This has the advantage of being more compact in terms of text and vocabulary size and having stronger guarantees that each subword unit has been seen in the respective language's training corpus.
- Joint BPE: learning the encoding on the union of the two vocabularies. This improves consistency between the source and the target segmentation.

When BPE is applied independently, the same name may be segmented differently in the two languages, which makes it harder for the neural models to learn a mapping between the subword units. To increase the consistency between English and Russian segmentation despite the differing alphabets, they transliterated the Russian vocabulary into Latin characters with ISO-9 and learnt the joint BPE encoding. Then they transliterated the BPE merge operations back into Cyrillic to apply them to the Russian training corpus. For OOVs, the baseline strategy of copying unknown words works well for English $\rightarrow$ German. However, when alphabets differ, like in English $\rightarrow$ Russian, the subword models do much better.

The final symbol vocabulary is equal to the size of the initial vocabulary plus the number of merge-operations. Also, the number of merge operations acts as a hyperparameter which can be further tuned for optimal performance.

11 Pivot Based MT

Having looked at some of the literature that is present in the domain of subword translation, we now look at another way we can alleviate the problem of data scarcity for low resource languages namely **Pivot based approaches**. In this approach we use an intermediate language called a *Pivot* language such that we have parallel corpora from the actual source to the pivot and from the pivot to the actual target. In this way, we circumvent the need for a parallel corpus from the actual source to the actual target language. This usage of pivot languages is a boon for us especially in situations where the parallel corpora for the source and the target is very scarce. For pivot based translation we have several approaches that we can use :

1. **Triangulation Approach**: In this approach, we combine the source-pivot phrase table and the pivot-target phrase table to get the source-target phrase table.
2. **Sentence Translation Approach**: In this approach, we first translate from source to pivot and then from pivot to target language.
3. **Corpus Synthesis Approach**: In this approach, we build the source-target corpus by either translating the pivot sentences form the source-pivot corpus using the pivot-target translation system or by translating the pivot sentences from the pivot-target sentences using the source-pivot translation system.

One of the major problems in pivot based translation systems is the propagation of error. The translation produced by the pivot-target system is dependent on the quality of the translation of the source-pivot system i.e. any errors in the first stage are bound to be carried onto the second stage. The paper by Cheng (2019). addresses this concern by joint training for the pivot based NMT. The crux of the approach is that the source-to-pivot and the pivot-to-target systems are allowed to interact with each other during the training phase through either sharing of word embeddings on the pivot language or maximizing the likelihood of the cascaded model on a small source-target parallel corpus. The objective function can be visualized as consisting of a source-to-pivot likelihood, a pivot-to-target likelihood, and a connection term that connects both the models and whose influence is governed by a hyperparameter. The models are connected using word embeddings of the pivot. They can also be connected by using a small bridging corpus.

12 Word and Morpheme injected NMT

We suspected that BPE could cause ambiguity due to word segmentation as smaller word segments might have lesser context when compared to the whole word. This leads us to augment our BPE model by adding a word and morpheme features to it. The idea was to merge word and morpheme embeddings with underlying BPE embeddings to explore if this addition can help the model to mitigate the problem of ambiguity and learn better. For combining embeddings, we have used Multi-Layer Perceptron.

We updated this model by initializing it with pre-trained BPE embeddings. We've accomplished it by first obtaining pre-trained BPE embeddings from (Heinzerling and Strube, 2018) and then used these pre-trained BPE embeddings to initialize our model. We've managed to get pre-trained embeddings for around eighty per cent of our vocabulary, and the rest was initialized with random vectors.

We further updated the model by replacing the softmax function with sparsemax function while calculating attention. The intuition behind it was, there can be several words in a sentence which may not contribute at all to the attention, but the softmax function will give them some non zero probability score(though low). This could lead to an unnecessary diminishment in the intensity of the final context vector as the ultimate context vector is a weighted sum of probability scores from all the words. With the use of sparsemax function, we can eliminate this problem as sparsemax function will truncate those probability scores to zero, which are too low or say less than some given threshold

13 BERT augmented NMT

BERT(Devlin et al., 2019)(Bi-directional Encoder Representation From Transformer) makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. BERT is typically used for other downstream tasks like next sentence prediction and text classification. We aimed to leverage the BERT model in our NMT task. We accomplished this, by taking sentence vector(which captures the contextual relationship between words in a sentence) generated by BERT and then merging this sentence vector with the sentence vector generated by an encoder using the multi-layer perceptron(MLP). The sentence vector generated by BERT is of 768 dimensions, while that produced by the encoder is of 300 dimensions. The final vector after merging former two vectors is of 768 dimensions which is then passed to the decoder.

14 Conclusion

Machine Translation has come a long way since the time of RBMT. Now, humans do not need to hand craft their translation systems, programming it with rules and examples that would guide it. The approach didn't scale well, suffered from a high precision low recall fault, and was cumbersome and expensive. Data became the driver of MT systems with SMT learning mappings and their probabilities from corpuses. Such systems required calibration of several subsystems. Their success lead them to be the reigning paradigm for a long time. In recent years however, with powerful processors and massive amounts of data, Neural Networks managaed to catch up with SMT. SMT still performs better in low resource conditions, but NMT has the advantage of being an end to end model, that can be trained jointly. NMT achieves translations of good quality due to its attention mechanism, whereby it can concentrate on different parts of the input sentence and the output it has generated thus far, to predict the next time. To rectify the issue of NMT needing large amounts of parallel data, researchers are trying to adopt an unsupervised approach and leverage monolingual data. The key tenets of this are proper initialization of the model, good language models and the concept of back-translation. There is still scope for improving the performance of such models and to understand how exactly these models work.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Pushpak Bhattacharyya. 2015. *Machine translation*. CRC Press.

Yong Cheng. 2019. Joint training for pivot-based neural machine translation. In *Joint Training for Neural Machine Translation*, pages 41–54. Springer.

Mathias Creutz and Krista Lagus. 2002. Unsupervised discovery of morphemes. In *Proceedings of the*

ACL-02 workshop on Morphological and phonological learning-Volume 6, pages 21–30. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805.

Yoav Goldberg and Omer Levy. 2014. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*.

Benjamin Heinzerling and Michael Strube. 2018. BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Armand Joulin, Piotr Bojanowski, Tomas Mikolov, Hervé Jégou, and Edouard Grave. 2018. Loss in translation: Learning bilingual word mapping with a retrieval criterion. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2979–2984, Brussels, Belgium. Association for Computational Linguistics.

Anoop Kunchukuttan, Abhijit Mishra, Rajen Chatterjee, Ritesh Shah, and Pushpak Bhattacharyya. 2014. Sata-anuvadak: Tackling multiway translation of indian languages. *pan*, 841(54,570):4–135.

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

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

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013c. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

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

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.

Mozhi Zhang, Keyulu Xu, Ken-ichi Kawarabayashi, Stefanie Jegelka, and Jordan Boyd-Graber. 2019. Are girls neko or shōjo? cross-lingual alignment of non-isomorphic embeddings with iterative normalization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3180–3189, Florence, Italy. Association for Computational Linguistics.