Literature Survey: Neural Machine Translation

Ajay Anand Verma
Pushpak Bhattacharyya
CFILT,Indian Institute of Technology Bombay, India
{ajayanand, pb}@cse.iitb.ac.in

Abstract

Neural Machine Translation (NMT) is a new highly active approach for machine translation, which has showed promising results and due to its success it has attracted many researchers in the field. In this paper we investigate how NMT architecture changed over very short span of time. Starting with basic encoder-decoder architecture that suffered two problems, poor performance with longer sentences and out-of-vocabulary (OOV) problem. Attention based NMT performs better with longer sentences but it still faces the OOV problem. To deal with OOV problem attention based NMT along with subword segmentation called Subword NMT are being used. With word segmentations NMT systems are able to cover larger range of vocabulary. Later factored NMT were proposed which shows improvement in performance, but requires linguistic features annotated data, which may not be available easily for most of the language pairs.

1 Introduction

Neural machine translation is a new approach to machine translation, recently proposed by (Kalchbrenner and Blunsom, 2013), (Sutskever et al., 2014) and (Cho et al., 2014a). From the probabilistic point of view translation is finding the best possible target sentence that maximizes the conditional probability of $\mathbf{y}$ given $\mathbf{x}$ i.e. $\mathbf{p}(\mathbf{y}|\mathbf{x})$ (Bahdanau et al., 2014). Phrase based Statistical Machine Translation (PBSMT) (Koehn et al., 2003) solves the problem of machine translation by training sub-components separately like *language model* and *translation model*. In contrast to PBSMT in Neural Machine Translation an end-end to single large neural network model is trained to achieve the same goal.

Basic idea behind NMT is to encode a variable length sequence of words into a fixed length vector that can summarize the whole sentence. Then decode this encoded vector in target language to achieve the translation of source sentence. Whole encoder-decoder model is trained jointly to maximize the conditional probability $\mathbf{P}(y|x)$, where y is target sentence that NMT model generate and x is source sentence.

2 Neural Machine Translation

NMT models consists of two parts called *encoder* and *decoder*. Encoder is responsible for generating a real vector representation of sentence called summary vector or context vector that captures the necessary features of sentence. Ideally context vector should be able to represent all the information present in source sentence in real vector representation. Decoder process this context vector to generate target language sentence word by word, such that all the meaning present in source sentence is transferred in it.

Since the source sentences and target sentences can be of variable sizes, we use Recurrent Neural Networks (RNNs) to process them. Practically Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) variants of RNN are used.

2.1 Long Short Term Memory

RNN consumes sentence word by word and update the hidden state on processing each word. Simple RNN use $tanh$ or $sigmoid$ as activation function, which fail to capture various long term

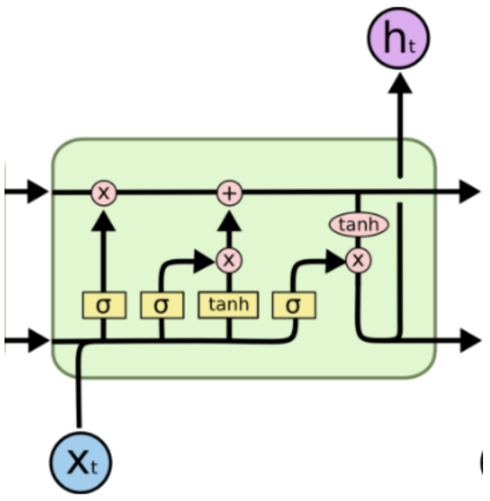
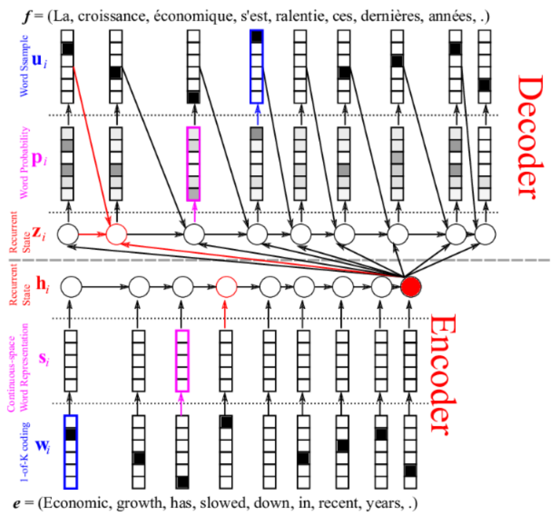


Figure 1: Long Short Term Memory cell architecture. From source[1]



f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Word Sample
u_i
Word Probability
p_i
Recurrent State
z_i
Recurrent State
h_i
Continuous-space Word Representation
s_i
1-of-K coding
w_i
Decoder
Encoder
e = (Economic, growth, has, slowed, down, in, recent, years, .)

Figure 2: Encoder-Decoder Model for NMT. From source[2]

dependencies in sentences as it gives more emphasis on lates words seen. Also this suffers from problem of vanishing gradient and exploding gradient while training. LSTM (Hochreiter and Schmidhuber, 1997) does not store every information of sentence in state, rather it learns to selectively forget and remember. This is achieved by *gating* mechanism. It has three gates, and a memory cell called candidate memory. Gates helps selectively forget and remember information in sentence and contents are stored in memory cell. Structure of LSTM network is shown in figure 1. Components of LSTM layers are

1. **Forget Gate (f_t):** It decides which information to forget from previous memory cell.

2. **Input Gate (i_t):** It decides which information to pass to remember by cell.

3. **Candidate Memory ($\acute{C}_t$):** It records the information about current input.

4. **Output Gate (o_t):** Cell memory is squashed to lie between -1 and 1 using $tanh$ function for the computation of current hidden state. Output gate controls what to output in hidden state at this point.

5. **Cell Memory (C_t):** It is actual cell memory updated after adding necessary information and removing unnecessary information.

[1]http://colah.github.io/posts/2015-08-Understanding-LSTMs/

So when an input x_t is given as input at time t hidden state of RNN is computed as

$$h_t = f(h_{t-1}, x_t) \tag{1}$$

For LSTM non-linear function $f(h_{t-1}, x_t)$ is computed as follow

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f) \tag{2}$$

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i) \tag{3}$$

$$\acute{C}_t = \sigma(W_C h_{t-1} + U_C x_t + b_C) \tag{4}$$

$$C_t = f_t C_{t-1} + i_t \acute{C}_t \tag{5}$$

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o) \tag{6}$$

$$h_t = o_t tanh(C_t) \tag{7}$$

2.2 Basic Encoder-Decoder model

This model can be divided into two parts *Encoder* and *Decoder* as shown in figure 2, both are implemented using RNNs. Encoder encodes the variable length sentence into fixed length vector called *summary vector* or *context vector*. Decoder takes this vector representation of sentence and generates target language translation.

Encoder: Encoder is an RNN whose state is updated each time it sees a word in sentence and the last state of it summarizes whole sentence which is called as summary vector h_T. Processes of encoding of a word is as follow

Step 1: Input word at any point of time while encoding sentence, is input to the encoder as *one-hot* vector w_i.

Step 2: Now *one-hot* vector of input vector w_i is transformed to low-dimension continuous space vector representation s_i. To do so, we can use previous learned word embeddings $\mathbf{E}$ or train them jointly. Word embedding matrix $\mathbf{E} \in R^{d \times V}$ contains as many coloumns as words in vocabulary. Each i^{th} column of word embedding matrix represent continuous vector space representation of i^{th} word of vocabulary. So when as a product of word embedding matrix and one-hot vector, continuous space vector of corresponding word is selected as shown in equation 8.

$$s_i = Ew_i \tag{8}$$

Step 3: In this step, RNNs hidden is updated to take into account new word s_i seen in the sentence.

$$h_i = f(s_i, h_{i-1}) \tag{9}$$

Where f is non-linear transformation function of RNN depending on variant of RNN (Vanilla, LSTM and GRU) used.

After processing last word of the sentence (T-th word if sentence length is T), state h_T that is obtained by encoder is called *summary vector* of sentence, which is a fixed dimensional vector representing whole sentence.

Decoder: Decoder is also an RNN, that takes input as summary vector, previous generated target word and last hidden state of it. After processing input, probability distribution over words in target language vocabulary is obtained. Target words are then sampled from this probability distribution. Process of decoding is described below:

Step 1: First internal state z_i of decoder RNN is calculated as

$$z_i = f'(h_T, z_{i-1}, u_{i-1}) \tag{10}$$

Where z_i and z_{i-1} are current and previous state of decoder, h_T is summary vector, u_{i-1} is previous generated target word. f' is non-linear transformation function of RNN.

Step 2: Based on the current state of the decoder, we compute the compatibility score for each word in vocabulary, later transform this score into probability.

$$e_k = w_k^T z_i \tag{11}$$

$$p(w_i = k | w_1, w_2, ..., w_{i-1}, h_T) = \frac{exp(e_k)}{\sum_j (exp(e_j))} \tag{12}$$

Here e_k, w_k are score and vector representation of k-th word in vocabulary. This score is high if it aligns with decoder well else low. $p(w_i = k | w_1, w_2, ..., w_{i-1}, h_T)$ is the probability of k-th word, for all $k \in 1, 2, ..., V$.

Step 3: From the probability distribution obtained from step 2, we sample target word. Decoder then again repeats steps 1 to step 3 until an *end-of-sentence* is not encountered. Hence a target sentence is generated corresponding to provided input source sentence.

$$e_k = w_k^T z_i \tag{13}$$

$$p(w_i = k | w_1, w_2, ..., w_{i-1}, h_T) = \frac{exp(e_k)}{\sum_j (exp(e_j))} \tag{14}$$

Here e_k, w_k are score and vector representation of k-th word in vocabulary. This score is high if it aligns with decoder well else low. $p(w_i = k | w_1, w_2, ..., w_{i-1}, h_T)$ is the probability of k-th word, for all $k \in 1, 2, ..., V$.

2.3 Encoder-Decoder with Attention Mechanism

In basic encoder-decoder model, encoder compresses the sentence into fixed length vector. This summary vector contains the information of all the words in sentence. But as the length of sentence

[2] https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-2/

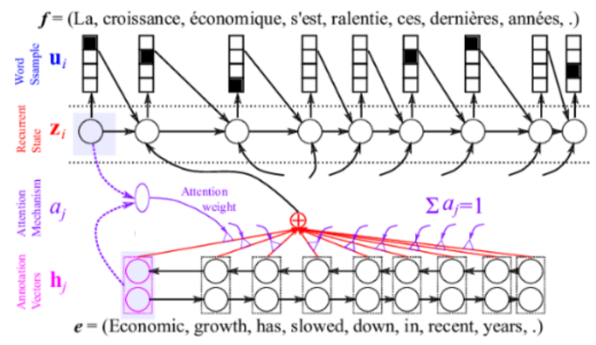

f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Word Ssample
u_i
Recurrent State
z_i
Attention Mechanism
a_j
Attention weight
Σa_j=1
Annotation Vectors
h_j
e = (Economic, growth, has, slowed, down, in, recent, years, .)

Figure 3: Attention based NMT architecture. source[1]

it fails to encode it efficiently in fixed summary vector, which degrades the performance of translation (Cho et al., 2014b). So in order to deal with this problem (Bahdanau et al., 2014) proposed soft-search model, that uses attention mechanism in encoder-decoder model. Idea in this model is to not represent sentence with fixed length vector rather represent each word by a fixed length vector called *annotation* vectors and while generating each word in target language look for source sentences which are more relevant in source sentence.

Only difference with basic encoder-decoder is in encoder part, decoder part is same in both the model. In basic model decoder takes encoded summary vector as input, but in attention model it takes ***context vector*** as input to generate target sentence.

Context vector is the convex combination of annotation vectors of words in source sentence. It is calculated each time decoder generates a new word, while in basic model summary vector used to calculate only once. Convex coefficient used in computation of context vector are called ***attention weights***. Architecture of attention based NMT model is shown in figure 3. **Annotation vectors:** Annotation vector for each word is calculated by *bi-directional* RNN. Bidirectional RNN reads sentence from both directions i.e. left-to-right and right-to-left. State from left-to-right $\overrightarrow{h}_i$ and right-to-left $\overleftarrow{h}_i$ is concatenated for each word, this concatenated state is called annotation vector h_i.

[1]https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/

Attention Weights: Attention weights gives *soft-alignment*, as these represent the probabilistic alignment of how words in source language and target language are aligned while generating target words. These are calculated each time decoder generate target word, and gives a probabilistic measure of how much each word in source language is important in the generation of current target word. To compute attention weights, first ***alignment score*** $e_{i,j}$ of each source word is computed that measures how relevant j-th word in source sentence is for i-th target word. This is computed for all source words in the generation of each target word by some alignment model **a** i.e.

$$e_{i,j} = a(z_{i-1}, h_j) \quad \forall j \in 1,2,...,T, \forall i \in 1,2,...,T' \tag{15}$$

where h_j is the annotation vector of j-th source word, T and T' are the lenghts of source and target sentences respectively and z_{i-1} is decoders last state. Then these alignment scores are transformed to probabilistic measure using softmax function. These probabilistic measures are called attention weights α_{ij}.

$$\alpha_{ij} = \frac{exp(e_j)}{\sum_{j'} exp(e_{j'})} \tag{16}$$

Since attention weights are probabilistic measures, we compute ***context vector*** as expected annotation vector by equation 17.

$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j \tag{17}$$

Here c_i is the context vector obtained while generating i-th target word, T is the lenght of source sentence, α_{ij} is attention weight of j-th annotation vector and h_j is j-th annotation vector.
Once context vector is computed we compute decoders next state as a non-linear function (as of LSTM or GRU) of context vector c_i, previous target word u_{i-1} and decoders last state z_{i-1} as shown in equation 18

$$z_i = f(c_i, u_{i-1}, z_{i-1}) \tag{18}$$

After that we compute probability distribution over target vocabulary and sample target word in the same way as done basic encoder-decoder model repeatedly till complete sentence is not generated.

3 Subword NMT

NMT systems are trained on a limited size vocabulary, but test data can have different words than those in vocabulary such words are called unseen, rare or out-of-vocabulary words. Most of the out-of-vocabulary words are named entities, compound words and cognates (via morphological transformation) (Sennrich et al., 2015).

Named entities can be copied to target language translation if it shares the alphabets with source language, else transliteration is required. For Cognates and loan words character level translation rules are sufficient. Translation of compound words can be achieved by translating its morphemes separately (Sennrich et al., 2015).

3.1 Byte-Pair Encoding

(Sennrich et al., 2015) proposed BPE based word segmentation method. In this method two vocabularies are maintained called training vocabulary and symbol vocabulary. Words in training vocabulary are represented as sequence of characters, plus an end-of-word symbol. All characters are added to symbol vocabulary. Then using BPE technique the most frequent symbol pair is identified, and all its occurrences are merged, producing a new symbol that is added to the vocabulary. This BPE step is repeated until a set of merge operations have been learned. Number of BPE merge operations in this method is also a hyper parameter.

Byte-Pair Encoding (BPE)[1] (Sennrich et al., 2015) is originally a data compression technique (Gage, 1994). Idea behind BPE is

"Find the most frequent pair of consecutive two character codes in the text, and then substitute an unused code for the occurrences of the pair." (Shibata et al., 1999)

Below example explains the BPE method:

Let the original text be
T_0 = PQPQRSUQSUVPQSUPQR.

Most frequent pair in T_0 is PQ, so we replace it by A. Modified Text is
T_1 = AARSUQSUVASUAR.

[1]https://github.com/rsennrich/subword-nmt

Now most frequent pair in T_1 is SU, so we replace it by B. Modified Text is
T_2 = AARBQBVABAR.

Now most frequent pair in T_2 is AR, so we replace it by C. Modified Text is
T_3 = ACBQBVABC.

In T_3 no pair is repeated so BPE algorithm stops here. Using BPE algorithm text of length $|T_0| = 18$ is compressed to a text of length $|T_3| = 9$. Additional information that is required to decode encoded text is list of encodings i.e. PQ $\rightarrow$ A, SU $\rightarrow$ B and AR $\rightarrow$ C. Since we perform BPE merge operations on character level first, this performs character level segmentation. As the number of merge operations are increased frequent sequence of characters and even full words are also encoded as a single symbol. This a allows a trade-off between the NMT model vocabulary size and the length of training sequence.

If there will be much larger merge operations then almost every word will belong to symbol vocabulary, that will prevent the sub-word level segmentation of words. When using BPE for sub-word segmentation, size of the sentences is increased as sub-words are separated by special symbols to allow decoding later. Larger the sentence size, it becomes difficult for NMT to learn well on them. So number of BPE merge operations is an important hyper parameter that is needed to be tuned properly before use.

4 Factored NMT

(Sennrich and Haddow, 2016) proposed an approach to incorporate linguistic features as input with words in NMT. They idea behind putting linguistic features is to examine if NMT system can learn linguistic features from parallel corpora and if this is not the case, then does it help to improve NMT systems. Their results showed improvement with linguistic features. Unlike in factored SMT (Koehn and Hoang, 2007), here linguistic features are used only at source side.

4.1 Linguistic Features in NMT

There are various linguistic features such as *lemma*, *POS tags*, *dependency parsing labels* and *subword tags* etc. which can be incorporated in

NMT systems at the source side very easily. How these features are incorporated in NMT system is described in detail in section 4.2. Following are the key idea behind using some specific linguistic features in NMT system.

1. **Lemma:** Providing lemmas as input features can help to improve system, over rare words, or morphological different words. For example if the word *happier* is already learned by the system along with its lemma and *happily* is never seen by the system, then their common lemma *happy* can help to place this word close to *happier* and some meaning can be drawn from it.

2. **Subword tags:** This can help system to understand that two subwords of a word are not independent and give extra information about actual boundary of whole word. So this way while translation system should consider all the subwords. These subwords tags are of four types *viz. beginning (B), inside (I), end (E) and whole word (O).*

3. **POS tags:** These can help in disambiguation, when same word can have different meanings and/or semantics as per the sentence.

For extracting linguistic features for English language *Stanford CoreNLP* (Manning et al., 2014) can be used.

4.2 Representation of Linguistic Features in NMT

NMT systems take the continuous space representation of words, so to incorporate linguistic features into it their continuous space representation or embeddings have to feed in it. In subword NMT linguistic information is represented at subword level, and all the subwords of a word share the same linguistic features except subword tags which will be explained later in this section.

All the linguistic information is provided at encoder part along with word embedding in the form of concatenated embeddings of features. Rest of the architecture remains same as of attention based encoder-decoder NMT systems.

Each subword/word is represented as a one-hot notation x_i which then gets transformed to continuous space vector s_i using equation 8 as shown below, where E is the word embedding matrix.

$$s_i = Ew_i$$

With linguistic features, each linguistic feature is initially represented by one-hot encoding which then transformed to continuous space representation using equation 8, but with their own embedding matrix and finally gets concatenated to represent final embedding of input word/subword along with linguistic information as shown in equation 19.

$$s_i = \|_{k=1}^{K} E_k f_k^i \tag{19}$$

Where $\|$ is the vector concatenation operator, E_k is the matrix that contains word embeddings of k^{th} features and f_k^i is the k^{th} feature of i^{th} word/subword of the sentence. Rest of the whole architecture of attention based encoder-decoder is same as described in section 2.3.

4.3 Factored NMT Results

(Sennrich and Haddow, 2016) performed experiments on WMT'16 English-German corpus of approximately 4.2 million sentences. They found that input linguistic features help to improve performance of NMT systems. Their results are shown in figure 4.

	German→English					English→German				
system	ppl ↓	BLEU ↑		CHRF3 ↑		ppl ↓	BLEU ↑		CHRF3 ↑	
	dev	test15	test16	test15	test16	dev	test15	test16	test15	test16
baseline	47.3	27.9	31.4	54.0	58.0	54.9	23.0	27.8	52.6	56.0
all features	46.2	28.7*	32.9*	54.8	58.5	52.9	23.8*	28.4*	53.9	57.2

Figure 4: German-English translation results: best perplexity on dev (newstest2013), and BLEU and CHRF3 on test15 (newstest2015) and test16 (newstest2016). BLEU scores that are significantly different (p ¡ 0.05) from respective baseline are marked with (*).source[1]

5 Conclusion

We have also gone through the neural machine translation approach. We realized the NMT model suffers from long sentence translation and rare or out-of-vocabulary words. The long sentence translation issue is addressed by attention based NMT model using LSTM activation units. Rare or Out-of-vocabulary words translation problem can be addressed using sub-word segmentation like Byte-pair Encoding (BPE). We explored factored NMT

[1](Sennrich and Haddow, 2016) `https://arxiv.org/pdf/1606.02892.pdf`

approach which makes use of linguistic information of sentence, and realized it also improves the performance of NMT systems.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR* abs/1409.0473. http://arxiv.org/abs/1409.0473.

KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014a. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR* abs/1409.1259. http://arxiv.org/abs/1409.1259.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014b. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* .

Philip Gage. 1994. A new algorithm for data compression. *C Users J.* 12(2):23–38. http://dl.acm.org/citation.cfm?id=177910.177914.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.* 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. Association for Computational Linguistics, Seattle.

Philipp Koehn and Hieu Hoang. 2007. Factored translation models. In *EMNLP-CoNLL*. pages 868–876.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for Computational Linguistics, pages 48–54.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*. pages 55–60. http://www.aclweb.org/anthology/P/P14/P14-5010.

Rico Sennrich and Barry Haddow. 2016. Linguistic input features improve neural machine translation. *CoRR* abs/1606.02892. http://arxiv.org/abs/1606.02892.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *CoRR* abs/1508.07909. http://arxiv.org/abs/1508.07909.

Yusuke Shibata, Takuya Kida, Shuichi Fukamachi, Masayuki Takeda, Ayumi Shinohara, Takeshi Shinohara, and Setsuo Arikawa. 1999. Byte pair encoding: A text compression scheme that accelerates pattern matching.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. *CoRR* abs/1409.3215. http://arxiv.org/abs/1409.3215.