Comparison of Embedding Methods and Model Architectures for French-to-English Machine Translation

Travis Twigg,[1] Dmitry Pankratov,[1] Rahul Gite,[1] Nikhil Goparapu,[1] Abhiram Kalidindi,[1] and Tony Diana[2]
[1] *Data Science Graduate program, University of Maryland, Baltimore County, MD, USA.*
[2] *Policy Sciences, University of Maryland, Baltimore County, MD, USA.*
(Dated: December 3, 2020)

ABSTRACT – Neural machine translation (NMT) is a recent approach to machine translation based on neural networks. NMT models are frequently composed of an encoder-decoder network. The encoder extracts a fixed-length representation from a variable-length input sentence, and the decoder generates a translation from this representation (Cho, 2014b)[1]. In this paper, we analyze the impact of an attention mechanism on an encoder-decoder network, compare the effectiveness of various embedding methods towards French-English translation, and develop a machine translation scoring method that incorporates syntactic and semantic similarities between sentences. We show that the addition of an attention mechanism has little effect on performance for medium length sequences, and that learned embeddings outperform pre-trained embeddings on a 10,000 sentence pair dataset.

I. Introduction

Machine translation (MT) is the automatic translation from one natural language to another using software. The origins of MT can be traced back to 1949, when Warren Weaver drafted the first set of proposals for computer based machine translation, only a couple of years after ENIAC was created (Weaver, W., 1955)[2]. The importance of language translation cannot be understated. There are more than 7000 languages that exist today in the world and the ability to connect with other cultures is essential. "It is an undeniable truth that in this era of globalization, language translation plays a vital role in communication among the denizens of different nation's" (Patel, 2020)[3].

There are three main approaches to machine translation: rule-based translation (RBMT), which relies on built-in linguistic rules and millions of bilingual dictionaries for each language pair; statistical machine translation (SMT), which uses statistical analysis and predictive algorithms to define rules that are best suited for target sentence translation; and neural machine translation (NMT), a recent approach that utilizes artificial neural networks to perform translation. One of the most popular model architectures for NMT is the encoder-decoder network and was proposed by Kalchbrenner and Sutskever in 2013 (Kalchbrenner, 2013)[4](Sutskever et al., 2014)[5].

The aim of this study is to analyze the impact of an attention mechanism on an encoder-decoder network, compare the effectiveness of various embedding methods towards French-English translation, and develop a machine translation scoring method that incorporates syntactic and semantic similarities between sentences. Our group hypothesizes that the addition of an attention mechanism will outperform a traditional encoder-decoder network, and the use of pre-trained embeddings will produce better results compared to learned embeddings. After a thorough literature review, we can confidently say this research is original, as no other work has compared learned embeddings and pre-trained embeddings for machine translation.

II. Methodology

The group applied two model architecture approaches, as well as five word embedding techniques to the task of machine translation. Four evaluation metrics were used to assess performance on the training set. All models were created using PyTorch v.1.7.0 with Python v.3.6.9.

A. Dataset

The dataset used in this study was sourced from Tatoeba.org. Tatoeba is a large database of example sentences translated into multiple languages by volunteers and is geared towards foreign language learning (Kelly, C., 2020)[6]. The Tatoeba text corpus has been used for a variety of natural language processing tasks, including machine translation (Tiedemann, 2020)[7]. The original dataset consists of 135,842 English-French sentence pairs. Since the project is volunteer driven, the dataset comes with some drawbacks for machine translation, as multiple translations exist for the same sentence, some sentences are grammatically inaccurate, and some translations are incorrect (Kelly C., 2020)[6]. Below is an example sentence pair from the dataset:

'I wish you'd let me use your car.', 'J'aurais aimé que tu me laisses prendre ta voiture.'

B. Preprocessing Methods

In order to prepare the data, several common NLP preprocessing steps were completed:

- Removal of french characters with accents by converting Unicode text to plain ASCII text.
- Lowercasing.
- Removal of extra whitespace.
- Replacement of punctuation with whitespace (with the exception of periods, commas, exclamation marks, and question marks).
- Tokenization.

Since the goal of this study was to train and compare multiple model architectures and word embeddings, we chose to subset the dataset to ensure quicker training times and allow for more experimentation. The original dataset consists of 135,842 English-French sentence pairs. The dataset was trimmed to 10,599 sentence pairs, 7.8% of the original dataset, by filtering out sentences with more than 10 words and sentences that do not start with an English prefix ('I'm', 'He's', 'She's', 'You're', 'We're', 'They're').

C. Model Architectures

For the task of machine translation, two model architectures were utilized for the study. Both models were sequence-to-sequence encoder-decoder networks with gated recurrent units (GRU) and employed teacher forcing. The difference between the two was the use of an attention mechanism.

1. Seq2Seq

Sequence to sequence (Seq2seq) is a family of encoder-decoder approaches used in many language processing tasks such as language translation, image captioning, and text summarization. Seq2seq was first introduced by Google for machine translation in 2014 (Sutskever, I, Oriol Vinyals, and Quoc Le., 2014)[5]. Seq2seq learning implies training an encoder and decoder model to map sequences from one domain (French language in our case) to sequences in another domain (English language). Each model consists of a Recurrent Neural Network (RNN), GRU in our case, and is a powerful tool in language processing as it can map sequences of different lengths to each other (Kostadinov, S. 2019)[8].

The encoder receives a variable-length input sequence and encodes the information into a fixed-length context vector (final hidden state) which aims to encapsulate the information of all input elements. The decoder initializes its hidden state from the context vector and starts to generate the output sequences. At each time step, a recurrent unit accepts the hidden state from the previous unit and produces a prediction and a new hidden state.

2. GRUs

As mentioned earlier, the encoder-decoder models utilize Gated Recurrent Units (GRU) as the RNN of choice. GRUs, proposed by Cho et al in 2014 (Cho K., 2014)[9], solves the long term memory problem of RNNs, known as the vanishing gradient problem, by using update and reset gates to regulate the flow of information through the network. These gates are vectors which decide what information should be passed to the output. The update gate decides what information to throw away and what new information to add. The reset gate is used to decide how much past information to forget (Phi, M., 2020)[10].

GRUs utilize activation functions within their gates to regulate the values flowing through the network. Vectors undergo many transformations as they move through the network, sometimes leading to very large values and undue significance. The tanh activation function is used to control this by limiting the range of values between -1 and 1. A sigmoid activation function, which limits values between 0 and 1, helps to update or forget data since any vector multiplied by 0 is 0 (forgotten) and any vector multiplied by 1 is 1 (unchanged). An advantage of GRUs is that the gates can be trained to keep information from much earlier in an input sequence. GRUs were chosen over LSTMs, as GRUs are typically faster to train and perform better on less training data for language modeling problems (Gautam, D., 2019)[11].

3. Teacher Forcing

Sequence prediction models typically use the output from the last time step as input for the current time step. This can lead to slow convergence, instability, and poor results while training (Brownlee, J., 2019)[12]. Teacher forcing, developed by Williams et al., is an efficient strategy to overcome this issue by using the ground truth from a prior time step as input for the current step in a RNN (Williams, R., 1989)[13]. The approach was originally described and developed as an alternative technique to backpropagation through time for training a RNN. It is an old technique but still it is widely used for sequential problems today.

4. Attention

The encoder RNN converts the informational content of the input sequence into a fixed size content vector. As an input sequence increases in length, this fixed size content vector struggles to summarize all parts of the input sequence and tends to "forget" earlier parts, creating a bottleneck in performance (NLP From Scratch, 2017)[14].

Bahdanau et al. proposed the idea of attention in 2015. They found that an encoder-decoder model with attention significantly outperformed conventional models for any input length and was much more robust to the length of the input sequence (Bahdanau, D., 2015)[15].

An attention mechanism helps the decoder learn to focus over a specific range of the input sequence, instead of the whole sequence. The learned attention weights are concatenated with the encoder's hidden states and sent to the decoder with the context vector to predict the next output sequence. Before attention mechanisms

were adopted, encoder-decoder networks could only reliably be used with short input sequences (Antonio, M., 2019)[16]. Attention also helps address a common issue in machine translation, word alignment, as words can sometimes be arranged differently depending on the language. An example is: *'european economic area' and 'zone économique européenne.'*

D. Word Embeddings

A word embedding is a learned representation of text where words that have a similar meaning have a similar representation. Each word is represented by a real-valued vector in a high dimensional space. The distributed representation is learned based on the usage of words. This allows words that are used in similar ways to result in having similar representations, naturally capturing their meaning. Word embeddings have increased performance for numerous downstream language processing tasks including sentiment analysis, information retrieval, and question answering (Brownlee, J., 2019)[12]. In our study, we utilized two word embedding techniques: learned embeddings and pre-trained embeddings. The pre-trained embeddings include GloVe, Word2Vec, frWac, and FastText.

1. Learned Embeddings

PyTorch's embedding layer creates a randomized matrix of size (Vocabulary Size x Embeddings Dimension) to act as a look-up table, where a word assigned to index i has its embedding stored in the *ith* row of the matrix (PyTorch Tutorials 1.7.0 documentation, 2020). As the model trains, the embeddings are updated to capture the relationship between words. Learned embeddings trained on a large enough dataset tend to perform better for a specific task compared to a more generalized pre-trained embedding[17].

2. frWac

For our models that utilized pre-trained word embeddings, French words were embedded using frWac2Vec (except the FastText study), a pre-trained word2vec model developed by Jean-Philippe Fauconnier at Apple. frWac2Vec is a 200 dimensional word embedding model created using the continuous skip-gram model architecture, and was trained on the frWac corpus, a 1.6 billion word corpus constructed from a web crawl of the .fr domain. frWac2Vec has been cited in over 30 natural language processing research papers to date (Fauconnier, 2020)[18].

3. Word2Vec

Word2Vec, developed by Mikolov, et al. at Google in 2013, is a statistical method for efficiently learning word embeddings from a text corpus. Word2Vec introduced two unique learning models to learn embeddings: the Continuous Bag-of-Words model (CBOW) and the Continuous Skip-Gram model. The CBOW model learns the embedding by predicting the target word based on its context (sum of surrounding words). The Continuous Skip-Gram model learns by predicting the surrounding words given the distributed representation of the current word. The CBOW model is many times faster to train compared to the skip-gram model, and has better accuracy for words with a higher frequency. The skip-gram model performs better with smaller datasets and better encodes the meaning of infrequent words (Brownlee, J., 2019)[12]. The Word2Vec embeddings used in this study are 300-dimensional vectors representing 3 million words trained on the Google News Dataset.

4. GloVe

The Global Vectors for Word Representation algorithm (GloVe), developed by Pennington, et al. at Stanford, is an extension to the word2vec method for efficiently learning word vectors. GloVe is a distributed model for word representation and an unsupervised learning algorithm that is used for obtaining vector representations of words. GloVe combines the power of matrix factorization techniques (such as LSA) at using global text statistics (word co-occurrence) with Word2Vec's ability to use local statistics (local context information of words) to create vector space model representations (Brownlee, J., 2019)[12]. The GloVe embeddings used in this study are 100-dimensional vectors representing 400,000 english uncased words and were trained on Wikipedia 2014 and Gigaword 5. (Pennington, J., 2020)[19]

5. Fasttext

FastText is an extension to Word2Vec, developed by Facebook in 2016. Instead of feeding individual words into the model FastText breaks words into several n-grams (sub-words). The word embedding vector for a target word will be the sum of all n-grams. FastText supports training CBOW or Continuous Skip-Gram models (FastText, 2020)[20]. The FastText embeddings used in this study are 300-dimensional vectors representing 1 millions words trained on Wikipedia 2017, UMBC domain corpus, and statmt.org news dataset. FastText embeddings were used for both the English and French embeddings in the FastText study.

E. Metrics

Four evaluation metrics were utilized to test the performance and robustness of our models: BLEU score, GLEU score, a custom scoring method, and comparison against Google Translate.

1. BLEU

BLEU or Bilingual Evaluation Understudy is a "measurement of the differences between a machine translation and one or more human-created reference translations of the same source sentence" (Microsoft.com, 2020)[21].

The BLEU algorithm compares consecutive phrases of a machine translation output with the consecutive phrases of a reference translation, and counts the num-

ber of matches. The matches are position independent and a higher match degree indicates a higher degree of similarity with the reference translation and a higher score. BLEU calculates the average precision for n-grams where n goes from 1 to 4 and adds a brevity penalty for short translations. The BLEU score ranges between 0 and 1. For our methods, we utilized only uni-gram (single word) BLEU scores due to the shortness of sentences used. Higher n-grams produce perfect BLEU scores for sentences that are too short, and biases scores towards perfect values. Drawbacks include:

- Does not consider the meaning of a word.
- Does not consider sentence structure.
- It cannot deal with languages lacking word boundaries.
- Has some undesirable properties when used for single sentences, as it was designed to be a corpus measure.

2. GLEU

The GLEU or Google-BLEU is a variant of the BLEU metric. GLEU calculates the precision and recall for n-grams from uni-gram to four-gram and returns the minimum value. Recall is defined as the ratio of the number of matching n-grams to the number of total n-grams in the target (ground truth) sequence. Precision is defined as the ratio of the number of matching n-grams to the number of total n-grams in the generated output sequence. The GLEU score is the minimum of recall and precision. Like BLEU, it ranges between 0 and 1 (Wu, Schuster, 2016)[22]. Compared to BLEU, it approaches human judgments more closely, but can overly penalize for altered word order. For our methods, we utilized only uni-gram (single word) GLEU scores due to the shortness of sentences used. Higher n-grams produce perfect GLEU scores for sentences that are too short, and biases scores towards perfect values. Table I illustrates the difference between BLEU and GLEU scores on a sample sentence. Drawbacks include:

- It cannot deal with languages lacking word boundaries.
- It does not consider the meaning of a word.
- Tends to overly penalize for altered word order or where two words are represented as one: e.g. *'unconvinced'* vs *'not convinced.'*

Input:	je ne suis pas du tout fatigue .
Target:	i am not at all tired .
Prediction:	i am not tired at all .
BLEU Score:	1.000
GLEU Score:	0.556

TABLE I: Example of BLEU and GLEU scores on a sample sentence.

3. Custom Scoring Method

Since BLEU and GLEU scores only compare sentences lexically, they sometimes penalize semantically correct translations (Fauconnier, 2020)[18]. See Table II. In order to evaluate the semantic similarity between sentences and to address specific issues with our models, we developed a custom scoring method that not only takes BLEU and GLEU into account, but also utilizes the cosine similarity between mismatched words and penalizes duplicate word prediction. Like BLEU and GLEU, the custom score ranges from 0 to 1. See Equation 1 for the custom score equation. See Table III for an example of scoring a sample translation.

Custom Score = Weighted Average(BLEU+GLEU) + Weighted Semantic Bonus - Double Word Penalty.
Equation 1.

Input:	je suis content de te voir .
Target:	i am happy to see you .
Prediction:	i am glad to see you .
BLEU Score:	0.83
GLEU Score:	0.50

TABLE II: Example of semantically correct prediction and its respective BLEU and GLEU scores.

Input:	vous etes tous fous .
Target:	you are all mad .
Prediction:	you are all crazy .
BLEU Score:	0.750
GLEU Score:	0.600
Avg Score:	0.713
Semantic similarities:	['mad', 'crazy', 0.7385839]
Semantic similarity bonus:	+ 0.22157517671585084
Double word penalty:	- 0.0
Custom Score:	0.934

TABLE III: Custom scoring on sample translation.

Below are the steps involved in the custom scoring method:

1. The weighted average of the BLEU and GLEU scores was calculated. BLEU scores were weighted more in the average because the authors felt that GLEU overly penalized situations where two words were predicted for one: e.g. *'unconvinced'* vs *'not convinced.'*

   Weighted average = 0.75BS + 0.25GS.

2. Mismatched words that are not stopwords were computed. Stop words chosen for filtering were: 'a', 'an', 'of', 'the', 'to', 'on', 'in', 'as.'

3. The cosine similarity between mismatched words was calculated using word2Vec. After testing, we found that word2Vec pretrained vectors produced the best cosine similarities between words. Other models produced high values when comparing unrelated words such as *'not'* and *'author'*.

4. A semantic similarity bonus was calculated for the number of matches with a cosine similarity score above 0.3. The cosine similarity score threshold was chosen to be 0.3 based on the observation that word pairs with scores over this could be switched and would not affect the overall meaning of the sentence.

5. The semantic similarity bonus was weighted based on how close the weighted average from step 1 was to 1.0. This ensured custom scores did not go over 1.

6. A double-word-penalty was calculated and applied when the model duplicates a word in its prediction. This was a consistent failure point for our learned embedding models. See Table IV for an example of duplicate word prediction.

Target:	she is wearing a nice hat .
Prediction:	she is wearing a hat hat .

TABLE IV: Duplicate word prediction.

4. Google translate API

The Google Translation API provides a simple interface for dynamically translating an arbitrary string into any supported language using state-of-the-art neural machine translation. It can also be used to detect a language in cases where the source language is unknown. At the moment of writing this paper, Google utilizes a transformer encoder and a RNN decoder architecture (Caswell, 2020)[23]. The model is trained using examples of translated sentences and documents, which are typically collected from the public web. The use of Google Translate as an evaluation tool gave us the ability to compare our model to a state-of-the-art architecture. Drawbacks include:

- Free testing version of Google translate API limits the number of translations per day.

III. Results

In our study, we found that the addition of Attention into the encoder-decoder model architecture showed no significant improvement over models without. Alternatively, we found that performance varied significantly based on the word embedding method. Table V displays evaluation results for each model. The models that utilized learned embeddings instead of pre-trained embeddings performed the best, even outperforming Google Translate in every metric. Out of the models that utilized pre-trained word embeddings, GloVe and FastText performed similarly well, while Word2Vec had the worst performance. See Table VI for sample translations.

The models that utilized learned embeddings took much longer to train (2500 epochs) compared to the models that used pre-trained embeddings. The pre-trained embedding models were able to produce reasonable results within only 30 epochs of training, while the learned embedding models took closer to 1000 epochs to produce reasonable results. See FIG. 4, 5 for loss curves of all models.

Embedding Method	*Avg BLEU*	*Avg GLEU*	*Avg Custom Score*	*Rank*
With Attention:				
Learned Embeddings	0.980	0.967	0.984	2
Without Attention:				
Learned Embeddings	0.984	0.973	0.988	1
Glove + frWac2Vec	0.764	0.676	0.815	4
FastText (Eng + Fra)	0.755	0.670	0.803	5
Word2Vec + frWac2Vec	0.710	0.604	0.764	6
Open Source Model				
Google Translate API	0.851	0.751	0.905	3

TABLE V: Comparison of embedding methods on evaluation metrics.

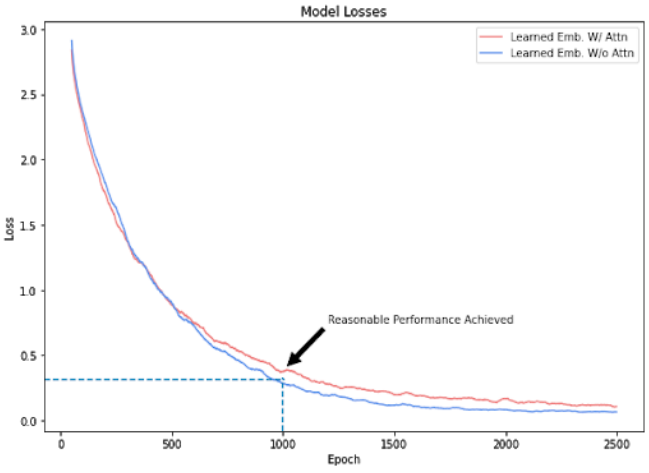

FIG. 1: Loss curves for learned embedding models.

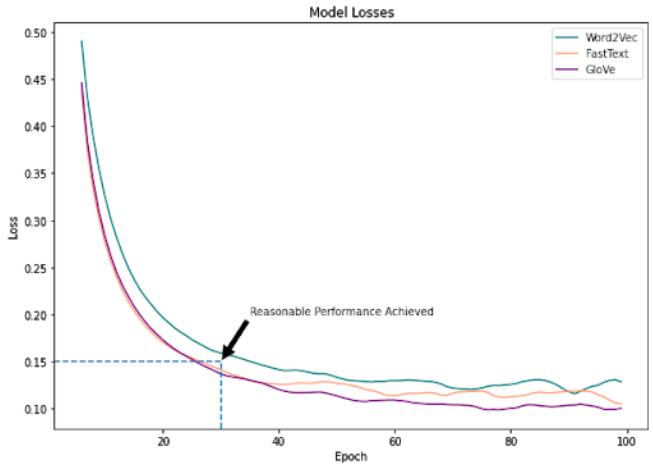

FIG. 2: Loss curves for pre-trained embedding models.

Input:	vous etes deux fois plus forts que moi .
Target:	you are twice as strong as i am .
GloVe:	you are twice as strong as i am .
Word2Vec:	you are twice as strong as me .
FastText:	you are twice as strong as me .
Learned Emb.:	you are twice as strong as i .
Input:	je cherche du travail .
Target:	i am looking for a job .
GloVe:	i am looking for a job .
Word2Vec:	i am looking for a job .
FastText:	i am looking for a job .
Learned Emb.:	i am looking for work .
Input:	nous en avons termine .
Target:	we are all done .
GloVe:	we are through .
Word2Vec:	we are done .
FastText:	we are finished already .
Learned Emb:	we are finished .

TABLE VI: Sample translations.

IV. Discussion

In this study, we aimed to compare the performance of different model architectures and embedding methods for machine translation. The research group hypothesized that the addition of an attention mechanism would increase translation performance and that pre-trained word embeddings would outperform learned embeddings.

The results showed that the addition of an attention mechanism did not significantly improve translation performance but did lower the efficiency of the model (time per epoch). This was unexpected and is believed to have occurred due to the maximum length requirement placed on input sentences during the pre-processing stage of the dataset. Since sentences with less than 10 words were used as input, the non-attention encoder-decoder network did not struggle to encode the context vector with information from the full sequence. We hypothesize that the encoder-decoder model with attention would outperform the standard encoder-decoder on longer sentence input.

The results also show that the models which utilized learned embeddings outperformed the models which utilized pre-trained word embeddings. This too was unexpected, as research has shown that pre-trained embeddings trained on a large corpus considerably outperform learned embeddings in both precision and recall (Farahmand, M., 2019)[17]. These results could be due to the size of the training dataset (10,599 sentence pairs), as research has shown that pre-trained embeddings are most effective in scenarios where there is "very little training data but not so little that the system cannot be trained at all" (Qi, Y., 2017)[24]. GloVe and FastText embeddings outperformed Word2Vec embeddings for translation in the study. This was expected, as GloVe and FastText are improvements upon the Word2Vec model. The most unexpected finding was that the learned embedding models outperformed the Google Translate API, which is considered a state-of-the-art platform.

1. *Lessons learned*

From the study, the research group has gained a better understanding of encoder-decoder networks, translation evaluation schemes, and word embedding methods. This research has shown that pre-trained word embeddings, as useful as they are in many circumstances, aren't always the best method for natural language processing tasks. Another lesson from this research is that decisions concerning choice of dataset and data processing steps can have a large impact on results and should be taken into consideration when selecting model architecture.

2. *Challenges*

The research group faced two main challenges in this study. First, a lack of knowledge in the French language made preprocessing decisions difficult, particularly around contractions. Second, developing an evaluation method that captures the syntactic and semantic similarities between translated sentences and references proved to be difficult and requires future research.

3. *Future work*

Future research should include exploring the impact of fine-tuning pre-trained embeddings on the training dataset, incorporating POS tagging into the learning and evaluation process, exploring semantic evaluation metrics, and exploring other architectures such as transformers.

[1] K. Cho, "On the properties of neural machine translation: Encoder–decoder." https://arxiv.org/pdf/1409.1259.pdf (2014).

[2] W. Weaver, "Translation. in machine translation of languages: Fourteen essays, w. n. locke and a. d. booth, eds." MIT Press, Chapter 1, 15–23. (1955).

[3] H Patel, "Neural machine translation (nmt) with attention mechanism." towardsdatascience.com/neural-machine-translation-nmt-with-attention-mechanism-5e59b57bd2ac (2020).

[4] Blunsom P. Kalchbrenner N., "Two recurrent continuous translation models. in proceedings of the acl conference on empirical methods in natural language processing (emnlp), pages 1700–1709. association for computational linguistics." (2013).

[5] Quoc Le. Sutskever I, Oriol Vinyals, "Sequence to sequence learning with neural networks. in advances in neural information processing systems (nips 2014)," (2014).

[6] C. Kelly, "Bilingual sentence pairs from the tatoeba project," www.manythings.org/bilingual (2020).

[7] J. Tiedemann, "The tatoeba translation challenge – realistic data sets for low resource and multilingual mt." arxiv.org/pdf/2010.06354.pdf (2020).

[8] S. Kostadinov, "Understanding encoder-decoder sequence to sequence model," towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346 (2019).

[9] K. Cho, "Learning phrase representations using rnn encoder–decoder for statistical machine translation. acl anthology," arxiv.org/pdf/1406.1078v3.pdf (2014).

[10] M. Phi, "Illustrated guide to lstm's and gru's: A step by step explanation." towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21 (2020).

[11] D. Gautam, "An overview of rnn lstm gru - voice tech podcast." medium.com/voice-tech-podcast/an-overview-of-rnn-lstm-gru-79ed642751c6 (2019).

[12] J. Brownlee, "A gentle introduction to calculating the bleu score for text in python. machine learning mastery," machinelearningmastery.com/calculate-bleu-score-for-text-python/ (2019).

[13] R. Williams, "A learning algorithm for continually running fully recurrent neural networks - mit press journals and magazine." Ieeexplore.Ieee.Org. ieeexplore.ieee.org/document/6795228 (1989).

[14] PyTorch Tutorials 1.7.0 documentation., "Nlp from scratch: Translation with a sequence to sequence network and attention," (2017).

[15] D. Bahdanau, "Neural machine translation by jointly learning to align and translate," arxiv.org/pdf/1409.0473.pdf (2015).

[16] M Antonio, "Attention mechanism in seq2seq and bidaf — an illustrated guide," towardsdatascience.com/the-definitive-guide-to-bidaf-part-3-attention-92352bbdcb07 (2019).

[17] M. Farahmand, "Pre-trained word embeddings or embedding layer? — a dilemma." https://towardsdatascience.com/pre-trained-word-embeddings-or-embedding-layer-a-dilemma-8406959fd76c (2019).

[18] J Fauconnier, "Jean-philippe fauconnier," fauconnier.github.io/index.html#wordembeddingmodels (2020).

[19] J. Pennington, "Glove: Global vectors for word representation." nlp.stanford.edu/projects/glove/ (2020).

[20] "Fasttext," fasttext.cc (2020).

[21] Microsoft.com, "What is a bleu score? - custom translator - azure cognitive services." docs.microsoft.com/en-us/azure/cognitive-services/translator/custom-translator/what-is-bleu-score (2020).

[22] Schuster M. Wu, Y., "Google's neural machine translation system: Bridging the gap between human and machine translation." arxiv.org/pdf/1609.08144.pdf (2016).

[23] I Caswell, "Recent advances in google translate. google ai blog," ai.googleblog.com/2020/06/recent-advances-in-google-translate.html (2020).

[24] Y. Qi, "When and why are pre-trained word embeddings useful for neural." https://www.aclweb.org/anthology/N18-2084.pdf (2017).