An Applied Evolutionary Analysis of Neural Machine Translation

Torre Elia
University of California - San Diego

March 18, 2022

Abstract

This paper presents the research undertaken in exploring three major leaps in the context of Neural Machine Translation. In order to do so, three milestone articles in the field of Neural Machine Translation are posed under analysis, i.e., "Sequence to Sequence Learning with Neural Networks" [1], "Neural Machine Translation by Jointly Learning to Align and Translate" [2] and "Attention Is All You Need" [3]. The architectures described in the articles are implemented from scratch in the notebook associated with this paper (Github) and their performances are evaluated according to BLEU score on the German to English translation task based on "Multi30K" dataset. [4]

I. Introduction

Machine translation (MT), is a branch of computational linguistics that deals with the translation of text or speech from one natural language to another. MT is a complex task that involves many different steps, including text analysis, language processing, and generation of the target language text. [5] In recent years, the automatic translation of text from one language to the other, has experienced a major shift. Statistical Machine Translation, which relies on count-based models and has dominated the field for decades, has now been substituted by Neural Machine Translation (NMT).[6] NMT is a neural network-based approach to machine translation that is used to automatically translate one natural language to another. NMT models are trained on large amounts of parallel text and can produce high-quality translations. The objective of this paper is that of analysing and evaluating the results of three of the major leaps in architecture developments in the context of NMT. The architectures described in the articles are implemented from scratch in the notebook associated with this paper (Github) and their performances are evaluated according to Perplexity metric and BLEU score. It is necessary to note the reader that the implementations will slightly differ from the ones described in the papers on two major aspects:

- Architecture: the models implemented in the associated notebook will, in some cases, differ in the depth of the network and weight initialization process, in the first case this is due to the available computational power, while in the second case is purposely done to follow more recent implementations of the architectures (e.g., BERT [7]) described in the articles.
- Dataset: The training and evaluation of the architectures in all of the three articles has been performed on a specific sub-sampling of the "WMT '14" French to English dataset [8]. On the contrary, again due to limitations in computational power and a seek for

novelty, the training and evaluation of the architectures implemented in this paper has been performed on the "Multi30K" German to English dataset. [4]

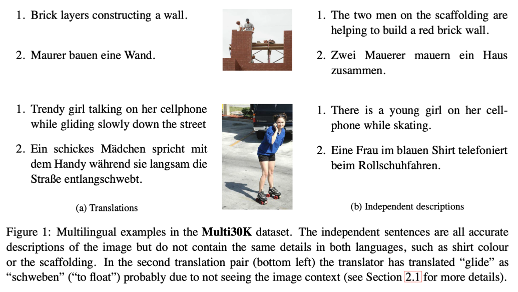
II. Dataset

The dataset used to perform the following analysis is, as previously mentioned, the "Multi30K" German to English dataset. [4]. The Multi30k dataset is a collection of 31,014 parallel English-German sentences that are used for training and evaluating neural machine translation models. The sentences are split into train, validation, and test sets, and the dataset also includes a human-annotated English-German parallel corpus. In the referenced article of 2016, the Multi30K dataset is introduced as an extension of the Flickr30K dataset, which has been developed as a dataset containing images sourced from online photo-sharing websites, each of which paired with five English descriptions, which were collected from Amazon Mechanical Turk1. In particular, the Multi30K dataset presents the following characteristics:

	English	German
Sentences	31,014	31,014
Tokens	357,172	333,833
Types	11,420	19,397
Characters	1,472,251	1,774,234
Avg. length	11.9	11.1
Singletons	5,073	11,285

III. Pre-Processing

The pre-processing steps performed on the "Multi30K" dataset are common to all of the three architectures, except for a slight variation in the implementation of the first architecture that will be further explained later. The pre-processing of the dataset can be summarized in four main steps:



1. Brick layers constructing a wall.
2. Maurer bauen eine Wand.

1. The two men on the scaffolding are helping to build a red brick wall.
2. Zwei Mauerer mauern ein Haus zusammen.

1. Trendy girl talking on her cellphone while gliding slowly down the street
2. Ein schickes Mädchen spricht mit dem Handy während sie langsam die Straße entlangschwebt.

1. There is a young girl on her cellphone while skating.
2. Eine Frau im blauen Shirt telefoniert beim Rollschuhfahren.

(a) Translations

(b) Independent descriptions

Figure 1: Multilingual examples in the **Multi30K** dataset. The independent sentences are all accurate descriptions of the image but do not contain the same details in both languages, such as shirt colour or the scaffolding. In the second translation pair (bottom left) the translator has translated “glide” as “schweben” (“to float”) probably due to not seeing the image context (see Section 2.1 for more details).

Figure 1: *Dataset content example from "Multi30K: Multilingual English-German Image Descriptions"*

- **Token**: the "spacy modules" for English and German are loaded and two functions (German and English) are defined with the purpose of exploiting spacy library to tokenize the input sequences. Then, pytorch's "Field" function is used to append <*SOS*> and <*EOS*> tokens and transform all of the words in the sentences to lowercase.
- **Split**: pytorch's "dataset split" function is used to the obtain a Training (29k instances), Validation (1k instances) and Test (1k instances) sets, furthermore it specifies German as the source language and English as the target.
- **Vocabulary**: the vocabularies for the two languages are created from the training data enforcing that only words which appear at least twice are included, otherwise an <*UNK*> token is used.
- **Iterator**: in the last step, an iterable object with source and target information that maps tokenized words to their index in the vocabulary is created through "BucketIterator" with a batch size of 128.

The difference in the pre-processing of the dataset for the first architecture consists in a rearranging of the sequence order in the source sentence. Indeed, the paper explains that the order of the source sequences has been reverted as this has empirically shown a better translation. The intuition behind this choice is explained in the paper as:

> [...] we found it extremely valuable to reverse the order of the words of the input sentence. So for example, instead of mapping the sentence *a,b,c* to the sentence α, β, γ, the LSTM is asked to map *c,b,a* to α, β, γ where α, β, γ is the translation of *a,b,c*. This way, *a* is in close proximity to α, *b* is fairly close to β, and so on, a fact that makes it easy for SGD to "establish communication" between the input and the output. We found this simple data transformation to greatly improve the performance of the LSTM. [1]

IV. Background

The steps preceding the publication of the papers under analysis involved *Feed Forward Neural Networks* (NN) and *Recurrent Neural Networks* (RNN). In the first case, the input sequence is multiplied by trainable weights to obtain intermediate hidden representations, i.e., layers, to reach the final output sequence. Hence, inputs and outputs of this network are mapped together through complex non-linear mathematical relations. On the other side, RNNs introduce a concept of *Memory*, i.e., the computations of the current hidden state exploit both the current inputs and the hidden state derived from the preceding layer. However, a major limitation of this architecture is embedded in the process of weights-updating through *backpropagation*. Indeed, this type of network suffers of *vanishing gradients* problems, which makes it extremely difficult to exploit this architecture with long input sequences. To address these limitations, *Long-Short Term Memory* (LSTM) and *Gated Recurrent Units* (GRU) have been introduced. The ability of exploiting previous hidden states justifies the application of RNNs to Natural Language Processing (NLP) tasks. Indeed, a sentence can be interpreted as a sequence of words where each word is influenced by the previous ones.

V. Sequence to Sequence Learning with Neural Networks

"Sequence to Sequence Learning with Neural Networks" [1], published in 2014, represents a major advancement in the usage of Neural Networks to solve Sequence to Sequence tasks. Simple RNN architectures can be used in Neural Machine Translation, but they achieve poor translations. Indeed, they address the task similarly to a word-to-word mapping, which generates several limitations when the lengths of the input and output sequences differ. To overcome these limitations, the paper introduces the *Encoder-Decoder* architecture.

i. Architecture

This architecture is composed of two LSTM networks, one representing the encoder and the other the decoder. The LSTM Encoder is used to map the input sequence to a vector of a fixed-dimensionality, and then the LSTM Decoder maps this vector to the target sequence. The intuition behind such approach is to be searched in the way humans translators address this task (we can notice that this brain-inspired approach is recurrent and will make its appearance also in the analysis of the other two papers). Indeed, a human-translator would firstly read the entire sentence and only secondly start to translate it. A similar approach is followed by this architecture, where firstly the encoder processes the entire sentence and then the decoder starts its decoding process. The flow of this architecture is represented in Figure 2.

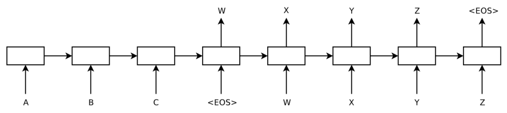


Figure 2: *[1] Model Representation*

ii. Implementation and Training

The model trained in the paper [1]:

- Encoder: 4 layers LSTM with 1,000 cells and 1,000 word-embeddings.
- Decoder: 4 layers LSTM with 1,000 cells and 1,000 word-embeddings.
- Final: Ensemble of 5 deep LSTMs architectures.
- Parameters: 384M parameters and 8,000 dimensional state each.
- WMT' 14 English to French, 12M sentences consisting of 348M French words and 304M English words.
- Input/Output Vocab.: 160,000/80,000.
- BLEU Score: 34.81.

iii. Replica Implementation

The implementation provided in the referenced Github Repository reduces the amount of computational power, making it suitable to be run on the GPU of portable computer. This implementation has been inspired by, and are worth of a mention, these other implementations on Github. [9] [10] [11] [12]
The architecture developed exploits:

- Encoder: 2 layers LSTM with 512 cells and 256 word-embeddings.
- Decoder: 4 layers LSTM with 512 cells and 256 word-embeddings.
- Parameters: approx. 14M.
- Loss: Cross Entropy.
- Optimizer: Adam.
- Multi30K English to German, approx. 31K sentences consisting of approx. 360k German words and 360k English words.
- Input/Output Vocab.: 8,000/6,000
- Epoch training time on NVIDIA Tesla K80: 30s x 15 Epochs.
- BLEU Score: approx. 14.5 depending on the random seed.

VI. Neural Machine Translation by Jointly Learning to Align and Translate

"Neural Machine Translation by Jointly Learning to Align and Translate" [2], published in 2016, elaborates over the *Encoder-Decoder* architecture proposed by the previous article. In particular, it addresses the potential issues related with the compression of the information in the fixed-length vector. Indeed, the previous architecture struggles to cope with long sentences, which is shown in the previous paper[1] by a rapid deterioration of the performance as the length of the input sentence increases. To overcome this issue, the paper [2] introduces the concept of *Attention*. Again, the intuition behind this is inspired to the human brain. In the process of translating a sentence, the human brain poses attention to specific parts of the sentence sequentially (e.g., understanding the gender of noun may require to focus on the article used, instead of the noun itself).

i. Architecture

This concept of *Attention* is implemented through an extension to the encoder-decoder model which learns to align and translate jointly. In particular, the core concept introduced relies on *intermediate context vectors* and an *"alignment score"*. It consists of a weighted sum of the intermediate context vectors of all the time-steps, where the weights are represented by the *"alignment scores"*. This concept is visualized in Figure 3 and it is mathematically described as follows [2]:

$$s_t = f(s_{t-1}, y_{t-1}, c_t) \tag{1}$$

$$h_i = [h_i^{\rightarrow\ T}; h_i^{\leftarrow\ T}]^T \tag{2}$$

$$c_t = \sum_{i=1}^{n} \alpha_{t,i} h_i \tag{3}$$

$$\alpha_{t,i} = softmax(a(s_{t-1}, h_i)) \tag{4}$$

$$a(s_{t-1}, h_i) = v_a^T tanh(w_a[s_{t-1}; h_i]) \tag{5}$$

where s_t represents the *decoder state*, h_i the *hidden state*, c_t the context vector and $\alpha_{t,i}$ the probability that the target word y_t is aligned to a source word x_i.

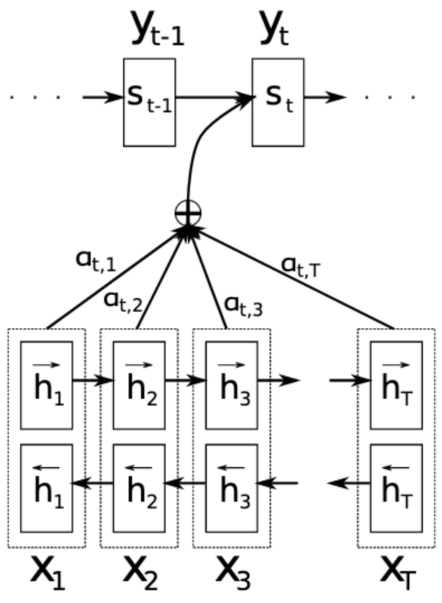


Figure 3: *[2] Model Representation*

The Encoder architecture implemented in the paper exploits a *Bidirectional Recurrent Neural Network* (BiRNN). As described in the paper [2]:

> A BiRNN consists of forward and backward RNN's. The forward RNN $f^{\rightarrow}$ reads the input sequence as it is ordered (from x_1 to x_{T_x}) and calculates a sequence of *forward hidden states* $(h_1^{\rightarrow}, ..., h_{T_x}^{\rightarrow})$. The backward RNN $f^{\leftarrow}$ reads the sequnce in the reverse order (from x_{T_x} to x_1), resulting in a sequence of *backward hidden states* $(h_1^{\leftarrow}, \ldots, h_{T_x}^{\leftarrow})$. We obtain an annotation for each word x_j by concatenating the forward hidden state $h_j^{\rightarrow}$ and the backward one $h_j^{\leftarrow}$, i.e., $h_j = [h_j^{\rightarrow T}; h_j^{\leftarrow T}]^T$

The intuition behind the implementation of a BiRNN is that a word might be influenced by the following words, hence by introducing the forward and backward hidden states, each context vector contains information regarding the entire sentence sequence. This makes it possible to exploit alignment and scoring to attention the words in the sentence which are more relevant in the translation of each word.

ii. Implementation and Training

The model trained in the paper [2]:

- Encoder: BiDirectional GRU with 1,000 units.
- Attention: Weighted sum of all the intermediate context vectors through tanh activation.
- Decoder: GRU with a 1,000 hidden units.
- WMT' 14 English to French, 12M sentences consisting of 348M French words and 304M English words.
- Input/Output Vocab.: 160,000/80,000.
- BLEU Score: 36.15.

iii. Replica Implementation

The implementation provided in the referenced Github Repository reduces the amount of computational power, making it suitable to be run on the GPU of portable computer. This implementation has been inspired by, and are worth of a mention, these other implementations on Github. [13] [14] [15] [16]
The architecture developed exploits:

- Encoder: BiDirectional GRU with 512 units.
- Attention: Weighted sum of all the intermediate context vectors through tanh activation.
- Decoder: GRU with 512 hidden units.
- Parameters: approx. 20.5M.
- Loss: Cross Entropy.
- Optimizer: Adam.
- Multi30K English to German, approx. 31K sentences consisting of approx. 360k German words and 360k English words.
- Input/Output Vocab.: 8,000/6,000
- Epoch training time on NVIDIA Tesla K80: 57s x 10 Epochs.
- BLEU Score: approx. 31.5 depending on the random seed.

VII. Attention Is All You Need

"Attention Is All You Need"[3], published in 2017, addresses the limitations of [2] in terms of attention computational complexity. Indeed, the task of evaluating different context vector at every state of the decoder is sequential, i.e., due to RNN's nature, computing h_2 requires having already computed h_1. This implies that computational parallelism cannot be exploited to reduce the training time. This paper [3] introduces the *Transformer* architecture which exploits the concept of *Self-Attention* [17] as a solution to the issue.

i. Architecture

The *Transformer* architecture is among the most successful implementations of *Self-Attention*. The core concepts of the transformer revolve around the mechanisms introduced by the paper [3]: *Scaled-Dot Product Attention* and *Multi-Head Attention*. These mechanisms are mathematically described as follows [3]:

$$Att(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \qquad (6)$$

$$MultiHead(Q,K,V) = Concat(head_1, \dots, head_h)W^O \qquad (7)$$

with

$$head_i = Att(QW_i^Q, KW_i^K, VW_i^V) \qquad (8)$$

where the projections are parameter matrices W_i^Q and $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$.

A visual representation of these two mechanisms is provided in Figure 4.

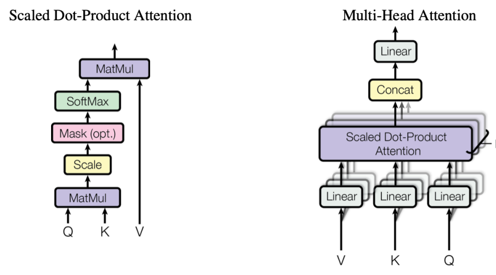


Figure 4: *[3] Scaled-Dot Product Attention and Multi-Head Attention*

As explained in the paper [3] the transformer uses Multi-Head attention in three different ways:

(1) In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sentence [...]
(2) The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
(3) Similarly, self-attention layers in the decoder allow each position

in the decoder to attend to all positions in the decoder up to and including that position. [...]

The architecture of the transformer model follows the Encoder-Decoder fashion, where each layer in the Encoder and Decoder presents a Multi-Head Attention and a fully-connected Feed-Forward network. Where the Position-wise Feed-Forward Network consists of two linear transformation with a ReLU activation in between, mathematically represented as [3]:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \quad (9)$$

To give an intuition of the relation between this architecture and the one discussed in the previous article[2], we can think of equation (6) as the calculation of vector α_i from equation (4) in the self-attention fashion. However, in the transformer architecture, multiple α_i are computed, weighted, concatenated and passed through the feed-forward NN layer. On the other side, the decoder receives the expected output. The "Multi-Head Encoder-Decoder Attention" takes as input the encoder outputs and the two are combined by the decoder.

A visual representation of the transformer architecture is provided in Figure 5.

ii. Implementation and Training

The model trained in the paper [3]:

- Encoder:
  - Multi-Head Attention: Splitted parallel computation of the Scaled-Dot Product Attention.
  - Position-wise Feed-Forward: Fully-connected feed-forward network with ReLU activation.
- Decoder:
  - Multi-Head Attention: Splitted parallel computation of the Scaled-Dot Product Attention.

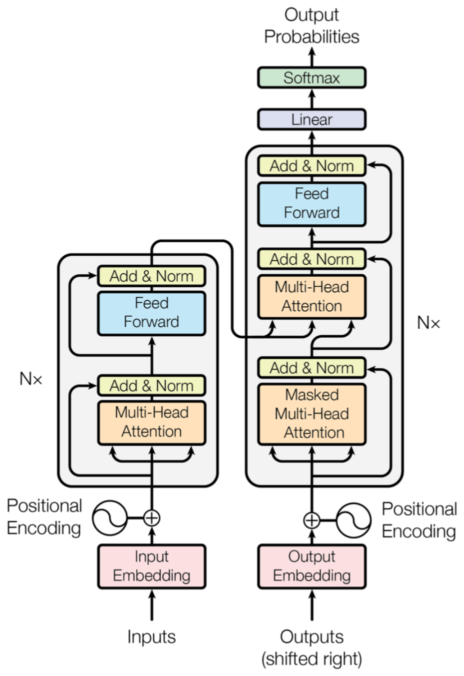


Figure 5: *[3] Transformer Architecture*

  - Position-wise Feed-Forward: Fully-connected feed-forward network with ReLU activation.
- WMT' 14 English to French, 36M sentences.
- BLEU Score: 41.8.

iii. Replica Implementation

The implementation provided in the referenced Github Repository reduces the amount of computational power, making it suitable to be run on the GPU of portable computer. In addition, some slight variations (documented in the notebook) have been performed to obtain This implementation has been inspired by, and are worth of a mention, these other implementations on Github. [18] [19] [20] [21] [22]
The architecture developed exploits:

- Encoder:
  - Multi-Head Attention: Splitted parallel computation of the

Scaled-Dot Product Attention.
  - Position-wise Feed-Forward: Fully-connected feed-forward network with ReLU activation.
- Decoder:
  - Multi-Head Attention: Splitted parallel computation of the Scaled-Dot Product Attention.
  - Position-wise Feed-Forward: Fully-connected feed-forward network with ReLU activation.
- Parameters: approx. 9M.
- Loss: Cross Entropy.
- Optimizer: Adam.
- Multi30K English to German, approx. 31K sentences consisting of approx. 360k German words and 360k English words.
- Input/Output Vocab.: 8,000/6,000
- Epoch training time on NVIDIA Tesla K80: 17s x 10 Epochs.
- BLEU Score: approx. 35.7 depending on the random seed.

VIII. Evaluation

i. BLEU Score

The metric used to assess the performance of these implementations is BLEU Score. This represents the most widely adopted metric to assess Machine Translations within the literature. The metric has been introduced by *"BLEU: a Method for Automatic Evaluation of Machine Translation"* [23] with the aim of introducing a measure that was exploitable for quick or frequent evaluations of translations. It compares a reference sequence n-gram with the candidate sequence n-gram to return a score (0 - 100) of closeness to the reference translation.

ii. Results

In the following table, the BLEU scores of the different implementations discussed are reported:

	FR - EN	DE - EN
Seq2Seq	34.81	-
Seq2Seq Rep	-	14.5
Attention	36.15	-
Attention Rep	-	31.5
Transformer	41.8	-
Transformer Rep	-	35.7

where *Seq2Seq* refers to [1] architecture, *Attention* to [2] architecture and *Transformer* to [3] architecture. Whereas the *Rep* indicates the *Replica*, i.e., the "computationally-efficient" implementation presented in the notebooks associated to this paper.

Given the nature of the BLEU metric and the diversity of the training datasets, it is difficult to provide a comparison of the performances of the replicated architectures to the ones of the original implementations.

However, we can perform an analysis among the replicated architectures. Indeed, we can notice that the advancements proposed by the papers are reflected in an evolution of the performances in the replicated architectures. In particular, the increase in performance from the introduction of the "Attention" mechanism is the most significant as it leads to an increase of approximately 17 BLEU score points.

IX. Literature

The purpose of this analysis was that of exploring three milestone articles in the context of Neural Machine Translation. However, new advancements in the field have already taken place. Although not disruptive architecture changes occurred, Bidirectional Encoder Representations from Transformers (BERT) has been developed. It represents Google's state-of-the-art language modelling architecture based on the transformers model [7]. This model allows fine-tuning of the pre-trained language model according to obtain state-of-the-art performances on specific NLP tasks.

References

[1] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[2] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[4] D. Elliott, S. Frank, K. Sima'an, and L. Specia, "Multi30k: Multilingual english-german image descriptions," *arXiv preprint arXiv:1605.00459*, 2016.

[5] H. Somers, "Machine translation," *The Oxford handbook of translation studies*, 1996.

[6] F. Stahlberg, "Neural machine translation: A review," *Journal of Artificial Intelligence Research*, vol. 69, pp. 343–418, 2020.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[8] O. Bojar, C. Buck, C. Federmann, B. Haddow, P. Koehn, J. Leveling, C. Monz, P. Pecina, M. Post, H. Saint-Amand, *et al.*, "Findings of the 2014 workshop on statistical machine translation," in *Proceedings of the ninth workshop on statistical machine translation*, pp. 12–58, 2014.

[9] farizrahman4u, "Seq2seq." `https://github.com/farizrahman4u/seq2seq`, 2017.

[10] astorfi, "sequence-to-sequence-from-scratch." `https://github.com/astorfi/sequence-to-sequence-from-scratch`, 2019.

[11] bentrevett, "pytorch-seq2seq." `https://github.com/bentrevett/pytorch-seq2seq`, 2021.

[12] macournoyer, "neuralconvo." `https://github.com/macournoyer/neuralconvo`, 2017.

[13] thomlake, "pytorch-attention." `https://github.com/thomlake/pytorch-attention`, 2019.

[14] bentrevett, "pytorch-seq2seq." `https://github.com/bentrevett/pytorch-seq2seq`, 2021.

[15] graykode, "nlp-tutorial." `https://github.com/graykode/nlp-tutorialh`, 2019.

[16] Nick-Zhao-Engr, "Machine-translation." `https://github.com/Nick-Zhao-Engr/Machine-Translation`, 2021.

[17] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," *arXiv preprint arXiv:1601.06733*, 2016.

[18] jadore801120, "attention-is-all-you-need-pytorch." `https://github.com/jadore801120/attention-is-all-you-need-pytorch`, 2021.

[19] graykode, "nlp-tutorial." `https://github.com/graykode/nlp-tutorialh`, 2021.

[20] bentrevett, "pytorch-seq2seq." `https://github.com/bentrevett/pytorch-seq2seq`, 2021.

[21] soofware, "attentions." `https://github.com/sooftware/attentions`, 2022.

[22] astorfi, "sequence-to-sequence-from-scratch." `https://github.com/astorfi/sequence-to-sequence-from-scratch`, 2019.

[23] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318, 2002.