Build your own Transformer - Term Paper

Rricha Jalota
Universität des Saarlandes
Matriculation Nr. 7010592
rrja00001@stud.uni-saarland.de

Abstract

The aim of this work is to gain experience and familiarity with different variants of sequence to sequence models for neural machine translation. In particular, the focus is on the implementation of the transformer architecture from scratch using PyTorch. The models are evaluated using different automated metrics like BLEU and METEOR. In this work, the effect of hyperparameter tuning has also been investigated.

1 Introduction

Machine Translation is a prominent research area in NLP, in which a sequence of input given in one language is converted into an output sequence in another language. Historically, statistical models based on words or phrases were used to carry out translation from one language to another. In a word-based translation system, first, word alignments from source to target language are learned and then a language model is used to predict words in target language given the context words in source language. In contrast, phrase-based translation systems are able to handle more complex syntactic dependencies than word-based systems. However, both the systems are not able to capture long-term dependencies.

Sequence to sequence or "Seq2Seq" paradigm introduced by Sutskever et al. (Sutskever et al., 2014) aimed to tackle this drawback of statistical translation systems. This end-to-end learning system consists of two neural networks, dubbed *Encoder* and *Decoder*. An Encoder takes an input sequence in source language and converts it into a single, fixed-length context vector. The decoder then employs this context vector to generate an output sequence in the target language.

In this work, three different sequence to sequence architectures have been studied and implemented in PyTorch. The goal is to compare the performance of the state-of-the-art Transformer architecture (Vaswani et al., 2017) with other recurrent and non-recurrent architectures and investigate which architecture works the best with limited data and training.

2 Related Work and Background

2.1 Sequence to Sequence learning using Recurrent Neural Networks

This variant of Seq2Seq paradigm was first introduced by (Sutskever et al., 2014) and is the simplest of all. Both the encoder and decoder are made up of recurrent neural networks (LSTM (Hochreiter and Schmidhuber, 1997) or GRU (Cho et al., 2014)). A recurrent neural network processes the input sequence at each time step, thereby allowing variable length input. The hidden state of the network acts as memory that stores relevant information about the sequence until that time step. The hidden state of the network at the last time step thus contains summarized/relevant information about the entire input sequence and is also referred as *context vector*. This context vector is then decoded by another RNN (decoder) word by word. To start the decoding process, a special token like <sos> is given to the decoder. At the time of training, except for the first time step (where the special token is the input), at all other time steps, the ground truth $y(t)$ is given as input at time-step, $t+1$. This is called *teacher forcing*. At the time of evaluation or inference, instead of the ground truth, the predicted word at time-step t, $\hat{y}(t)$ is given as input at $t+1$. In this study, a simple LSTM-based encoder-decoder Seq2Seq model has been trained. Details of the experimental setup can be found in Section 5.

2.2 Sequence to Sequence learning using Convolutional Neural Networks

This non-recurrent architecture was proposed by (Gehring et al., 2017) and has been proven to work better than recurrent NMT models. Each convolutional layer consists of a number of filters (or kernels) with a specified width, w, which allows it to see w consecutive tokens in a text. Each filter slides across the input sequence from the first token to the last token and extracts a different feature from the text. The resultant set of features are then used as input to another convolutional layer. Finally, a feature representation of the source text is used to generate text in target language.

The encoder of a convolutional seq2seq model produces two context vectors, namely conved vector and combined vector, for each token in the input sequence. These two context vectors are then used by the decoder for translation. The conved vector is generated after each token is passed through convolutional layers. The combined vector is the elementwise sum of the conved vector and the embedding of an input token. Figure 1 depicts the architecture.

Due to no time-step information as in recurrent models, for non-recurrent architectures, we somehow need to provide the word-order information. To this end, positional embeddings are used and an elementwise sum of token and position embeddings forms an embedding vector. This embedding vector is then passed through a linear layer which transforms the shape of this vector to that of a specified hidden dimension. This hidden vector is then passed to a convolutional layer, consisting of N convolutional blocks. The output from convolutional layer is then passed to the linear layer, which gives us the conved vector and thereafter, the combined vector for each input token.

In the decoder, there is no sequential processing. Rather, all target tokens are generated in parallel. The actual target sequence is given as input to the decoder and using padding (mask), only the left context is made available to the convolutional blocks. Thereafter, using attention mechanism and conved and combined vectors from the encoder , target tokens are predicted. For more details, please refer the original paper (Gehring et al., 2017). In this work, a convolutional seq2seq architecture has been trained and evaluated against other implemented NMT architectures. The experimental setup can be found in Section 5.

2.3 Current state of the art

The Transformer architecture proposed by (Vaswani et al., 2017) is also a non-recurrent architecture that aims to tackle long-range dependencies and reduce the runtime of sequential computation (similar to convolutional-based seq2seq models). Several works have been built on top of it to extend its capabilities, namely Transformer-XL (Dai et al., 2019), BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019), etc. In this work, however, the original/vanilla Transformer architecture has been studied and compared against its predecessors. In section 6, the differences have been elucidated.

3 Transformer Architecture

In ConvS2S models, for handling long-range dependencies, the number of operations required to relate information from two arbitrary input or output positions grows linearly w.r.t. the distance between positions. This makes it inconvenient to learn long-range dependencies. The Transformer architecture (Vaswani et al., 2017) aims to handle this issue and requires only a constant number of operations to learn the dependencies between two distant positions.

The transformer architecture, as shown in Figure 2, consists of N encoder and decoder layers. Similar to ConvS2S, the Transformer's encoder does not compress the source sequence into a single context vector. Rather, it generates a series of context vectors. A context vector is different from a hidden vector in the sense that, a context vector has already seen tokens at all positions of an input sequence. In contrast, a hidden vector in an RNN can only see tokens that come before it.

The (multi-head) attention mechanism is the most crucial component of the transformer models, which is present in both the encoder and decoder, as a sub-layer. To compute attention, three vectors named as query, key and value are used. The dot product of query and key vectors is used to get an attention vector, which is passed via softmax function and then multiplied by the value vector. The dot product of query and key is scaled by the square-root of their dimension d_k, and hence this operation is termed as scaled dot-product attention. Figure 3 shows pictorial representation of scaled dot-product attention.

Instead of applying the attention function on key, value and query vectors only once, the authors found it beneficial to linearly project queries, keys

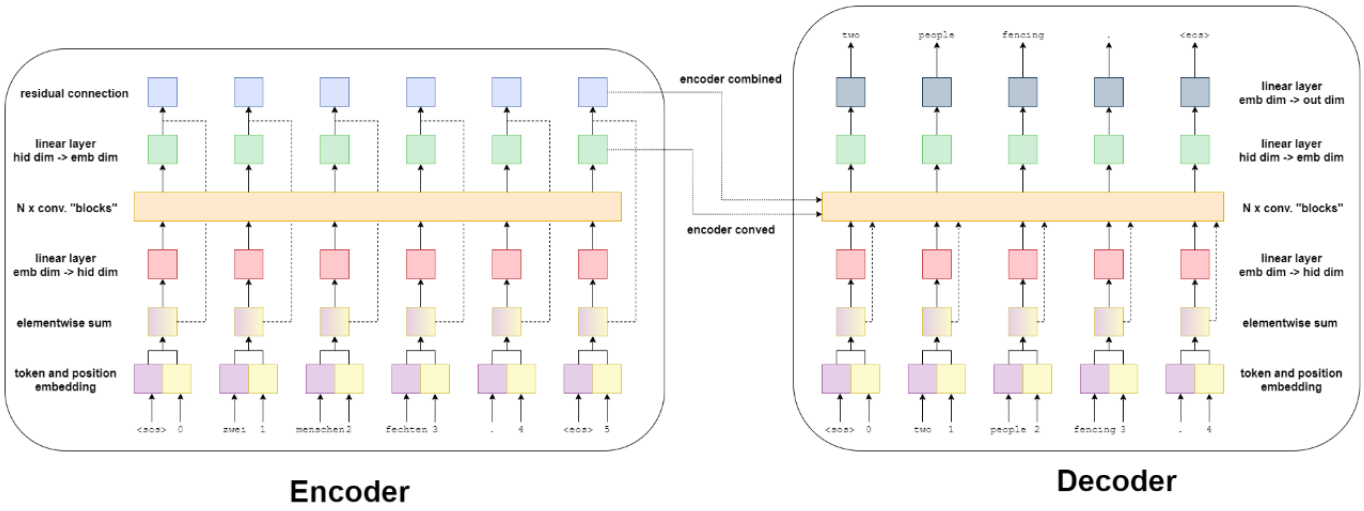

Figure 1: Convolutional Seq2Seq Architecture. Source: https://tinyurl.com/yedkak4t

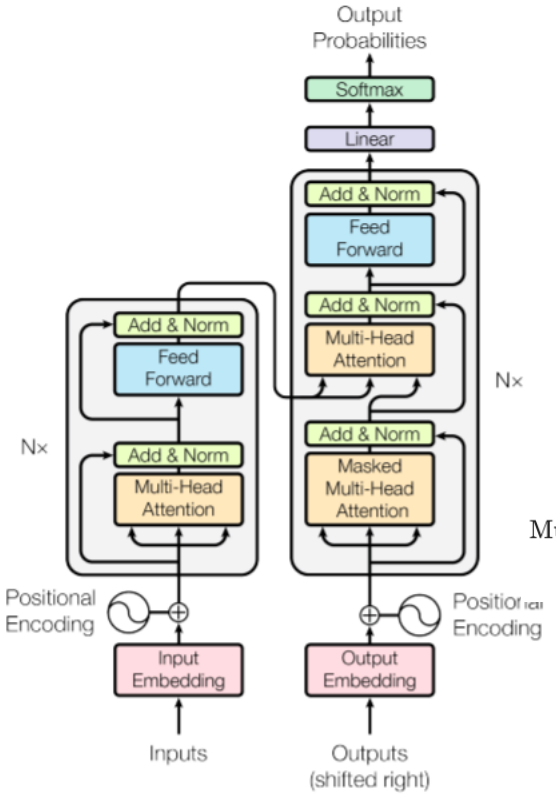

Figure 2: Transformer Architecture. Source (Vaswani et al., 2017)

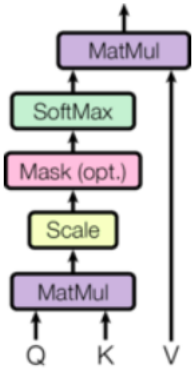

Figure 3: Scaled-dot product attention. Source (Vaswani et al., 2017)

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$
$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Figure 4: Multi-head attention formula. Source (Vaswani et al., 2017)

and values h times in order "to jointly attend to information from different representation subspaces at different positions" (Vaswani et al., 2017). Figure 4 shows the formula of multi-head attention and Figure 5 displays the corresponding architecture.

Multi-head attention mechanism is used in three different ways:

1. In encoder for self-attention. The input to encoder is divided into three identical vectors (or sub-vectors) to get key, value and query vectors. It is called self-attention because each position can attend to all other positions of the previous layer of encoder or embedding layer.

2. In decoder to attend to all positions up to and including that position (i.e. to attend to previously generated outputs).

3. In encoder-decoder attention layers, where the query vector comes from the previous decoder layer, and the key and value vectors are drawn

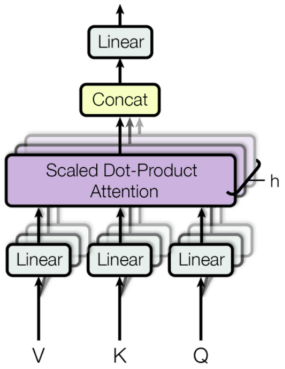

Figure 5: Multi-head attention. Source (Vaswani et al., 2017)

from the output of the encoder. Figure 2 illustrates this.

Position-wise Feedforward Network forms another major sub-component of Transformer's encoder and decoder. It consists of 2 linear layers with a ReLU activation in between.

As can be seen from the Transformer architecture in Figure 2, layer normalization is done after each sub-layer. Similar to ConvS2S, to account for the word-order information, an additional **positional encoding** is added to the input embeddings. To allow the model to extrapolate to any arbitrary sequence length, sine and cosine functions of different frequencies (same as the one mentioned in the original paper) are used to encode positional information of the input tokens.

Masking also plays an important role during both the encoding and decoding phases. Source mask allows to only pay attention to sentence tokens and not to pad tokens while target mask aims to prevent the decoder from "looking ahead" in the target sequence. Further details about the architecture can be found in the original paper (Vaswani et al., 2017).

4 Evaluation Metrics

Two most commonly used automated MT evaluation metrics - BLEU (Papineni et al., 2002) and METEOR (Banerjee and Lavie, 2005) have been used to qualitatively evaluate the translations produced by the studied model architectures.

1. **BLEU** To calculate the BLEU score, modified n-gram precision is multiplied with the best match length score a.k.a brevity penalty (recall).

   - Modified N-gram Precision - N-gram precision refers to the fraction of n-grams in the generated text that are present in the reference text. Using modified n-gram precision, we match the generated n-grams only as many times as they occur in the reference text. Then, a geometric mean of all the modified n-gram precision scores is taken to calculate the final precision for the text sequence. The formula for precision is given in Figure 6:

$$Precision = \exp(\sum_{n=1}^{N} w_n \log p_n), \quad \text{where} \quad w_n = 1/n$$

Figure 6: Precision for BLEU. Here, p_n is the modified n-gram precision.

   - Best Match Length - For a predicted translation text, there can be more than one reference texts. This presents a challenge to calculate recall. However, intuitively, if the generated text is longer, then there is a greater possibility of it having some match with the reference text(s). Therefore, to compute recall, brevity in generated texts is penalized. The formula is shown in Figure 7.

$$BP = \begin{cases} 1, & \text{if } c > r \\ \exp(1 - \frac{r}{c}), & \text{otherwise} \end{cases}$$

Figure 7: Recall for BLEU. Here, c is the length of the translation text. r is the average length of all the references.

2. **METEOR** is considered to have a better correlation with human evaluation as it modifies the precision and recall scores that are computed in BLEU. In BLEU, since the brevity penalty uses average length of the reference corpus, the scores of individual sentences are compromised. METEOR addresses this problem by using a weighted F-score instead. To compute the **weighted F-score**, first, the largest subset of unigram mappings that form an alignment between the generated and reference translation are found. To find unigram mappings, exact matches, Porter Stemming and WordNet synonymy are used. If m

is the number of mapped unigrams between two texts, then precision is m/c and recall is m/r, where c and r are candidate (or generated text) and reference lengths, respectively. The formula of weighted F-measure is given in Figure 8. In METEOR, incorrect word

$$F = \frac{PR}{\alpha P + (1-\alpha)R)}$$

Figure 8: Weighted F-score for METEOR. Alpha parameter controls relative weights of precision and recall.

order in generated text is also penalized using the penalty function, shown in Figure 9. The METEOR score is finally computed as $(1 - Penalty)F$.

$$Penalty = \gamma(\frac{c}{m})^{\beta}, \quad \text{where} \quad 0 \leq \gamma \leq 1$$

Figure 9: Penalty for METEOR. Beta parameter controls shape of penalty function.

5 Experiments

5.1 Dataset Used

All experiments are carried out against a subset of Multi30k dataset (Elliott et al., 2016). This dataset contains around 30,000 parallel English-German sentences, each with around 12 words per sentence. This dataset is available within the PyTorch framework. For training, due to limited computational resources, only 5120 out of 29000 sentences were used. For validation, 1014 sentences and for test, 1000 sentences were used.

5.2 Experimental Setup

All the models were trained under the following configurations:

- Training epochs: 50 with early stopping. Early stopping is a regularization method which is used to avoid overfitting. Early stopping with a patience value of 5 is used in the experiments. This means, a maximum of 5 iterations are allowed to pass to improve model's fit on training data even though the generalization/validation error starts increasing.
- Gradient clipping: 1
- Learning rate: 0.005 for LSTM-based seq2seq; 0.0005 for all others
- Encoder-decoder dropout: 0.2
- Adam optimizer with Cross Entropy loss
- All words occurring less than 3 times in the corpus were discarded (i.e. were not added to the source/target vocabularies).

Model parameters specific to each architecture have been shown in tables 1, 2 and 3. All experiments were performed on Google Colab.

Table 1: Parameters used for training LSTM-based Seq2Seq Model

Parameter	**LSTMS2S**
# ENC-DEC layers	2
ENC-DEC embedding dim.	256
Hidden dim.	512
Teacher-forcing ratio (during training)	0.8
Total Training parameters	12,236,236

Table 2: Parameters used for training Convolutional Seq2Seq Architecture

Parameter	**ConvS2S**
ENC-DEC kernel-size	3
# conv. blocks in ENC-DEC	6
Hidden dim.	512
Embedding dim.	256
Total Training parameters	23,440,076

Table 3: Parameters used for training Transformer-based Seq2Seq Architecture

Parameter	**Transformer1**	**Transformer2**
# ENC-DEC heads	8	8
# ENC-DEC layers	3	4
Hidden dim.	256	256
Position-wise feedforward dim.	512	512
Total Training parameters	6,878,412	7,933,388

For brevity, training-validation epoch-loss plots for only Transformer models is shown in Figures 10 and 11. Epoch-Loss plots for other models can be found in the code repo attached.

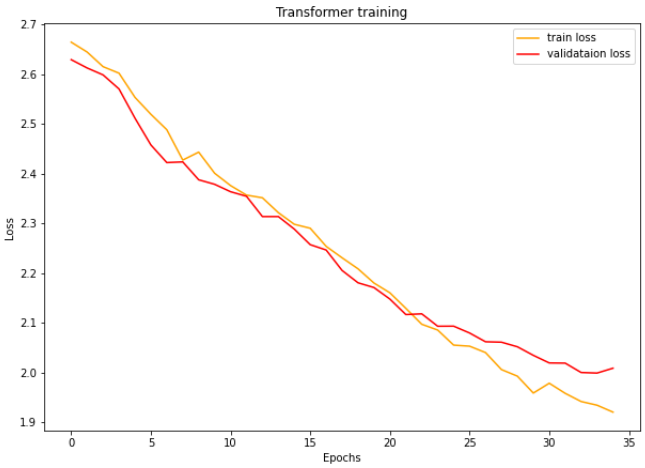

Figure 10: Epoch-loss plot for Transformer1 model.

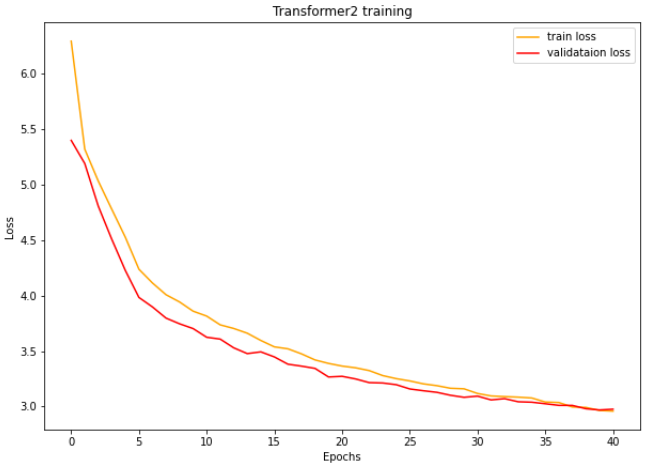

Figure 11: Epoch-loss plot for Transformer2 model.

6 Results and Discussion

Before looking at the results, from the tables 1, 2 and 3, we can already hypothesize that due to reduced number of training parameters, transformer models must perform better and faster than LSTMS2S and ConvS2S. Results of the experiments are presented in Table 4 and Table 5 shows predictions from the trained models for a few example sentences.

With only (approx.) 5000 training examples and 35 training epochs, Transformer1 model performs impressively and gives the highest BLEU and METEOR score (and lowest test loss). In the example sentences, the candidate text generated by transformer-1 model almost matches the reference text and shows only minimal hallucination. However, when the number of encoder-decoder layers is increased to 4 while keeping all other parameters constant, as per BLEU and METEOR scores, the Transformer2 architecture performs worse than ConvS2S model even though it has fewer training parameters. Looking at the translations of the two models in Table 5, one can argue that both the models depict almost the same degree of hallucination. So the BLEU and METEOR scores might not give a fair comparison of the two models. For LSTMS2S model, the BLEU and METEOR scores are rather low and in fact, the same translation is being produced by the model for all the input sentences. This means that the model was not able to learn, with the set hyper-parameters and given number of training examples. For LSTMS2S, I experimented with different learning rates ranging from 1e-1 to 1e-5, different number of Encoder-Decoder layers: 2,3,4; higher dropout values (0.5), lower hidden (256) and embedding dimensions (128), but still the model showed a similar performance. This means, to be able to learn decently well, an LSTMS2S would require more number of training examples. This also makes sense because due to their sequential nature, RNN architectures are not able to hold long-term dependency information and therefore, to be able to generate sensible text, they would first need to go through a rigorous training.

Apart from the given configuration of the Transformer2 model, I experimented with various other parameters for the transformer architecture. When the number of Encoder-decoder layers were increased to 5 or 6, the model showed terrible performance similar to that of the RNN model. Changing the learning rate and gradient clipping factor to account for the increased number of trainable parameters also did not help. So, the configuration of Transformer-1 model can be considered optimal for the given number of limited training examples.

Overall, we can see how with only minimal training examples and carefully chosen hyper-parameters, transformers can perform better than all the other architectures. The benefit of using multi-head mechanism and positional encoding can be seen because by processing entire sentences at once, relationships between words can be learned better, while keeping the trainable parameters considerably lower than the other two Seq2Seq archi-

tectures.

7 Conclusion

The goal of this work was to gain experience with the implementation and working of different neural machine translation architectures, and understand how they differ from each other. In this paper, three different seq2seq architectures were studied - recurrent, convolutional and transformer-based. All the three models were trained on a limited set of training examples with similar configurations and it was seen that the transformer-based seq2seq model outperformed other two models. The effect of using different hyper-parameters was also investigated. It would have been interesting to see the results of beam-search decoding from the transformer model but due to limited time and resources, it still remains a future work. It would also be interesting to see how these models perform on the task of transcoding i.e. converting one programming language into another, similar to the works of (Roziere et al., 2020).

References

Satanjeev Banerjee and Alon Lavie. 2005. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Desmond Elliott, Stella Frank, Khalil Sima'an, and Lucia Specia. 2016. Multi30k: Multilingual english-german image descriptions. In *Proceedings of the 5th Workshop on Vision and Language*, pages 70–74. Association for Computational Linguistics.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, pages 1243–1252. PMLR.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Baptiste Roziere, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised translation of programming languages. *Advances in Neural Information Processing Systems*, 33.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Table 4: Neural Machine Translation results of classifiers trained on 5120 sentences of the Multi30k dataset. Higher the BLEU and METEOR score, the better the model is.

Model	**# Epochs**	**Test Loss**	**BLEU**	**METEOR**
LSTMS2S	7	5.210	2.69	24.51
ConvS2S	18	2.839	16.54	42.47
Transformer1	35	1.993	26.95	59.84
Transformer2	41	2.972	8.23	30.96

Table 5: Qualitative comparison of the translations from the trained models.

Model	**Text**
Source Text	eine glückliche frau bereitet in einem coffee-shop eine erfrischung zu .
Reference Text	a happy woman is preparing a refreshment at a coffee shop .
LSTMS2S	a man in a blue shirt and a . ⟨eos⟩
ConvS2S	a blond woman in a pink shirt is being in a ⟨unk⟩ . ⟨eos⟩
Transformer1	a happy woman prepares a baton in a gymnasium . ⟨eos⟩
Transformer2	a woman is taking a picture of a picture of a ⟨unk⟩ . ⟨eos⟩
Source Text	eine frau spielt ein lied auf ihrer geige .
Reference Text	a female playing a song on her violin .
LSTMS2S	a man in a blue shirt and a . ⟨eos⟩
ConvS2S	a woman playing a guitar on a sidewalk . ⟨eos⟩
Transformer1	a woman plays a song on the violin . ⟨eos⟩
Transformer2	a woman is playing a guitar in the microphone . ⟨eos⟩
Source Text	ein kleines kind fotografiert mit einer kamera auf einem stativ .
Reference Text	a small child taking a picture with a camera on a tripod .
LSTMS2S	a man in a blue shirt and a . ⟨eos⟩
ConvS2S	a little child wearing a blue shirt is sitting on a small bench . ⟨eos⟩
Transformer1	a young child taking pictures with a camera on a tripod . ⟨eos⟩
Transformer2	a young boy is standing in front of a tree looking at the camera . ⟨eos⟩