# ConvSeq2Seq in JoeyNMT

**Jakob Forstmann**
Heidelberg University
Proseminar Introduction to Neuronal Networks and Sequence to Sequence Learning
forstmann@cl.uni-heidelberg.de

## 1 Introduction

This paper describes the implementation of Convolutional Sequence to Sequence Learning in JoeyNMT(Kreutzer et al., 2019) following the orignal implementation [1]. As we are aiming for a simpler version compared to original implementation we left out some optimizations described in the section Implementation.However we followed the same preprocessing as described in the next section as closely as possible.

## 2 Preprocessing

In order to pre-process the data we simply adopted the existing shell script to our needs. It includes two steps,tokenization and cleaning and applying Byte Pair Encoding (BPE) [2]. First of all, we normalized punctuation, removed non printable characters and tokenized the dataset using scripts from the moses library. Furthermore we cleaned the test data by removing unnecessary HTML tokens from the test split. Considering that the paper(Gehring et al., 2017) evaluates on the task of neuronal machine translation the out of vocabulary problem could arise. Following the implementation of the paper we applied BPE(Sennrich Rico,Haddow Barry and Birch,Alexandra, 2016) to alleviate this problem. The algorithm can be break down into the two steps, learning the representations and applying them. For the first step, a symbol vocabulary is built using the characters of the tokens from step one and a special symbol indicating the end of word. Then the most frequent pair of token from our vocabulary are merged replacing the two tokens from that pair. This process is repeated until a chosen amount of merge operations are performed whereat each step the performed merge operation is stored. In our case we used 4000 merge operations to learn the BPE representations. Following the second step the words from the corpus are first split into characters in the same way as during learning and then use the stored operations to segment the words into sub words. As an example directly taken from the paper consider the out of vocabulary german compound word *Gesundheitsforschungsinstitute*. In order to translate the word is split into five sub words which are then translated one by another. As a consequence using BPE we can tackle the open vocabulary problem with a fixed size vocabulary.

## 3 Model Architecture

In a broad view the model follows the common encoder-decoder framework, however instead of using Transformers it uses CNNs for the encoder and the decoder. Both of them embeds the source respective the target sentence and concatenate the embedding with positional encoding of size $f$. In contrast to the encoder the input for the decoder is right shifted such that the decoder can not see future words. Furthermore the k-1 element elements on the left and the right side of the decoders input are padded to hide future positions. For example for the first element of the right shifted sentence the padded decoders input would be (<pad>)*k-1<start of sequence> effectively hiding the k-1 after the <start of sequence> symbol. After the convolution was performed the k-1 element are discarded from the output as their prediction is useless and the size of the decoders output should mirror the size of the encoders output. On the other hand the encoder uses zero padding to match the sequence length of the input sequence.

Then again the encoder as well as the decoder consists of a chosen amount of layers which in turn contains a 1D convolution operating on a one sequence as opposed to operating on for example images with a 2d convolution. In both parts each convolution applies a kernel $W^{2dxkd}$ producing

[1] available at https://github.com/facebookresearch/fairseq

[2] As in the original paper we used the implementation from https://github.com/rsennrich/subword-nmt

vectors of size $d$, where k denotes the kernel width e.g. the size of the sliding window and d is the embedding dimension. As a consequence prior to applying the kernel the embedded input is projected to the convolution size d . Lastly the Gated Linear Unit (GLU) transforms the input of size 2d back to the embedding dimension d by splitting the input in half, pushing the second half through a sigmoid function and adding the results point wise with the first half. Moreover, residual connections around each layer are applied.
To conclude the model architecture we will look at the introduced attention mechanism which the authors called multi-step attention. The attention score is calculated between every decoder state and the last encoder state. In Addition the target embedding are added to the current decoder state prior to computing the attention score. Likewise the source embedding is added to last encoder state,but is instead used to compute the final context vector. Eventually the context vector is computed as the weighted sum between the normalized attention score using the softmax function and the concatenated last encoder output as described earlier. This calculation is iteratively done until the last decoder layer is reached as the context vector of the previous layer is added to the input of the next decoder layer. Then, in order to predict the next word the top feature map of the last decoder state is transformed using a linear layer and pushed through a softmax function.

## 4 Implementation

Overall the implemented version closely follows the description and the original implementation including the performed initialization of the different layers. However while we did scale the different layers according to the paper we left out scaling the gradients of the encoder output by the number of attention layers since the code was hard to understand. In Addition instead of implementing a CNN layer nearly from scratch with a custom class we used the the conv1d class from pytorch directly . The downside is, that in their implementation they make use of the general purpose matrix product which they claim in the docstring of the module is faster for smaller kernels. The most notably difference is that the original implementation uses incremental inference whereas we use beam search provided by JoeyNMT.

## References

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122.

Julia Kreutzer, Joost Bastings, and Stefan Riezler. 2019. Joey NMT: A minimalist NMT toolkit for novices. *To Appear in EMNLP-IJCNLP 2019: System Demonstrations*.

Sennrich Rico,Haddow Barry and Birch,Alexandra. 2016. Neural Machine Translation Systems for WM. In *In Proceedings of WMT*, Edingburgh.