# Sequence-to-sequence Architecture Using BERT

**Claudio Scheer** and **José Fernando Possebon**
Pontifical Catholic University of Rio Grande do Sul - PUCRS
claudio.scheer@edu.pucrs.br, jose.possebon@edu.pucrs.br

**Abstract**

Abstract.

Introduction.

## Related works

Related works.

## Deep Learning

In this section, we will discuss the sequence-to-sequence model using recurrent neural networks and transformers. In addition, we also discuss how a BERT model works.

### Sequence-to-sequence

The encoder-decoder architecture was initially proposed by (Cho et al. 2014). Although simple, the idea is powerful: use a recurrent neural network to encode the input data and a recurrent neural network to decode the encoded input into the desirable output. Two neural networks are trained.

(Graves 2013) - Generating sequences with LSTM

(Bahdanau, Cho, and Bengio 2015) - Proposed attention

(Vaswani et al. 2017) - Attention is all you need

### BERT

(Devlin et al. 2018) - BERT

Similarly to the original sequence-to-sequence model using a recurrent neural network, the model discussed in this paper uses two BERT neural network: one neural network to encode the input and another to decode the input encoded.

## Dataset

As we focused our project on automatic email reply, we used The Enron Email Dataset[1] to train our model. The dataset contains only the raw data of the emails. Therefore, we created a parser[2] to extract the email and the replies from the raw data of the email.

To identify whether an email has a reply or not, we look for emails that contain the string `-----Original Message-----`. After filtering only emails with non-empty replies, we parse those emails in an input sequence (the original email) and in the target sequence (the reply email). The entire extraction was done automatically, that is, we did not manually extract or adjust any email.

We used two libraries to parse the dataset: `talon`[3], provided by Mailgun, and `email`, provided by Python. The `email` package returns the email body with the entire thread. To extract only the last reply from an email thread, we use the `talon` package.

The original dataset contains $517\,401$ raw emails. After parsing the raw dataset, we created a dataset with $110\,205$ input and target pairs. As we have limited resources available, we limit the length of emails to $256$ characters. We ended up with $40\,062$ emails with reply in the dataset.

Of that total, we selected $10\,002$ emails to train the sequence-to-sequence model. From the rest of the emails, we chose random emails to evaluate the model, as shown in Section Results.

## Implementation

We use a pre-trained BERT model, provided by Hugging Face[4]. Hugging Face also provides a PyTorch library for using the pre-trained models. Therefore, this library was used to implement the sequence-to-sequence model.

To train the model, we use the following hyperparameters:

- Learning rate: $3\mathrm{e}{-5}$;
- Epochs: $128$;
- Adam epsilon: $1\mathrm{e}{-8}$;
- Batch size: $8$;
- Beam search hypothesis: $3$;

The batch size was limited by the amount of memory available on the GPU used to train the model. A TITAN X with 12 GB of memory was used. $8$ was the highest possible number.

Different values were tested for the learning rate and epochs. We did not change the default value for Adam epsilon. We also tested to accumulate gradients from more than one batch, but did not get good results.



[1] https://www.kaggle.com/wcukierski/enron-email-dataset

[2] https://www.kaggle.com/claudioscheer/extract-reply-emails

[3] https://github.com/mailgun/talon

[4] https://huggingface.co/

Training the sequence-to-sequence model in less epochs resulted in a model with a lot of noise in the reply email. As we had a limited batch size, increasing epochs increased the training time considerably. Training time is also the reason why the entire dataset was not used. The final model took about $6$ days to train.

When we increased the learning rate to $1\mathrm{e}{-4}$, the model converged faster until the value of the loss function was about $2^5$. After that point, the loss started to increase and decrease according to the batch. This same behavior is true when using $3\mathrm{e}{-5}$ as learning rate. However, the difference is that the loss function starts to have this behavior when the loss value is closer to zero.

The last changed hyperparameter was the number of hypothesis explored in each branch of the beam search. We tested a number greater than $3$. However, higher values resulted in some words being out of context in the generated reply email.

## Results

The evaluation of the model was

https://huggingface.co/blog/how-to-generate

## References


[Bahdanau, Cho, and Bengio 2015] Bahdanau, D.; Cho, K.; and Bengio, Y. 2015. Neural machine translation by jointly learning to align and translate. In Bengio, Y., and LeCun, Y., eds., *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

[Cho et al. 2014] Cho, K.; van Merrienboer, B.; Gülçehre, Ç.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR* abs/1406.1078.

[Devlin et al. 2018] Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR* abs/1810.04805.

[Graves 2013] Graves, A. 2013. Generating sequences with recurrent neural networks. *CoRR* abs/1308.0850.

[Vaswani et al. 2017] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is all you need. *CoRR* abs/1706.03762.


[5] We forgot to capture the loss values over the training epochs. And, as the GPU is shared with other research groups, and we had monopolized the GPU for about three weeks, we were unable to run the experiments again.