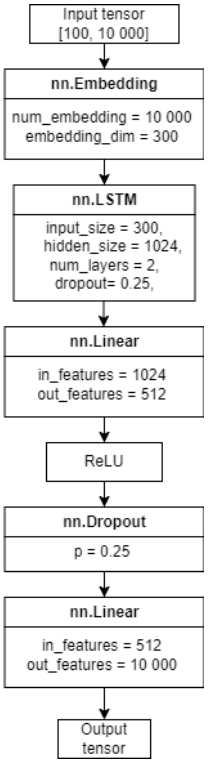
Solution Building Report

Baseline: Simple LSTM

Idea: In one of the lab exercises, we used LSTM for part of the speech tagging. But what will happen if, instead of parts of speech, we predict detoxified words. Therefore, my first model will be a simple LSTM with a fully connected linear layer at the end for word prediction.

Model architecture:



Metrics:
average BLEU: 0.02123
average TER: 2.4416
average ROUGE1 F1: 0.339749
average ROUGE2 F1: 0.154285
(Calculate on subset of ParaNMT with 1000 text pairs)

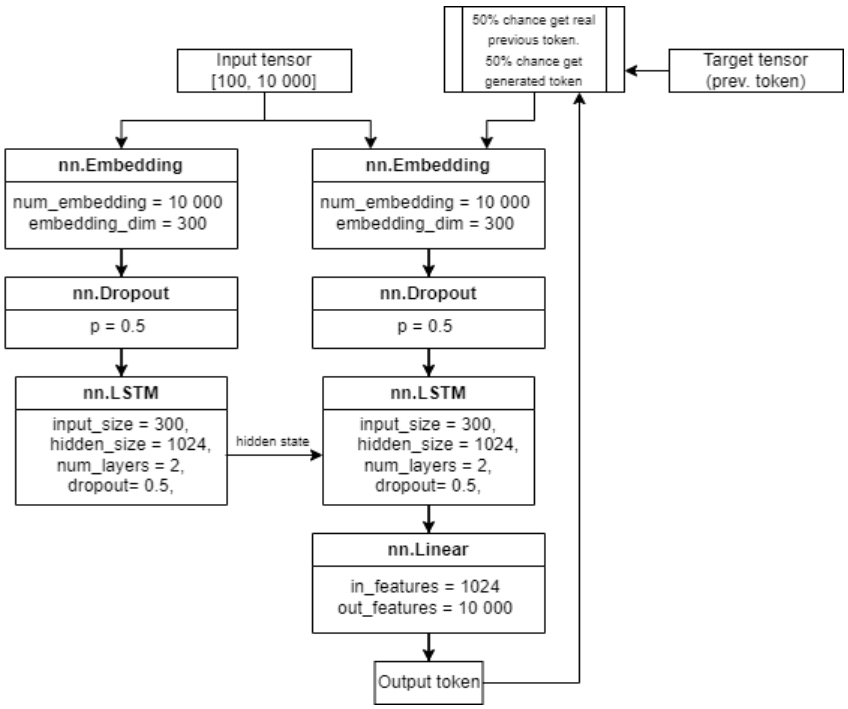
Detoxification example:
Input: you are very stupid.
Output: you 're very bad . <eos>

Hypothesis 1: Encoder-decoder LSTM without attention mechanism

Idea: Maybe the better strategy will be to analyze the whole toxic sequence and map it to some dimension and then generate detoxified text sequence token by token from this hidden dimension. Like and Encoder-Decoder architecture. In this case we will carry the full context of the given text.

Problems: The main problem I encountered when testing a hypothesis with LSTM was the model training time. Training 5 epochs on 2x T4 video cards took me about 10 hours. The result was also not very good. The model often generated a <unk> token. Overall scores were not much better than simple LSTM.

Model architecture:



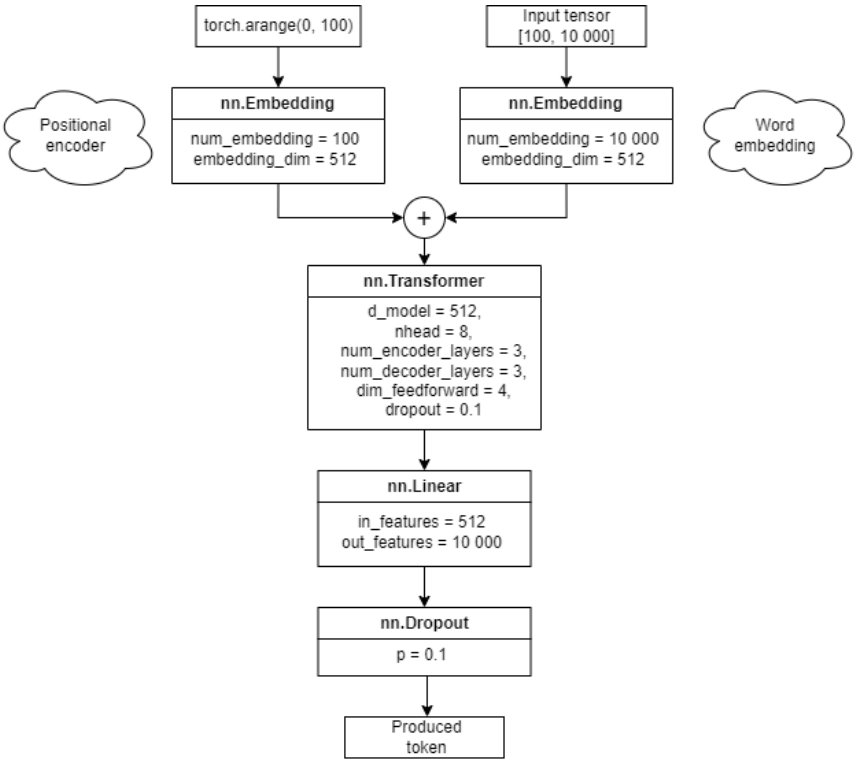
Metrics:
average BLEU: 0.00249
average TER: 0.0644
average ROUGE1 F1: 1.5939
average ROUGE2 F1: 0.23202
(Calculate on subset of ParaNMT with 1000 text pairs)
Detoxification example:
Input: you are very stupid.
Output: <unk> you 're too young .

Hypothesis 2: Custom transformer

Idea: To further improve the quality of detoxification, it is necessary to have an attention mechanism. But the problem of time still existed, so it was decided not to train recurrent models, but to go straight to transformers.

Problems: The long training time problem was only half solved. Now 15 epochs took about 7 hours to train, which is better than LSTM training but still takes a long time. In addition, although the quality has increased significantly, it is still far from production levels. However this could be solved with longer training.

Model architecture:



For implementing the transformer I used PyTorch nn.Transformer layer. The architecture was inspired by this youtube video.

Metrics:
average BLEU: 0.066942
average TER: 1.08759
average ROUGE1 F1: 0.54086
average ROUGE2 F1: 0.30527

(Calculate on subset of ParaNMT with 1000 text pairs)

Detoxification example:
Input: you are very stupid.
Output: you 're very unreasonable .

Hypothesis 3: Pre-trained transformer model

Idea: We can look for large transformer models that are already pre-prepared for our task. With this approach, we do not need to train and build our own model from scratch, which will take a lot of time and resources. Moreover, we can achieve better results since such models were trained over a longer period of time.

Model architecture: Architecture is T5-base with weights from t5-paranmt-detox

Metrics:
average BLEU: 0.08652
average TER: 0.64247
average ROUGE1 F1: 0.56947
average ROUGE2 F1: 0.34273
(Calculate on subset of ParaNMT with 1000 text pairs)

Detoxification example:
Input: you are very stupid.
Output: You're very naive.