Solution Building Report

As a solution to the Text Detoxification Problem I have considered the following ideas:

- Using a ready solution made by Scoltech. This seemed like cheating, though. Besides, it is quite a complicated model to use, and I do not have enough computational resources to train and use the model.
- A transformer-based neural network. This approach would also be quite complex and unsuitable for the little computational power I have; it would be ridiculously slow to fine-tune even a pre-trained LLM in google colab. Therefore, I decided not to use this approach.
- Another approach would be to determine the exact words and phrases that are associated with a toxic style and replacing them with non-toxic alternatives. It is not clear, however, how to do so. Moreover, replacing toxic words might require changing the whole sentence structure for the sentence to be grammatical.

As a result, I decided to use an LSTM-based language model. Although it would be not as powerful as using transformers and attention, it seemed to be a reasonable compromise between complexity, requirements to the computational power, and accuracy.

The next step was clear enough: the model can work only with numbers and not with words and sentences. So I decided just to read all the words from the dataset into a set and then create a mapping from words to integers and a mapping from integers back to words.

I had two different options how to preprocess the raw sentences:
- I could leave them as they are,
- Or I could implement the whole NLP preprocessing pipeline that includes stop-words removal, punctuation, stemming, lemmatization.

Since the output of the model should be a grammatical sentence, the second option is not feasible. Therefore, an optimal solution seemed to be just to cast the words to lowercase and to encode the punctuation separately. This approach would produce satisfactory output, and the set of vocabulary would still be smaller than compared to encoding raw text.

The next step is to choose the neural network architecture. It was at this moment when I knew I'm still low on computational power. Choosing a larger and more expressive NN

would come at a great cost of running out of GPU quota and time. Choosing a smaller NN would produce terrible results.

Right before I started randomly guessing the best NN structure, I found out that I cannot use the whole dataset - google colab just runs out of memory. I had no choice but to cut the training dataset by 70%.

After some experiments with this subset, I found out that the NN does not have to be large. Setting the number of LSTM layers to values greater than one already produced garbage output. Also, neither the number of fully connected layers, nor their size did not have much effect on the output, so the existing NN of 32 embedding layers, a single poor LSTM layer and hidden size of 512 looked really attractive.

One thing I realized while doing all of this is that if all the sentences are padded, and the loss function considers the whole sentence including padding, then it would be possible (and even reasonable) to just ignore padding then training the model. The nn.CrossEntropyLoss() even has a most convenient argument: ignore_index.

With this argument, however, the sentences were bogus. So I made use of two loss functions: if the padding was predicted correctly, I ignored it; if there was even the slightest of a mistake, I calculated the loss on the whole sentence.

I have no means of testing the hypothesis whether this works better or not, but it felt like the right thing to do.

That was a brief story of how I came up with the solution. Thank you for reading it until the end :)