1. Building Solution

The second step is to come up with solutions. I decided that it was worth taking three solutions, one of which would be simple, that is, a baseline. And the other two will have more complex architecture.

1.1 Baseline (BertClassifier) Solution

Since we have already seen what our data consists of, we can already build a text detoxification solution. First we need some kind of baseline. I wanted to do something similar to the removal method in the *ParaDetox: Detoxification with Parallel Data* article, which you can find at *./references/ParaDetox Detoxification with Parallel Data.pdf*

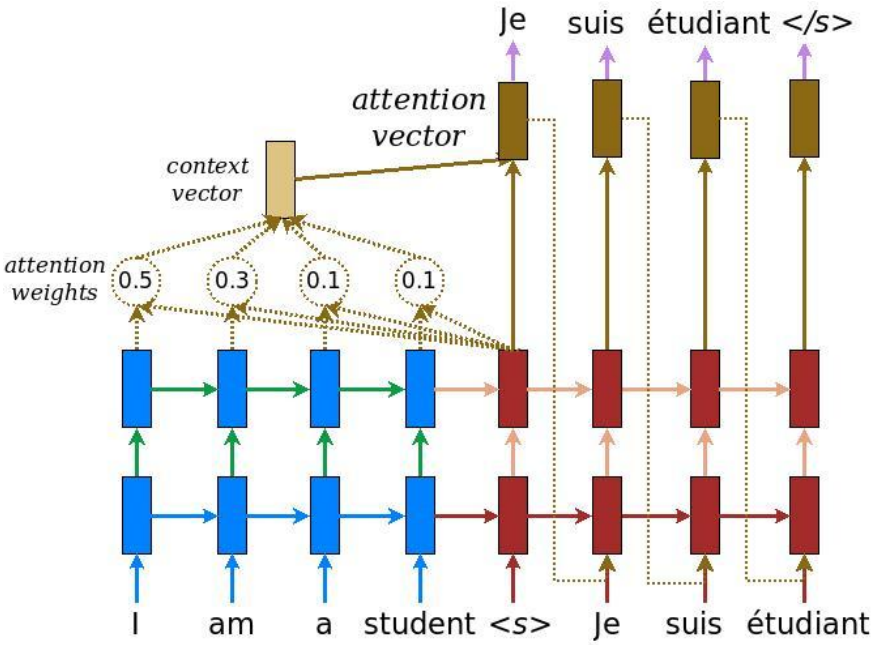
I decided to take the BERT. It's is a bidirectional transformer pretrained using a combination of masked language modeling objective and next sentence prediction on a large corpus comprising the Toronto Book Corpus and Wikipedia. So, I want to train a BERT classifier, then find the words that most influence the toxicity decision and remove the words one at a time until the classifier determines the sentence to be non-toxic.

The architecture of my BERT classifier looks like this: First I take sentence embeddings and then pass it through 1 linear layer with 2 outputs. And then I use softmax to get the probabilities of belonging to the toxic and non-toxic class.

As a dataset, I decided to take the very popular Jigsaw dataset with toxic comments. The EDA of this dataset you can see in 1.5 Section. Since data is unbalanced, I decided to do downsampling, so I took 20.000 samples for training, where 10.000 are toxic and 10.000 are non-toxic. For validation I took 2000 samples, where 1000 are toxic and 1000 are non-toxic.

1.2 Encoder-Decoder Solution

Here I am going to perform machine translation using a deep learning based approach and attention mechanism. I implemented an encoder-decoder model with attention using PyTorch from the official documentation of TensorFlow, also I took Information from an NLP course of Innopolis University. The following diagram shows that each input word is assigned a weight by the attention mechanism which is then used by the decoder to predict the next word in the sentence.



The goal of the encoder is to process the context sequence into a sequence of vectors that are useful for the decoder as it attempts to predict the next output for each timestep. I used a GRU to do the processing.

The encoder:

- Takes a list of token IDs
- Looks up an embedding vector for each token (Using a nn.Embedding).
- Processes the embeddings into a new sequence (Using a GRU).
- Returns the processed sequence. This will be passed to the attention head.

The attention layer lets the decoder access the information extracted by the encoder. It computes a vector from the entire context sequence, and adds that to the decoder's output. I was using Bahdanau's attention.

The decoder's job is to generate predictions for the next token at each location in the target sequence.

- The decoder receives the full output of the encoder.
- It uses GRU to keep track of what has been created so far.
- It uses its GRU output as a request for attention to the encoder output, creating a context vector.
- It combines the GRU output and the context vector to create an “attention vector.”
- It generates logit predictions for the next token based on the "attention vector".

As a dataset I took the ParaNMT-detox dataset. I splitted it into train and val sets. They contain 462.221 and 115.556 samples respectively.

1.3 T5-small Solution

T5 is an encoder-decoder model pre-trained on a multi-task mixture of unsupervised and supervised tasks and for which each task is converted into a text-to-text format. I took Information from the PMLaDL course of Innopolis University. I chose a small size T5 for the fine-tuning. The article you can find here.

As a dataset I took the ParaNMT-detox dataset. I splitted it into train and val sets. They contain 462.221 and 115.556 samples respectively. Same as 2.2 Section.

2. Training

2.1 Baseline (BertClassifier) Solution

So, I took this configuration for training. The max_length of the comment is 120, which is twice as much as the average number.

```
EPOCH_NUM = 1
lr = 2e-5

bert      = BertModel.from_pretrained(bert_model_name)
model     = BertClassifier(bert, 2).to(device)
loss_func = nn.BCELoss()
optimizer = optim.AdamW(model.parameters(), lr=lr)

# The triangle learning rate advances linearly until half of
the first epoch, then linearly decays.
w_steps = 10 ** 3
t_steps = len(train_iterator) * EPOCH_NUM - w_steps
scheduler = get_linear_schedule_with_warmup(optimizer,
w_steps, t_steps)
```

It took 03:58 for training and gives me ROC_AUC = 0.908 for val jigsaw dataset.

2.2 Encoder-Decoder Solution

So, I took this configuration for training. I took embedding_dim = 256 since the max number of words is 253, and my machine can train it.

```
embedding_dim = 256
units = 512
```

```
BATCH_SIZE = 32
EPOCHS = 1

vocab_inp_size = len(inp_lang.word2idx)
vocab_tar_size = len(targ_lang.word2idx)

encoder = Encoder(vocab_inp_size, embedding_dim, units,
BATCH_SIZE)
decoder = Decoder(vocab_tar_size, embedding_dim, units, units,
BATCH_SIZE)
optimizer = optim.Adam(list(encoder.parameters()) +
list(decoder.parameters()),
                       lr=0.001)
```

It takes 6:05:13 for training.

2.3 T5-small Solution

So, I took this configuration for training.

```
args = Seq2SeqTrainingArguments(
    output_dir=SAVE_PATH,
    overwrite_output_dir=True,
    num_train_epochs=1,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    warmup_steps=300,
    weight_decay=0.01,
    learning_rate=3e-5,
    logging_steps=1000,
    eval_steps=1000,
    evaluation_strategy='steps',
    save_total_limit=1,
    save_steps=1000,
)
```

It takes 1:05:49 for training.

3. Inference

Here is one example of inference. You can find it in *./notebooks/4.*-model-inference-*.ipynb* or run *./src/models/predict_model_*.py* files by yourself.

	Baseline	Encoder-Decoder	T5-small
i am fucking cool.	i am cool.	i m a good thing.	I'm cool.

4. Validation

For the validation I took val set ParaNMT-detox, that I got in Section 1.2, which contains 115.556 samples, but I used only 5000 samples from it for fast computation.

Metrics, that I used are:

- Style accuracy (STA) - percentage of nontoxic outputs identified by a style classifier. HF model is *"SkolkovoInstitute/roberta_toxicity_classifier"*
- Content preservation (SIM) – cosine similarity between the embeddings of the original text and the output computed with the model. HF model is *"sentence-transformers/all-MiniLM-L6-v2"*
- Fluency (FL) – percentage of fluent sentences identified by a RoBERTa-based classifier of linguistic acceptability trained on the CoLA dataset. HF model is *"textattack/roberta-base-CoLA"*
- Total - average of STA * SIM * FL

I was inspired by the article *ParaDetox: Detoxification with Parallel Data* article, which you can find at *./references/ParaDetox Detoxification with Parallel Data.pdf*

Results:

	STA	FL	SIM	Total
BERT Deletion	**0.953184**	0.471775	0.641421	0.292803
Encoder-Decoder	0.651788	0.533892	0.549632	0.197177
T5-small	0.612354	**0.739552**	**0.674842**	**0.315725**

The best in terms of STA is our baseline, this is due to the fact that it simply throws out all toxic words, so the value is close to 1, but its FL is the smallest due to the simplicity of the solution. And since the number of ejected words is not so large, his SIM is also quite large.

The encoder-decoder model performed averagely, and in some cases even worse than other models. It seems to me that this is due to the fact that she was trained only for 1 era and this was not enough for her, that is, she was undertrained. The small value of SIM suggests this idea to me, but this is not certain.

As you can see, T5-small showed the best result. Although it has the smallest STA, by other indicators it is the largest. Instead of simply deleting words, he rephrases, therefore his SIM

and FL are the highest. And its STA is low due to the fact that it does not remove toxicity from the text, but makes it less rude, it seems to me.