Flood Transformer in Telegram

Yaroslav Yudinskikh
Innopolis University / Innopolis, Russia
Data Science
`github.com/simonwt/flood-transformer-bot`
`y.yudinskih@innopolis.ru`

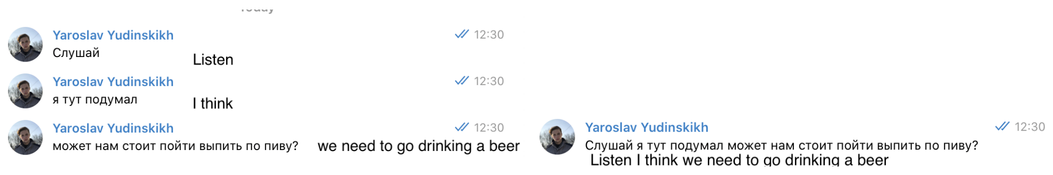
Abstract

This document contains a description of the project assignment for the NLP course. During this work, I described the process of creation telegram bot, which collect the consecutive messages from one participant in the chat which was sent in a small period of time. This project leads to deal with spam which some chat participants produce by writing one sentence in a lot of messages.

1 Introduction

Nowadays people participate in a lot of chats in their messengers. And some of the chat participants often write one sentence in a lot of messages. This is flooding. That why we need to restrict people to not doing that and write their thought in one message. Otherwise, we could create a bot that collects those messages and transform into one ease understandable, not taking up much height message.

The bot should collect the consecutive messages from one participant in the chat which was sent in a small period of time. Then pass it message by message to the language model which will predict whether two messages should be connected, if yes - merge pairs and pass compare it with a text message, else don't do anything with messages.


Yaroslav Yudinskikh 12:30
Слушай
Listen

Yaroslav Yudinskikh 12:30
я тут подумал
I think

Yaroslav Yudinskikh 12:30
может нам стоит пойти выпить по пиву?
we need to go drinking a beer

Yaroslav Yudinskikh 12:30
Слушай я тут подумал может нам стоит пойти выпить по пиву?
Listen I think we need to go drinking a beer

Figure 1: Example of spam and expected result of the bot (English translation provided).

2 Background

Unfortunately, I did not found similar works or products, which will provide the functionality which I expected from my work. Nevertheless, for predicting the end of a message we need to learn the language model. For this purpose I found article (Using the Output Embedding to Improve Language Models, 2016) which describes recurrent neural network with LSTM.

3 Method

During this project, I collected the data, experimented with preprocessing and learned chosen model. The difficult and biggest part was the collection of training data.

3.1 Baseline

The goal of the project is to learn a model to predict whether its end of a meaningful message or not end and it needed to be merged with a previous message. To do this we need to build the language model, which was learned on some real chat data. This model should predict the next word using the text of one message. If the word, which recognizes the end of the message, which we put input the training data, have the big probability to selected by the model, it means that this is the end of the meaningful sentence and we can merge messages without that end of the message word. Therefore, the model should learn words by the context, which means we could use LSTM with RNN. This is exactly what I used in this project

This is a simple RNN - LSTM model, with simple architecture. Just Dropout layer, then Embedding, LSTMs and Linear as a decoder. The model was taken from public repository (Word-level language modeling RNN, 2018) and it based on the original paper (Using the Output Embedding to Improve Language Models, 2016).

3.2 Data

Of course, for the project is very hard to find good quality-related data which will consist of chat message with the problem which the project tries to solve.

First of all, I looked in the direction of parsing text from Wikipedia on Russian, there exist a lot of open datasets with this data on the Kaggle and Github repositories. But it consists of text almost in formal style, unfortunately, it is not suitable for our case because most of the chat text represent informal style. This may complicate the model's task.

Furthermore, I opened the chat with my classmates and found out that this is really good data, it consists of the spam which we tried to avoid with the bot and fully sentence messages. the main drawback, in this case, is the dirtiness and collection of the data, but it could be solved by preprocessing and some libraries.

I collect the last 10000 messages from the chat using (Telegram Message Dump, 2019) python package. It uses account credentials to authenticate and provided chat name to select appropriate chat, and export the messages in CSV format with the timestamp, username and the content.

3.3 Approach

3.3.1 Preprocessing

I normalized the data, tokenized them and removed stop words from the text. Also, I needed to determine whether the message is part of one sentence divided into several messages or its complete thing message. To solve this problem I added the ¡eom¿ end of a message token. As the data is very big I wrote the small function which iterates through all message in chronological order and collects the message from each person in the window of 18s, after each pull was collected I add the ¡eom¿ token to the last message in this pull.

Also, I recognized that last messages in those pull or self-sufficient messages often end on emoji, it means that we don't need to remove emoji symbols during stop words removing and recognize messages with this symbols on the end as an end messages.

After preprocessing, I create the corpus and the dictionary. And got this:

- Messages before preprocessing: 10000
- Messages after preprocessing: 8569
- Dictionary size: 2600
- Corpus size: 3519
- Basically the all messages are in Russian but English words are also present

The number of messages was reduced because after preprocessing some messages was empty, in this case, I was needed to delete these messages from the dataset.

3.3.2 Deep learning

As mentioned in the Baseline subsection I used the RNN model with the LSTM module. During the deep learning I tried different hyper-parameters but finished on the standard ones because they help me to achieve readable results.

- Embeddings with dimmension 200. (It is large enough to cover the entire vocabulary and small enough to avoid the curse of dimensionsionality.)
- 650 hidden units per layer.
- 0.5 dropout probability to avoid overfitting.
- Batch size 12 to increase learning speed and average the gradient step.
- 0.25 gradient clipping to solve exploding gradients problem.

3.3.3 Telegram Bot

Further, I need to apply this trained model in the real application, which will interact with Telegram API and transform several spam messages to the good one. The easiest and a good way to do this is to create the chat bot which will the participant of the chat conference. It should read all messages, pass it to the model, choose the predictions with biggest probability. If it includes the ¡eom¿ token, then bot should edit the first message in the user's pull and concatenate it with the messages which were between fist one and the message with predicted ¡eom¿ token in the end, then delete all messages from the pull except first one. For implementing this bot I used very common python library (Python Telegram Bot, 2015).

... (Unfortunately this part not done due to lack of time) ...

4 Analysis

During the learning I saw that the model on first several epochs reached the some validation loss value, and further not decrease it radically. I tried to shuffle the data and not include emoji and it a little bit improved situation. I think that my preprocessing and labeling is not really good as a the quality of the training data. May be with better dataset model could show better results.

By the way trained language model of course with not awesome accuracy predict the end of the message. So it could be applied to the real telegram bot.

5 Conclusion

Language Modeling is one of the most interesting direction in NLP. Modern solutions in this area allow Software/ML engineers to create AI which will help to reduce notification and messaging noise in our daily life. I really enjoyed this project because I touched all stages of NLP engineer work: parse the data from the uncommon source, preprocess it, feed to AI and finally relish on the inference. Also, I meet problems with data labelling and model learning.

References

Using the Output Embedding to Improve Language Models. 2016. https://arxiv.org/abs/1608.05859

Word-level language modeling RNN. https://github.com/ceshine/examples/tree/master/word_language_model

Telegram Message Dump. https://pypi.org/project/telegram-messages-dump/

Python Telegram Bot. https://github.com/python-telegram-bot/python-telegram-bot