Milestone 2, Chat-MMA

Antoine Magron, Mathieu Desponds, Mekhron Bobokhonov

May 2023

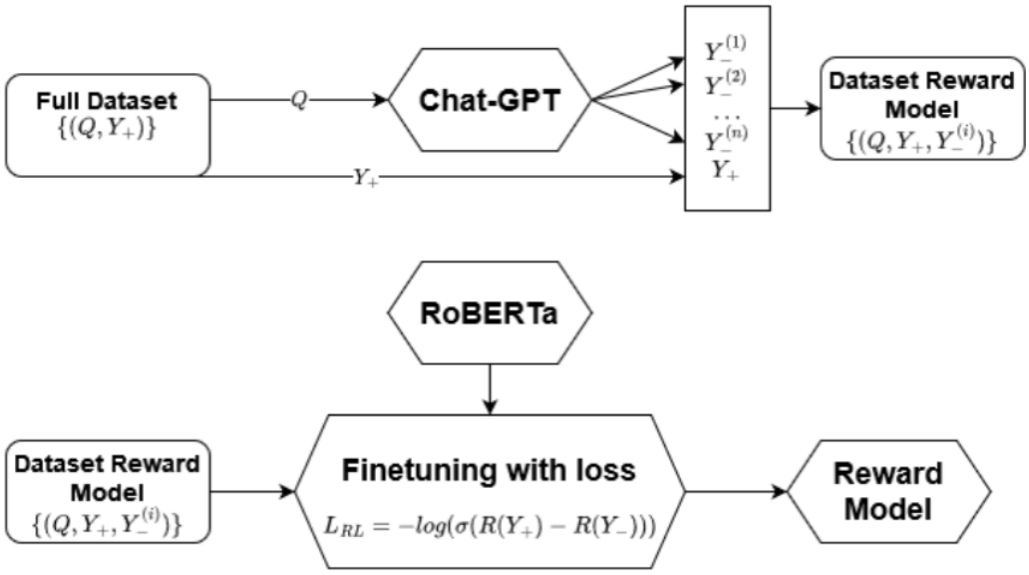

Figure 1: Milestone 2 plan

Figure 1 describe the steps that were done for this second milestone. The top part describes how we build the dataset and will be described in the first part. The lower part describes how we trained the Reward Model using the previously obtained data and will be described in the second part of this report.

1 Intro

In Milestone 2, we needed to build a reward model that would be used for the RLHF technique in Milestone 3. In Figure 1, we presented a general scheme for building our reward model. The reward model takes as input a prompt and a response of the ChatGPT and outputs a scalar reward. We trained RM on a dataset of comparisons between two model responses to the same question. As a result, the reward model gives a higher score to well-structured responses rather than poorly structured ones. In the following sections, we will discuss our dataset and model training in more detail.

Description of the deliverables :

- `model.py` ; Contains the implementations of the reward model and its associated config class
- `reward_model_full_partitioned.json` : File containing 3 slices, *"train"*, *"eval"*, and *"test"* of questions not preprocessed.
- `data/processed_test_data_full.json` : contains the preprocessed test samples (of the form {`"chosen": str, "rejected": str`}). These are the samples that can be feed into the `get_rewards` function.

2 Data sources

This part is described in the top part of Figure 1. To train the Reward model, we build a dataset consisting of tuples $(Q, Y_+, Y_-^{(i)})$ where Q is a question, Y_+ is a good answer to the question and Y_- is a bad answer. To do so, we will use the set of question associated given during Milestone 1.

2.1 Collecting Y_+

From the Milestone 1 dataset, we desire well-structured answers that can effectively train the reward model. Therefore, we begin by filtering out answers with a confidence level below 3, containing less than 6 sentences, and consisting of fewer than 500 characters. These choices are made to strike a balance between retaining only highly satisfactory answers and ensuring an adequate amount of data. The graph that helped us take such decision can be seen in Figure 2.1. Following this filtering process, we retain 6068 samples and discard 4767 samples. The discarded samples correspond to 3045 different questions, and due to the filtering, 942 questions remain unanswered, which is undesirable. To address this issue, we decide to generate three answers using our three prompt strategies for the unanswered questions and regenerate one answer using our best prompt strategy for the remaining questions from which samples were discarded. Consequently, our resulting dataset takes the form (Q, Y_+) and comprises 10948 samples.

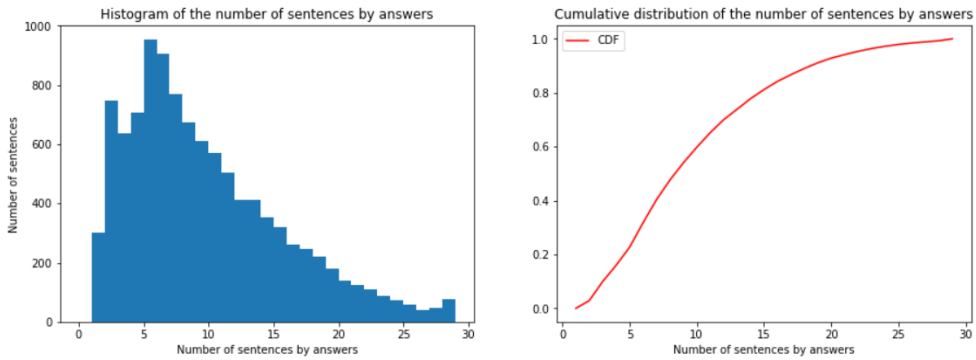

Figure 2: Histogram and distribution of the number of sentences of the Milestone 1 dataset.

2.2 Collecting Y_-

In order to collect Y_- responses we created three baseline prompts.

For Multiple Choice Questions

1. **instruction:** Develop each possible solution saying if they can be true or false and why it is the case. Finally give the number of the final and unique answer that is true in any cases.

   prompt: There is a multiple choice question with len('choices') possible solutions. Here is the question : 'question'. Here are the answers : 'answers'

2. **prompt:** You are taking an exam for a university course. You will be given a multiple-choice question and your task is to select only one correct answer from the offered list of options. Use the following format: Question: "' question here "' Choices: "' here is a list of possible answers "' Correct Answer: "' Here is the correct answer that you chose from the offered list. format: variant number. answer "'

   Explanation: "' Here is an explanation and justification for each option why it's true or false. format: variant number. "TRUE" or "FALSE": Your explanation "'

3. **instruction:** You a teacher in an insitute of technology. Your goal is to answer student's question and help them understand the answer. It needs to be very clear. You will list the proposition and whether they are true or false. You will end by stating the right answer among them and justify this one is correct.

   prompt: Here is a multiple choice question with len('choices') possible solutions. Here is the question : 'question'. Here are the choices : 'answers' """

For Simple Questions

1. **instruction:** Answer to the question and say why it is your answer

   prompt: 'question'

2. **prompt:** You are taking an exam for a university course. You will be given a question and your task is to answer correctly to the given question. Use the following format: Question: "' question here "' Correct Answer: "' your answer here "' Explanation: "' here is an explanation and justification of your answer. "'

3. **instruction:** The same instruction as in 3 prompt for MCQ.

   prompt: Here is the question : 'question'.

We divided the questions into 3 groups and used a corresponding prompt for each group. For each question, we generated one Y_- and got a total of 4446 pairs (Q, Y), a question and a bad answer.

2.3 Formation of the final dataset

Next, we merge (Q, Y_+) and (Q, Y_-) to obtain our final reward dataset in the form of (Q, Y_+, Y_-), comprising 10936 samples. Finally, we divide the dataset into three portions: $dataset_{train}$, which accounts for 70% of the dataset; $dataset_{eval}$, which constitutes 15%; and $dataset_{test}$, which also comprises 15%.

From this we generate `only_test.json` file made to be use with `evaluate.py` that takes as input an array of dict with keys "chosen" for the good answers, and "rejected" for the bad ones. For both field we want also to add the question alongside the answer so that the model learn to link what is a good answer related to the question. Therefore each sample in `only_test.json` are of the form

$$\{\texttt{chosen} : [CLS] + Q + [SEP] + Y_+ + [SEP], \texttt{rejected} : [CLS] + Q + [SEP] + Y_- + [SEP]\}$$

where $[CLS]$ and $[SEP]$ are the classification and seprator tokens.

Another consideration for the dataset involves the maximum limit of 512 tokens imposed by the Roberta-base model. Some answers, when tokenized, exceed this limit due to containing a significant number of mathematical expressions. However, upon examining these samples, we observed that only a few of them exceeded the 512-token threshold as we can see in Figure 2.3. Hence, we decided to truncate those samples thinking it would not have a substantial detrimental effect.

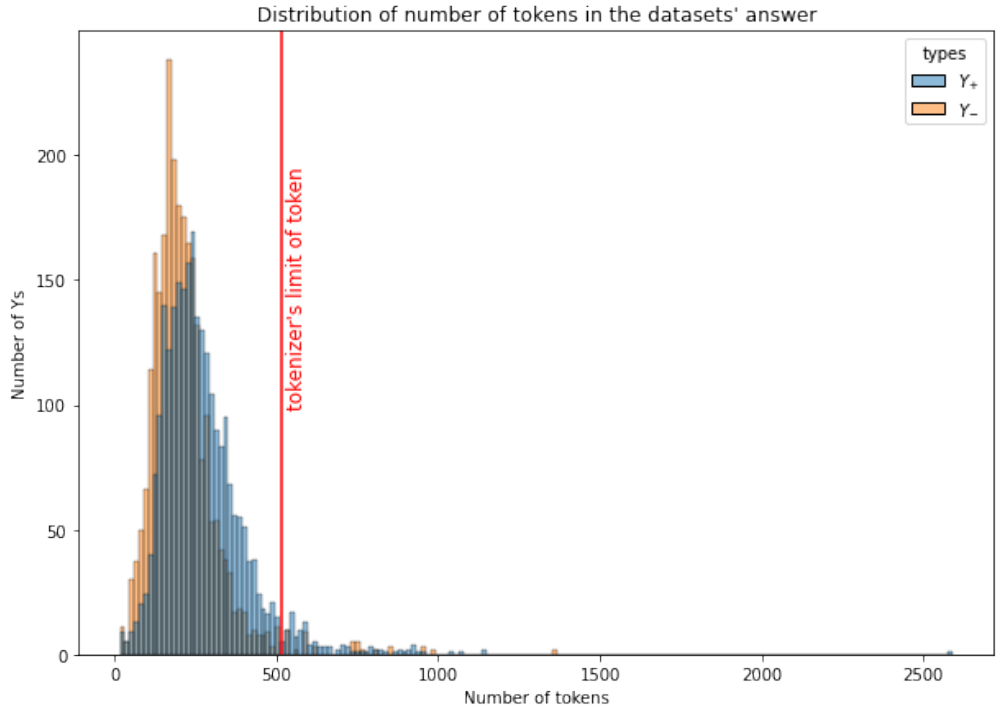

Figure 3: Distribution of the number of tokens in the datasets' answers.

3 Model

The model is implemented in `model.py` in the `RewardModel` class. It is composed of two inner models :

- A Bidirectional Transformer Encoder. We feed it the tokenized preprocessed input sentence and we keep the embedding of the `[CLS]` token that contains the embedding of the full sentence.
- A Feed Foward Network that takes the sentence embedding, and outputs a single number. The dimensions of the hidden layers, and the activations functions used in the model are modular and contained in the `RewardModelConfig` class.

After a few tests, we chose `RoBERTa-base` for the language model and a Feed Forward Network with 3 hidden layers. The input layer is the size of `RoBERTa`'s hidden dimension, the three hidden layers have sizes 128, 64, and 16 respectively. The activation function between each of these layers is ReLU. We then finally output a single number and apply no activation function to be able to have a greater range of reward function modeling.

We trained the model following the instructions described in this paper [1] using the following loss function:

$$\mathcal{L}_{RL} = -log(\sigma(R(Y_+)) - \sigma(R(Y_-))))$$

For Y_+ being the chosen input and Y_- being the rejected input. This loss is made to maximize the difference in reward between good and bad demonstrations. To evaluate our models we use 3 metrics :

- The proportion of pairs (Y_+, Y_-) in the dataset for which $R(Y_+) > R(Y_-)$
- The average difference of reward between Y_+ and Y_-
- The standard deviation of reward between Y_+ and Y_-

The model can be loaded and used in the following way:

```
from model import (RewardModelConfig, RewardModel)
from transformers import (AutoConfig, AutoModel, AutoTokenizer)

AutoConfig.register('RewardModel', RewardModelConfig)
AutoModel.register(RewardModelConfig, RewardModel)
tokenizer = AutoTokenizer.from_pretrained(path)
config = AutoConfig.from_pretrained(path)
model = AutoModel.from_pretrained(path, config=config).to(device)

input_ids = prepare_demo("This is an example sentence", tokenizer)
model(input_ids)
```

We saved a baseline that is really undertrained (trained on a few samples only), has only sigmoid as activation functions, and has smaller hidden dimensions. The fine-tuned model has been trained on the previously introduced training dataset for 5 epochs.

We trained mutiple models to try to understand how the model learns. We report the score of three models :

- **Baseline** : The model has been trained on a subset of samples for only 1 epoch. It's expected that the performances are very low.

	Proportion of correct ranking	Average difference	Standard deviation in difference
Baseline (barely trained)	15%	-3.69×10^{-9}	3.51×10^{-8}
model_hf_97 (undertrained)	66%	0.18	0.93
model_hf_full (fully trained model)	94.4%	5.49	2.44

- `model_hf_97` : The model has been trained for 5 epochs on the first dataset containing a quarter of the final one. It already provides a fair amount of correct classification but the average difference is very low and doesn't set a significant difference in reward for good and bad answer.
- `model_hf_full` : The model is trained for 5 epochs on the full dataset. It has the best performances. It classifies most of the pairs correctly and after a manual inspection of the classification, some of the generated Y_- were close or better than some Y_+ due to generation hasards. The difference is pretty wide and allows for a wide range of reward in between the minimum and maximum. The regression is not binary.

We will keep the `model_hf_full` as the final reward model. Figure 4 displays the distribution of reward for the test samples. We can see the clear spikes at ~ -5 and ~ 2 and the range of obtained values in between. Figure 5 shows the distribution of the difference in reward between associated Y_+ and Y_-. We can see a clear peak around 6 and a uniform number of samples for the intermediate positive scores. There a few negative difference which corresponds to pairs (Y_+, Y_-) for which $R(Y_+) < R(Y_-)$.

References

[1] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," 2022.

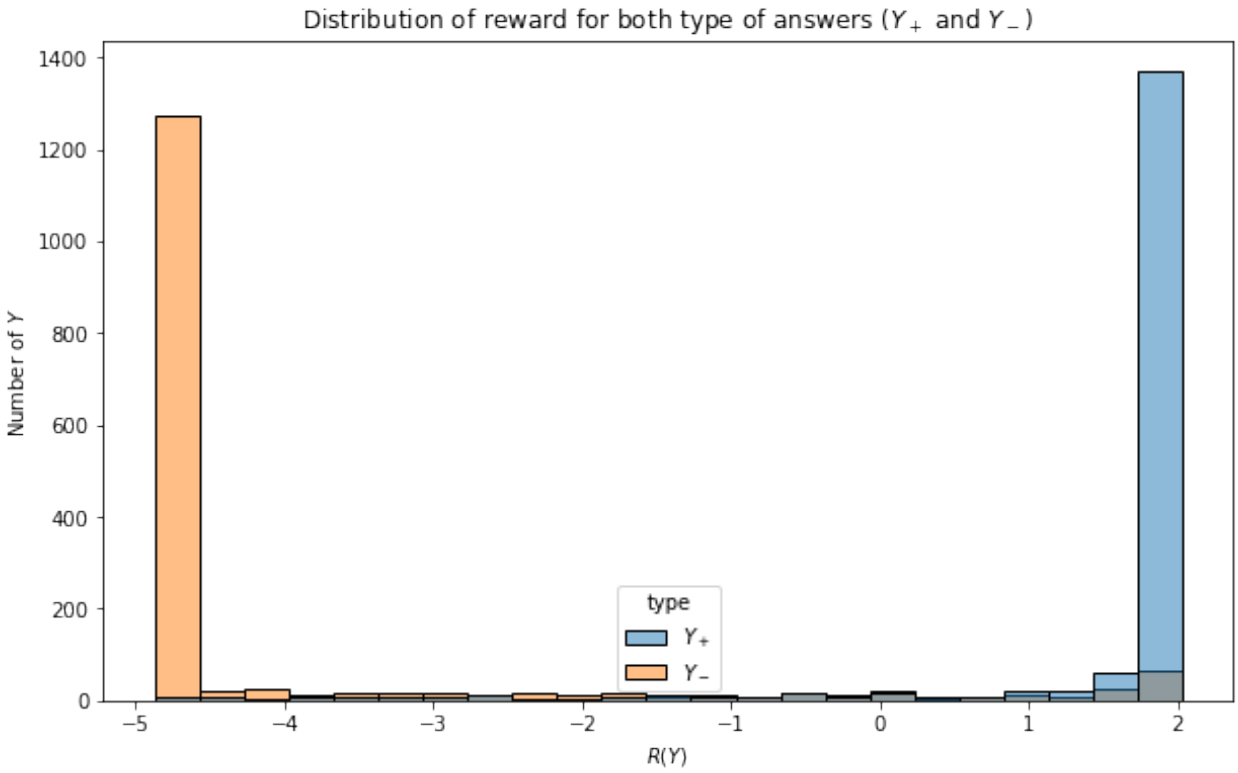

Figure 4: Distribution of $R(Y_+)$ and $R(Y_-)$

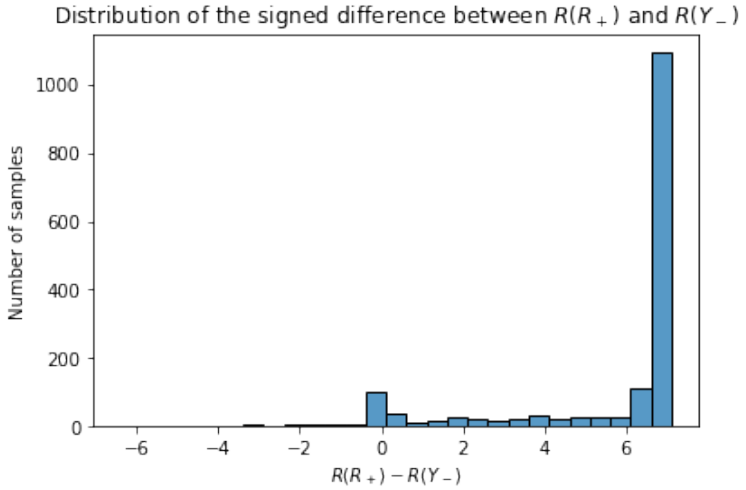

Figure 5: Distribution of $R(Y_+) - R(Y_-)$