NLP Course Project
Milestone 1: Project Plan

Antoine Bonnet
antoine.bonnet@epfl.ch

Silvia Romanato
silvia.romanato@epfl.ch

Alexander Sternfeld
alexander.sternfeld@epfl.ch

School of Computer Science
Ecole Polytechnique Fédérale de Lausanne

1 Data

For this project, we need data for two purposes. First, we require data tuples in the form *(prompt, winning_response, losing_response)* to train our reward model. Second, we want data tuples of the form *(prompt, correct response)* to fine tune our pre-trained model. Therefore, we aim at using the following three sources:

1. *Manually collected interactions with chat-GPT alongside the true answer.* For each of the 100 questions, we have collected three generated answers. Furthermore, for each answer we provided a confidence score, indicating how certain we are of the answer. By using these confidence scores, we can rank the answers. The highest ranked answer will be the ground truth, to which we also have access. This data can both be used to train the reward model and to fine-tune the pre-trained model.
2. *Stack Overflow Question and Answers.* As a second source, we will consider publicly available data from Stack Overflow, which is a publicly available question-and-answer platform. Specifically, we consider posts from the subjects *computer science* and *math*. The rich dataset contains information on the users, the posts, the comments and answers. Furthermore, it is indicated which answer is accepted by the user who posted the question. The data can both be used for the reward model and for fine-tuning the pre-trained model. We perform several steps of preprocessing for this data:
    - We retain only the posts that have an accepted answer. We choose to do this, as in these cases we can be fairly certain that the accepted answer is better than the other answers.
    - When training the reward model, we keep only the posts that have 2 or more answers. In this setting, we need more than 1 answer, as the reward model needs multiple (ranked) answers per question. When fine-tuning the pre-trained model, we can skip this step.
    - We can now compose our data points. For the training of the reward model, we compose pairs *(prompt, winning_response, losing_response)*, where the winning response is the accepted answer and we make a different pair for each other response. For the fine-tuning of the pre-trained model, we simply consider the pairs *(prompt, winning_response)*.
3. *Yahoo! Answers dataset.* As a third source, we will consider publicly available data from Yahoo! Answers, which is a publicly available question-and-answer platform. The dataset contains the topic, the question title, the question content and the best answer. We will filter the questions to only retain the questions belonging to the topics *Computers & Internet* and *Science and Mathematics*. This data will only be used to fine-tune the pretrained model, and will not be used to train the reward model.

Submitted to *CS-552: Modern Natural Language Processing* at EPFL in May 2023.

2 Strategy

2.1 Reward model

In Section 1, we described how we obtain our training data, which will be of the form *(prompt, winning_response, losing_response)*. For the training of the reward model, we will split the training data into a training set and a validation set. The validation set can then be used to evaluate the performance of the reward model. As a base model we use the RoBERTa model (*roberta-base*), which has been trained on approximately 160GB of English data, using a masked-language modeling objective [1].

Model architecture

The pre-trained RoBERTa model takes as input a piece of text and outputs an embedding of the text. Our aim with the reward model is to generate a scalar reward for a given input text. Therefore, we will put a regression head on top of the RoBERTa model. The input to the model will be of the form *(prompt, response)*, hence for each tuple in our training data we obtain the two inputs *(prompt, winning response)* and *(prompt, losing response)*. The model then outputs the scalar reward. Our aim is to have a higher reward for winning responses than for losing responses. This will be achieved through the loss function that is described next.

Loss function

The training samples are of the form *(prompt, winning_response, losing_response)*. When we have the model score s_w for the winning response and the score s_l for the losing response, we can compute the loss function as:

$$L(s_w, s_l) = -log(\sigma(s_w - s_l)), \tag{1}$$

where σ is a Sigmoid function. In other words, if the winning response gets a higher score than the losing response, the loss is low.

2.2 Final model

As our base model we have chosen the pre-trained GPT-2 mode, which has been trained on a corpus of 40GB of text data [2]. This model will both be used for our final generative model and for our reward model. For this purpose, we use both the Yahoo! Answers dataset and our own 100 questions with the ground truth answers. We will split this data into a train and test set, such that we will have a fair assessment of the performance.

Model architecture and fine-tuning

The GPT-2 model is an autoregressive model, which means that we do not have to provide targets for the fine-tuning stage. To illustrate, when we give the sentence *"Bob likes cookies"*. The model first considers only the word *"Bob"* and generates the second word *"X"*. It then considers the words *"Bob X"* and generates the third word. In our context, to train the model we provide the training samples *(prompt, correct response)*. However, before doing so we will tokenize the text and truncate the input to the maximum length of 1024.

The default loss function of the model is the negative log-likelihood. The model outputs a probability distribution over the whole vocabulary. The loss is then calculated as minus the logarithm of the probability of the true token.

3 Evaluation

3.1 Evaluation of the reward model

For the evaluation of the reward model, we will use the validation set of tuples *(prompt, winning_response, losing_response)*. We will split each tuple into two inputs *(prompt, response)* and generate scalar rewards. We consider the response with the highest scalar reward as the predicted winning response. We can then compute several metrics, specifically the accuracy, precision, recall and F1-score.

Furthermore, we can do a more qualitative analysis, by considering the distribution of the scalar rewards. Several interesting questions are:

- How are the scalar rewards distributed, are there outliers?
- How are the differences between the scalar rewards of the winning and losing responses distributed?
- When the model predicts the wrong response, is the difference between the scalar rewards relatively small?

3.2 Evaluation of the final model

Automatic evaluation metrics

For the final model, two evaluation metrics will be used:

- *BLEU* As a first simple yet naïve analysis, we will use BLEU scores, to assess the quality of the generated answers. BLEU considers matching n-grams between the true answer and the generated answer. Although this is reasonable for machine translation, it is not ideal for our setting. However, it may give a first glimpse of the performance of our model.
- *BERTScore* As a second evaluation metric, we use BERTScore. Instead of considering exact matches, BERTScore aims at assessing the semantic similarity. All the words are embedded using BERT, after which words in the generated sentence and the ground truth sentence are matched based on cosine similarity. We can then use inverse document frequency (idf) weights to assign a different importance to each word. Last, we use the similarities to calculate precision, recall and the F1-score.

Baselines and qualitative analysis

We will use one baseline methods to which we compare our fine-tuned model. Specifically, we will compare the results to the pre-trained GPT-2 model without finetuning. Since we want to analyse whether our fine-tuning and reinforcement learning are effective, it is logical to compare the results to the base pre-trained GPT-2 model.

Furthermore, we will do a qualitative analysis of the answers. We will consider a sample of 30 question and answer pairs, where we focus on obtaining a diverse sample of questions. We then manually evaluate the answers based on the following five criteria:

- *Correctness.* We consider how accurate the generated answer is. More specifically, we look for factual mistakes or flawed lines of reasoning.
- *Coherence.* We analyse whether different parts of the answer logically follow each other. Can the reader easily understand what information the answer conveys?
- *Relevance.* Does the answer provide relevant information? What proportion of the answer actually answers the question at hand?
- *Completeness.* We consider whether the provided answer is complete. In other words, does the generated response answer the question fully?

References

[1] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. July 2019. DOI: 10.48550/arXiv.1907.11692. URL: http://arxiv.org/abs/1907.11692 (visited on May 12, 2023).

[2] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: 2019. URL: https://www.semanticscholar.org/paper/Language-Models-are-Unsupervised-Multitask-Learners-Radford-Wu/9405cc0d6169988371b2755e573cc28650d14dfe (visited on May 12, 2023).