Intelligent Agents (TME286)
Assignment 2: Large language models and chatbots

Problem 2.1

Evaluating state-of-the-art LLMs (20p, mandatory)

In this problem, you will evaluate the problem-solving skills of a few state-of-the-art LLMs, with various prompting strategies. The procedure for accessing and using those models (for making batch runs over multiple question-answer pairs) in a Jupyter notebook can be found in Appendix B.3.2 in the compendium. As can be seen there you can either (i) run the LLMs in Colab (slow, but easy), (ii) download the LLMs and run them on your own computer (requires some manual work, and you must also have Python installed), or (iii) run them remotely via `openrouter.ai` (requires an API key, which you can obtain for free at that site). You can obtain a list of all available models (after first installing `gpt4all` as described in Appendix B.3.2) by entering the command `!llm models list`

Note: If you choose to run the models via Colab, make sure to print the relevant output to screen for each question-answer pair, so that you can extract it in case you are disconnected from Colab, which is very likely to happen, since the runs will take a quite a while to complete. If so, you can extract the answers to the processed questions and then start over from that point, and so on.

On the page for Assignment 2, you will find a very small subset (`TestSet.txt`) of the *Massive multitask language understanding* (MMLU) data set, which has been frequently used as a benchmark data set when evaluating LLMs. The full MMLU data set contains multiple-choice questions over a wide range of topics. Here, we will only consider a subset with questions about astronomy. Note that you must preprocess the data a bit, so that the various options are assigned a letter (A, B, C, or D). In the raw (tab-separated) data file (`TestSet.txt`) each row contains (i) a question, (ii) the four possible answers (which should be assigned the letters A, B, C, and D, respectively), and (iii) the correct answer. See also the file `ProcessingExample.txt` on Canvas.

You should run through the entire data set using at least two different LLMs (from the choices available at `gpt4all`), using two different prompting strategies, namely (i) zero-shot prompting (just asking the question as it is) and (ii) few-shot prompting, in which case you need to augment the data (in a separate file) with one or a few examples where you also give the correct answer (see also the file `AdditionalDataForFewShotPrompting.txt`).

An illustration of few-shot prompting can be found in the original MMLU paper, available at `https://arxiv.org/pdf/2009.03300`, which you should read (in its entirety) before starting with this problem; see especially Fig. 1a and the last paragraph of section 4.1 in that paper. You can also view the leaderboard for MMLU at `https://paperswithcode.com/sota/multi-task-language-understanding-on-mmlu`.

Run through the entire data set in the two ways described above, and collect the answers given by the LLM under consideration. Then compute the accuracy (for each LLM and for each prompting style) over the 50 questions in the data set.

Note: You may wish to edit the code in Appendix B.3.2 a bit, so that it can read a question that spans multiple lines (of text). Alternatively, you can put the entire question (and, where applicable, any other parts of the prompt) on a single line of text. The correct answer should appear on a separate row. Furthermore, it should be noted that one cannot fully rely on the LLMs to produce an output in the required format, i.e., a single letter. Thus, the output from the LLMs must be post-processed manually. You may wish to experiment a bit with the exact

formulation of the few-shot prompts, in order to get the desired behavior from the LLM. You can for example tell it that you will first give it (say) two examples of similar questions, and then a question that the LLM should answer. You may also instruct it to answer with a single letter.

What to hand in You should hand in a Jupyter notebook, which should contain all the required code for running through the entire data set, with each of the two prompting styles. You should also hand in the entire augmented data set for the few-shot prompting case (as a text file). In your report, write a section (`Problem 2.1`), where you describe which models you have used and also give some examples of the prompts (especially for the few-shot prompting case). Moreover, include a table with the accuracy (for each LLM and for each prompting style) and provide a comparative analysis (beyond just noting the results) of how the two prompting styles perform.

Evaluation For this problem, the Jupyter notebook and the (few-shot prompting) data file are worth 12p, and your results and analysis are worth 8p. If you must resubmit the problem, a maximum of 14p will be given.

	LLM 1		LLM 2	
	zero shot	few shot	zero shot	few shot
Accuracy				

Problem 2.2

Factual errors generated by LLMs (10p, mandatory)

In many cases, when answering a question, LLM-based chatbots embark on long rants containing incorrect, made-up statements. Such factual errors (often referred to as *hallucinations*, which is not a suitable term, but will be used below anyway, since it is the de-facto standard term) are a big and persistent problem in LLM-based chatbots. Even though much effort has been devoted to reducing the amount of hallucination in chatbots, the problem persists. The main problem with hallucinations is not that the chatbot makes occasional factual errors (so do humans) or that it does not know everything (neither does any human). The problem, instead, is that the chatbots are very good at producing plausible-sounding text, even in cases where some, or all, of the supposed facts are completely fabricated, often making it difficult for a human to tell fact from fiction. Moreover, in many cases, the chatbots seem unaware of their own limitations, happily producing a completely bogus answer instead of simply saying that they do not know. There are various ways to reduce the impact of hallucinations. For example, by careful prompting, a chatbot can sometimes be made to answer *I don't know* in cases where it would otherwise produce incorrect output. Moreover, retrieval-augmented generation may also help (see also Chapter 5 in the compendium).

In this problem you will investigate hallucinations in LLM-based chatbots. Your goal here will be to formulate questions such that either Gemini (`https://gemini.google.com/app`) or CoPilot (`https://copilot.microsoft.com/` (or both) hallucinate. In order to use Gemini, you must sign in at Google (you already have an account at this stage, for using Colab).

Here, one must bear in mind, again, that the answers are generated probabilistically. In some cases, the chatbot may give a perfectly correct answer, and in other cases (involving the same question) it might provide an incorrect answer. Moreover, new versions of chatbots (or, rather, the underlying LLMs) appear from time to time, meaning that any specific question that previously elicited hallucination may no longer do so.

However, it is generally not *very* difficult to make a chatbot hallucinate. Your task is to formulate at least 10 questions that lead to hallucinations (at least once) in either Gemini or Copilot, or both. Thus, you must formulate (at least) 10 questions, then run them a few times (every time with a fresh session of the chatbot) with each of the two chatbots, and log all the results. Save the results in a text file, for later use in your report.

You must formulate your own questions but, to help you, here are some examples that, at least occasionally lead to hallucinations in one (or both) chatbots. Example 1: *how many U.S. states have a name whose third letter is s?*, Example 2: *how many words in this sentence have exactly four letters?*, Example 3: *Susan has three brothers. Each brother has two sisters. How many sisters does Susan have?*, Example 4: *A man wants to cross a river with a goat and a cabbage. They all fit in a boat that is available at the river bank. How should they go about crossing the river?*

Note, again, that those questions (and other) may not *always* lead to hallucinations, but it is usually easy to find other questions that do.

What to hand in You should write a section (called `Problem 2.2`) in your report, where you first describe the problem of hallucinations. Start by writing your own text, then ask one of the chatbots to (briefly) describe what hallucinations are (in connection with chatbots), and include

the response as well, in your report. Next, include all your interactions with the chatbots, in neatly formatted text format (*not* screenshots), along with some analysis and your conclusions. *For every interaction, make sure to include both the correct (ground truth) response and the chatbot's response.*

Evaluation In this problem, the report is worth 10p. If you need to resubmit the problem, a maximum of 6p will be given.

Problem 2.3
Text classification with BERT (15p, voluntary)

In this problem you will use BERT (on Google's Colab) for classification of movie reviews, available at `https://ai.stanford.edu/~amaas/data/sentiment/`. You can find a full Jupyter notebook, containing all the required steps at `https://colab.research.google.com/github/tensorflow/text/blob/master/docs/tutorials/classify_text_with_bert.ipynb`

Classification with BERT

Your task will be to run through this notebook, selecting a suitable version of BERT for the analysis. However, you should also try to understand what the code does, and also make a simple benchmark classifier for comparison. Start by running through the notebook on Colab, which will train BERT using 20,000 reviews from the training set (keeping the remaining 5,000 for validation). There is also a test set with another 25,000 reviews. In the process of running through the notebook, answer the following questions:

(Q1) What is the average length (number of tokens) of the movie reviews in the test set (25,000 reviews), after BERT's tokenization? (you will need to add some code in the Jupyter notebook)

(Q2) What is the structure of the BERT version that you choose for your analysis? Follow the links under *Loading models from TensorFlow Hub*, read the corresponding scientific paper, and describe the model in as much detail as you can. Include at least one figure, with a clearly written figure caption (that you should write yourself).

(Q3) Describe, in detail, the output of BERT's preprocessing, i.e. the contents of the vectors `input_word_ids`, `input_mask`, and `input_type_ids`. In addition to the general description, provide a specific example, using a sentence of your choice.

(Q4) What does the `pooled_output` (under *Define your model*) do?

(Q5) How do the added layers (for classification) look (i.e. after the pooled output)? Draw a figure and include in your report. Also, explain clearly what the *dropout* layer does.

(Q6) Describe the chosen optimization method (AdamW per default).

Benchmark: Perceptron classifier

Next, merge the set of movie reviews into a training set, a validation set, and a test set, with class labels. That is, from the 20,000 separate files in the training set, generate *one* training set file, in the same format as was used in Assignment 1.2, where every row contains (i) the text and (after a tab character) (ii) the class label, i.e., 0 for negative reviews and 1 for positive ones. Do the same for the 5,000 validation reviews (thus generating a validation set) and the 25,000 test reviews (thus generating a test set). Then train your perceptron classifier from Assignment 1.2 on the training set, using the validation set to determine when to stop (i.e.,

holdout validation, as usual) and, finally, evaluate the best perceptron classifier over the test set, computing accuracy, precision, recall, and F1 score.

What to hand in Write a section (`Problem 2.3`) in your report, where you give clear and thorough answers to the six questions above (Q1) - (Q6). Do not copy-paste from explanations found online: Your answers must clearly indicate that you know what you are writing! Then, make a table showing the accuracy, precision, recall, and F1 scores over the test set (25,000 reviews) for (a) BERT and (b) the perceptron classifier. Thus, this table should contain two rows (BERT-test, perceptron-test). You should also provide some comments regarding the performance (difference) between the two models - the simple perceptron classifier and the very complex BERT model.

You should also hand in the entire Jupiter notebook for BERT (File - Download ipynb in Colab). Note that you do *not* need to hand in the C# code for the perceptron classifier, provided that you do not change it from Assignment 1.2 (you should not change it).

Note: You must hand in results both for BERT and for the perceptron classifier in order for the problem to be corrected. Partial solutions will not be considered!

Note Do not include screenshots in the report.

Evaluation For this problem, the notebook (for the BERT classifier) is worth 7p, the perceptron results 3p and the report 5p. Resubmissions are not allowed.

Problem 2.4
Chatbot based on $n-$grams (15p, voluntary)

In this problem you will build a chatbot based on $n-$grams, using Jelinek-Mercer interpolation to handle missing data. You must generate your own C# solution (see, for example, Tutorial 1 on the course web page) but you may use (and modify) the NLP C# library included in the C# source code for Assignment 1. Apart from that, you may only use C# libraries that are included by default in Visual Studio (e.g., `Math`).

To build your chatbot, you should use the (preprocessed) Penn Treebank data sets, consisting of a training set (`ptb.train.txt`), a validation set (`ptb.val.txt`), and a test set (`ptb.test.txt`). These data sets have a closed vocabulary of 10,000 tokens (plus the beginning-of-sentence (`BOS`) token, which is never predicted), where rare words are represented by an unknown (`unk`) token, all numbers are represented by the token `N`, and every sentence ends with an end-of-sentence (`EOS`) token. Thus, there is never any problem with out-of-vocabulary (OOV) tokens here: All tokens that appear in the validation and test sets also appear in the training set. Incidentally, these data sets were used by Mikolov in the development of the first RNN-based chatbot, and they have also been used by many other authors as a benchmark set for comparing chatbot performance based on the perplexity measure.

Here, you should implement a model that uses Jelinek-Mercer smoothing, where the probability of generating token t_k is given by

$$P_{\text{JM}}(t_k|t_{k-2},t_{k-1}) = \alpha P_3(t_k|t_{k-2},t_{k-1}) + \beta P_2(t_k|t_{k-1}) + (1-\alpha-\beta)P_1(t_k), \tag{1}$$

where P_3, P_2, and P_1 denote the (maximum likelihood estimates of the) trigram, bigram, and unigram probabilities, respectively (see Chapter 3 in the compendium), and where α and β are two tunable parameters. The chatbot should generate sentences one by one, applying Eq. (1) repeatedly until an end-of-sentence (`EOS`) token is predicted, at which point the sentence is complete and the chatbot stops. Every sentence starts with the `BOS` token (t_0) which is inserted deterministically, i.e., without using the model in Eq. (1). The first actual token (t_1) in the sentence is then predicted using

$$P_{JM,1}(t_1|t_0) = \gamma P_2(t_1|t_0) + (1-\gamma)P_1(t_0), \tag{2}$$

where γ is another tunable parameter and where $t_0 =$ `BOS`. For all subsequent tokens, Eq. (1) is used.

When you compute P_3, P_2, and P_1, use only the training set. Note that the training sentences should be processed one by one, so that the last predicted token (for the sentence in question) is the `EOS` token. Then process the next sentence, starting with the prediction of the token that follows immediately after the `BOS` token. In other words, do not cross sentence boundaries. For example, $P($`BOS`$|$`EOS`$)$ need not be computed, since `BOS` is never predicted (instead it is inserted deterministically, as described above).

Once you have built your model with the training set, write code for computing the perplexity (PP, see Section 3.6 in the compendium) over any data set. Start by using the default values of $\alpha = 0.25$, $\beta = 0.5$, and $\gamma = 0.5$, and compute the perplexity over the training, validation, and test sets. Then, use trial-and-error to set the three parameters such that the perplexity over the *validation* set becomes as small as possible. Finally, using those parameters, compute the perplexity over the test set.

The chatbot is now complete. Next, you should use it in two different ways: (i) running it as it is, generating sentences (until the `EOS` token is generated) using Eq. (1) and (ii) running it with the following modification: At any step k of a sentence, compute $P_{\text{JM}}(t_k|t_{k-2}, t_{k-1})$ for all possible (10,000) tokens t_k in the vocabulary. Then, extract the probabilities for the top ten tokens (i.e., the ten tokens with the highest probabilities). Rescale the probabilities of those ten tokens so that they sum to 1 (by dividing each original probability by the sum over the ten original probabilities), and then use *that* reduced, ten-token distribution to generate the token t_k (meaning that, here, the chatbot has only ten different tokens to choose from, at any given step). This running mode is similar to the *low-temperature* (more precise) mode of LLMs.

What to hand in You should hand in the full C# code for your chatbot, which should include code for computing and outputting the perplexities over the three sets, as well as the possibility of running it (for generating sentences) in either of the two modes (i) and (ii) described above. In your report, write a section (`Problem 2.4`) where you briefly describe your implementation and also provide a brief manual for how to run your program (both for computing perplexity and for generating sentences). *Failure to provide a clear description of this kind will result in a point loss.* Also, in the report, include the perplexities computed with the original parameter settings (see above) over the three sets, as well as the final parameter values (obtained via trial-and-error) and the corresponding perplexity values over the validation and test sets. Finally, provide ten examples of sentences for each of the two running modes (i) and (ii) (see above), as well as a brief comparative analysis of the sentences generated in the two modes.

Evaluation For this problem, the C# implementation is worth 10p and the report 5p. Resubmissions are not allowed.