Intelligent Agents (TME286)
Assignment 2: Large language models, chatbots, and task-oriented agents

Problem 2.1 DNN-based chatbot (10p, mandatory)

In this problem you will run through a Jupyter notebook to generate a seq2seq RNN-based chatbot (with GRU units). The script can be found at `https://colab.research.google.com/github/pytorch/tutorials/blob/gh-pages/_downloads/chatbot_tutorial.ipynb`

To train the chatbot, we will use the Cornell Movie Dialogue Corpus, which you can find here `https://www.kaggle.com/datasets/rajathmc/cornell-moviedialog-corpus` The Python implementation used in the notebook requires that the fields should be separated by the character string `+++$+++` , as is the case in the data set obtained from the link above. From this data set, you need to download the files `movie_lines.txt` and `movie_conversations.txt` and place them in a folder called `data/cornell movie-dialogs corpus/` in your Colab account. To do that, click on the folder icon on the left side of the Colab page. Then you can add the necessary folders and (for example) drag-and-drop the two text files. Once you have downloaded and placed the data sets in the correct folder (under Colab), run through the entire Python notebook to generate the chatbot (note that you need to uncomment one line towards the end of the notebook to activate the chatbot).

The line `mask = torch.ByteTensor(mask)` in the function `outputVar`(...) should be changed to `mask = torch.tensor(mask, dtype=torch.bool)`. Without this change, the training will crash, because there is a type change made in a more recent version of Pytorch. The chatbot has an annoying feature, namely that it says "Error: Encountered unknown word" if any word in the input sentence is not in the vocabulary. Please change the Python code so that it instead just ignores such words in the input sentence.

Then download some additional dialogue data (here referred to as the *test dialogues*), for example, a movie script (newer than the cornell data set, so that there is not exact overlap). From that script, extract 10 dialogue turns involving (only) two people talking to each other (where a turn consists of two statements One from Speaker A directly followed by one from Speaker B. Each statement should contain at least four words). Note that (for the purpose of the BLEU score calculation below), the dialogues must only contain words that appear in the vocabulary of the training set (which you can extract from the code in the notebook).

Once you have generated your chatbot, feed it with the Speaker A statements (see above), and store the chatbot's output for each such input statement (corresponding to Speaker B, in the data set, even though the chatbot will of course normally give a different output than the original statement from Speaker B).

Next, compute the BLEU scores (see `https://pytorch.org/text/stable/data_metrics.html`) for each of the chatbot output statements, using the statements from Speaker B as the reference. You may change the n-gram size (see previous link), for example if your reference statements are mostly very short. Then make your own evaluation of each of the chatbot output statements, using the SSA measure. Next, estimate to what degree the BLEU scores correlate with your own judgment of the chatbot's responses. Finally, compute the total BLEU score (by using all the reference input-output pairs as the input to the score function) and include it in your report.

What to hand in You should save the entire notebook (from Colab), showing that you have indeed run through all the steps. Moreover, in the report, add a section (`Problem 2.1`) where you list all the test dialogues, along with the output (given in response to each statement by Speaker A) from your chatbot, as well as the BLEU scores and SSA estimates, and your (informal) analysis of how the SSA scores correlate with the BLEU scores. Also, hand in your test dialogues as a simple text file.

Evaluation For this problem the Python notebook is worth 2p and the report 8p. If you need to resubmit the problem, a maximum of 6p will be given.

Problem 2.2 Prompting in LLMs (10p, mandatory)

In this problem, your task is to investigate how the performance of LLM-based chatbots, namely Bard (`https://bard.google.com/chat`) and CoPilot (`https://www.bing.com/search?q=Bing+AI&showconv=1`), varies with the quality of the prompt. In this task, you will have quite a bit of freedom to formulate the details of the problem yourself, but note that you then also have the *responsibility* of making a thorough investigation and writing a good report. You must use at least the two chatbots mentioned above, but if you wish you may *also* consider other chatbots for comparison, though it is not required.

Start by reading the references (available on the Canvas page for Assignment 2) on prompting. Next, formulate a set of at least 10 questions for which a true, unambiguous answer exists, but where also some form of reasoning (computation) is required for arriving at the answer, rather than just reading it directly in a text. As an example, just to try different prompting methods, you may first consider the question about Nobel prizes that was used as an illustration in Lecture 7 (where counting was required to arrive at the answer).

Then you should formulate your own questions. You may wish to consider counting, as in the example just mentioned (but applied to a different situation), but you can also consider some other elementary mathematical calculation. As another option, your question may involve ranking (of different options), comparison (of two or more things, for example, size, number, speed, ...), sorting and identifying a particular element (for example, the second-largest or third-fastest, or something like that), and so on. The questions should be sufficiently difficult so that the chatbots have trouble (or fail) to answer them with zero-shot prompting.

Next pose the questions to the chatbots using different prompting styles: (i) zero-shot prompting (just asking the question); (ii) few-shot prompting, where you provide at least one example of how to solve a problem of the kind you are considering (however, the example must not be exactly the *same* as the question that you then ask the chatbot to answer!); (iii) chain-of-thought prompting; (iv) (voluntary) any other, more advanced, prompting methods that you may wish to use. See also the relevant references, making sure that your prompts really *are* of the required kind.

Notes

(1) The answers obtained from LLM-based chatbots are generated stochastically, so any result you get may not be reproducible. Therefore, make sure to save the logs from your interactions with the chatbots. For each session, log (in text format, *not* a screenshot) the entire interaction, i.e., the question (prompt) and the answer.

(2) Use a separate session for each prompting style *and* for each question (bearing in mind that the chatbots can maintain long-range context. Here, each question (for each prompt style) should be applied to a fresh instance of the chatbot under consideration.

What to hand in In your report, write a section (called `Problem 2.2`) where you describe the different prompting styles used (explaining them to a reader who is not familiar with prompting). Write your *own* text here. Then, ask a chatbot (e.g., CoPilot) the question *Please list and describe three different prompting styles used when interacting with chatbots*,

and include the chatbot's answer (with any comments that you may have - you may also ask follow-up questions) in your report. Next, include all the logs (i.e., at least $2 \times 10 \times 3$) from your chatbot interactions. Do *not* include screenshots. Format the logs neatly, with clear separations between different sessions, and so on. Finally, based on the answers from the chatbot, estimate the (average) performance (fraction of correct answers) for each of the prompting methods. Place these results in a table in your report, and then discuss your findings.

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 LLM hallucinations (10p, mandatory)

In many cases, when answering a question, LLM-based chatbots embark on long rants containing incorrect, made-up statements that are passed as facts by the chatbot. Hallucinations are a big and persistent problem in LLM-based chatbots that, moreover, illustrate the limitations of statistical correlation. 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 Bard (`https://bard.google.com/chat`) or CoPilot (`https://www.bing.com/search?q=Bing+AI&showconv=1` (or both) hallucinate. 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 5 questions that lead to hallucinations (at least once) in either Bard or Copilot, or both. Thus, you must formulate (at least) 5 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 leads to hallucinations in one (or both) chatbots: *what is the record for walking across the English channel?*, *how many words in this sentence have exactly four letters?*, *which is the eighth largest planet in the solar system?*. 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 2.3`) 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.

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

Problem 2.4
Linguistic variability (10p, voluntary)

In this problem, we will investigate linguistic variability in a discussion related to a specific topic, where factual correctness is important, i.e., a task-oriented agent. That is, we will study how much variability the task-oriented agent may have to cope with, primarily as regards its input processing (i.e., matching the user's input to a specific template, for example). While we will not implement the task-oriented agent *per se* in this problem, the aim is to generate a data set that can be used for doing so by combining manual work with the use of LLM-based chatbots.

We will consider a rather simple (and, perhaps, not that critical) task, namely a situation where a medical professional, for example, a doctor, nurse, or counsellor, wants to define a daily schedule (with various activities) for a patient (at a hospital or a nursing home) in interaction with a task-oriented agent (that will then be used for interactions with the patient). The schedule should include the following activities. (i) wake-up, (ii) shower, (iii) breakfast, (iv) light exercise, (v) morning nap, (vi) lunch, (vii) a walk (with a nurse), (viii) afternoon nap, (ix) dinner, and (x) bedtime. In the final schedule, the activities should be in that order, but they need not necessarily be specified in the given order when the schedule is made. The schedule should be built in a series of statements that the user provides and where the (yet-to-be-defined) task-oriented agent responds. A typical (part of) a dialogue can go like this:

User (U): *I want to set up a daily schedule for Mrs. Smith in Room 102*
Agent (A): *OK, understood, schedule for Mrs. Smith*
U: *She should wake up at 08.00*
A: *OK*
U: *Then a shower at a quarter past eight*
A: *Understood*
. . .
U: *That's it*
A: *OK, I have defined the entire schedule*

Of course, the statements above are just *examples*. For instance, in the first sentence, the user could say *I want to make a daily plan for Mr. Green* or *I'd like to define a schedule for Mrs. Brown* or *Can you help me make a daily routine for Mr. White?*, and so on. Similarly, the agent may of course vary its answers.

Your task will be to make a varied data set (not an agent!) of the kind described above, consisting of several dialogues, such that each dialogue defines an entire schedule with the 10 events listed above. Note that (i) the dialogue should only involve the definition of the schedule, not other activities or other topics. Of course, ideally, the resulting agent should be able to converse with its users on a variety of different topics, but here we will only consider the definition of a schedule of the kind described above; (ii) for any given statement, the user should describe *at most* two activities, thus either one activity (as in the example above) or two activities as in *he should have lunch at 12.00 followed by a walk with Nurse Cooper at 13.00*; (iii) the user may, from time to time, change her or his mind, with statements such as *no, I meant lunch at 12.30, not 12* (followed by a corrective response from the agent, e.g. *understood.*

Lunch at 12.30 instead and so on.

Given these restrictions, you should first make a set of complete, manually defined dialogues (at least 5), where you try to vary the words and phrases used. Next, you should use a set of (at least two, preferably more) chatbots, for example ChatGPT, Bard, Bing Chat, and any other chatbot that you might wish to access, to do two things:

Paraphrasing Ask the chatbot to paraphrase the user sentences (the ones marked with U in the example above), in each of your dialogues. That is, you should provide a suitable prompt as well as the entire sentence and then ask the chatbot to generate a number of paraphrased versions, say 3-5 (for each user sentence, and for each chatbot tried). Some of the results will be correct, but some sentences will (probably) be slightly different, semantically. Make a note of which sentences are exactly correct (i.e., a true paraphrasing) and which sentences have a slightly (or completely) different meaning (you will need it in the report; see below).

Complete dialogues Next, write a suitable prompt (and save it as a text file, not a screenshot: It should be included in the report) that makes the chatbot(s) return *the entire schedule*. That is, you must give it instructions roughly as above, but do *not* just copy the entire text above; write a clear prompt, so that the agent can build the schedule and also fulfil conditions (i)-(iii) above.

What to hand in Make a section (`Problem 2.4`) in your report, where you (i) write down *all* your manually defined dialogues, *neatly* written (in their entirety, with nothing left out); (ii) write down the prompt(s) you defined for generating paraphrased sentences; (iii) make a table with two columns where, in the left column, you write down all the sentences in all your manually defined dialogues (yes, there will be quite many sentences), and in the right column you provide the paraphrases (3-5 for each sentence and from each chatbot). Sentences that are correct, i.e., those that are true paraphrases of your sentence (in the left column) should be marked in light green, whereas sentences that are incorrect (semantically different) should be marked in light red; (iv) write down the prompt(s) you defined for generating an entire dialogue (with a chatbot); (v) write down all the complete dialogues (so, at least two, preferably more) generated by the chatbots, and mark also (in red) any sentences that are incorrect, as per the specifications given above.

Evaluation The report is worth 10p. Please make sure to read the paragraph above *very* carefully, and include *all* the five steps described there, with maximum clarity. Resubmissions are not allowed.

Problem 2.5
Text classification with BERT (10p, 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 2.5`) 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 Do not include screenshots in the report.

Evaluation For this problem, the notebook (for the BERT classifier) and the report are together worth 10p. Resubmissions are not allowed.

Problem 2.6
Information-retrieval chatbot (10p, voluntary)

In this problem you will build an information-retrieval (IR) chatbot in C#. You must generate your own C# solution (see, for example, Tutorial 1 on the course web page) but you may use the NLP C# library included in the C# source code for Assignment 1. You may not use other C# libraries (and neither do you need to), except those that are included by default in Visual Studio (e.g. `Math`).

The chatbot should make use of TF-IDF embeddings, as described in Section 5.2 of the compendium. You must generate your own dialogue (pair) corpus using transcripts of spoken dialogue. Links to some suitable data sources can be found under Assignment 2 on the Canvas page, but you may use other sources if you wish.

The processed data should consist of a (large) number (N) of sentence pairs (S_1, S_2), such that sentence S_2 was given in response to sentence S_1. That is, the two sentences should (i) be consecutive in a dialogue; (ii) concern two people who are actually talking to *each other*; and (iii) should not be too long, at most (say) 15-20 words each, often shorter. Thus, for this problem, it is important to extract data with care, in order to make sure that the sentence pairs fulfil the three conditions just mentioned.

Once your program has loaded and tokenized the corpus, it should compute normalized TF-IDF vectors for all sentences S_1 (i.e. for the first sentence in every pair). The length of those vectors is given by the number of distinct tokens (words) in the corpus. You must thus write code for tokenization (and other preprocessing steps that you may wish to use, e.g., spelling correction, case folding, and so on), and TF-IDF computation. You may use your own code from Assignment 1 to the extent that it is possible (for instance, in order to carry out tokenization).

Next, when running the chatbot, for a given input sentence T, compute the cosine similarity between T and the first sentence (S_1) of *every* sentence pair (S_1, S_2) in the corpus, and sort the result (along with the index of the sentence pair in question) in decreasing order of cosine similarity. Then, rather than just selecting the pair with highest degree of similarity as in Eq. (5.6) in the compendium, select a sentence pair randomly from the top five pairs, so that the chatbot may display a bit more variability. From the selected sentence pair, return the sentence S_2 as the output from the chatbot (shown on-screen).

Note: When computing the cosine similarities, just ignore any (rare) words in the test sentence T that do not appear in the corpus. Do *not* modify the corpus once the chatbot has been loaded.

What to hand in You should hand in the full C# code for your chatbot, as well as the (processed) data file in the form of a single text file containing the corpus, i.e. all sentence pairs (one pair per row, with the two sentences separated by a tab character). Moreover, you should write a section (`Problem 2.6`) in your report where you briefly describe your implementation and also provide a brief manual for how to run your program, i.e. the steps required to (1) load the raw data file(s), (2) preprocessing the raw data to form the set of sentence pairs, (3) computing the TF-IDF vectors, and (4) running the chatbot. *Failure to provide a clear description of this kind, or omitting the data files, will result in a point loss.* Finally, in the report, include also a set of sample dialogues between the chatbot and a user (you).

Evaluation For this problem, the implementation is worth 7p and the report 3p. In particular, we will assess how well your corpus fulfils the conditions (i)-(iii) above. Resubmissions are not allowed.