# Assignment 5

NLP SS23

July 1, 2023

**Organizational:** This is a **group assignment**, i.e., submit your solutions in groups of **three** members. If you cannot find a third member, notify `sedigheh.eslami@hpi.de`. Since the number of students is not divisible by three, in rare exceptions we will allow groups of two. In case you have trouble finding any group mates, send an E-Mail to `sedigheh.eslami@hpi.de`, and we will try to find a group for you.

This assignment is **mandatory for participation in the exam**. You are required to obtain at least 50% of the total points in this assignment to become eligible for participating in the final exam.

**Note:** This assignment has a mini-project style and will require more time investment than previous assignments. Plan accordingly and do not start working on this only a week before the deadline!

**Due date: 22.07.2023, 9:00 a.m. (CEST)**. If we receive many requests, there is a possibility for extension!

## 1 Task 1: Masked Language Modeling

### 1.1 Background

Masked language modeling (MLM) is the task of predicting masked tokens in a text sequence. MLM as a self-supervised pre-training objective became popular, especially with the emergence of BERT models. During pre-training with MLM, given a big corpus, $x\%$ of the tokens are first randomly selected and replaced with the special token `<mask>`. The masking step is either done statically, e.g., in BERT, or dynamically during pre-training runtime, e.g., in RoBERTa. In BERT-based models, usually $x = 15\%$. After masking, a deep neural network (DNN) gets optimized for filling the masked tokens by selecting the tokens from the vocabulary that maximizes the probability of the text sequence. As you might have guessed, this DNN typically includes a number of Transformers with self-attention layers. The loss function for optimization is a cross-entropy between the sequence with the ground-truth tokens and the sequence with the predicted masked tokens.

### 1.2 Task (40 points)

In this task, you will be fine-tuning a pre-trained `distilroberta-base` model for the task of masked language modeling using the `ag_news` dataset and HuggingFace library. Fine-tuning pre-trained models for language modeling is often helpful when you want to adapt the language model into a specific domain, e.g., clinical texts or even new languages.

Unlike previous assignments, you are required to use all of the `ag_news` dataset for training this time. Therefore, the training takes multiple hours, at least 4 based on our experience with the Google Colab Notebook. So, you might want to start with this assignment sooner than later! :)

1. **(2 points)** Load the train and test splits from `ag_news`. Randomly select 10% of the training set as validation.

2. (3 points) Prepare your data pipeline including the text pre-processing. Set the maximum sequence length to 256. Do you need to replace the padding token `pad` with the end of the sequence `eos` token? Why or why not?

3. **(2 points)** Apply your pre-processing function to all data in all splits using the HuggingFace Dataset `map` function. Are there any columns in the `ag_news` dataset that is not required for the MLM task? If yes, remove it by defining it in the `remove_columns` in the `map` function.

4. **(4 points)** Define the suitable data collator for MLM with a masking probability of **0.1**.

5. **(2 points)** Load the `distilroberta-base` model with **pre-trained** weights.

6. **(2 points)** If you print the model, you notice that it has 6 layers in its encoder module. Each of these layers has an `attention, intermediate` and `output` layer. Update the dropout probability of the `output` layer in each of the 6 encoder layers (from 0.1) to 0.15.

   Hint: You can do this by either using `torch.nn.Dropout` or updating the pre-trained HuggingFace `config` for your model.

7. **(5 points)** Define the `TrainingArguments` with learning rate scheduler and weight decay.

8. **(5 points)** Define the `Trainer` with your updated model, training arguments, train and validation splits, and the data collator you defined in step 4.

9. **(5 points)** Train the model and try to tune the hyper-parameters, e.g., batch size, number of epochs, weight decay and learning rate. You do not have to reach a specific performance goal for this task. It is rather about building an understanding of how to perform masked language modeling. Although, a validation loss of more than 0.3 after epoch 3 means that things are probably not working as intended.

   What is the best validation loss you achieved after training? Describe your setup including final choices of hyper-parameters, optimizer, etc.

10. **(5 points)** Select the best model from step 7 where the minimum validation loss is achieved. Calculate the perplexity on validation and test splits and report them separately. Do you think there is a relationship between perplexity and cross-entropy?

    Hint: HuggingFace's tutorial on perplexity can help you! :)

11. **(5 points)** As an explicit inference, use your model to predict the `<mask>` token in the following text (taken from `ag_news`) and report the top 5 probable tokens predicted. Do you think these predictions make sense? Why or why not?

    ```
    text = "E-mail scam targets police chief Wiltshire Police warns about <mask> after
    its fraud squad chief was targeted."
    ```

**Submission:** For this task, each group is required to submit a ZIP file including:

- Your **runnable** Python code for this task with instructions for environment setup in a `README.md`. The `README.md` should also include the training command you used to obtain the best model.
- A PDF file which states your group members and answers to questions 2, 9, 10, and 11.

The name of the ZIP file must include the first name and last name of the group members following the pattern:

**mlm_{firstName1_secondName1+firstName2_secondName2+firstName3_secondName3}**.zip

For example, if your name is Hasso Plattner and you are submitting together with Peter Lustig and Albert Einstein, the file should be named:

**mlm_hasso_plattner+peter_lustig+albert_einstein**.zip

We have already provided you with a template Python script with a few hints.

### 1.3 Bonus Task (no points, but fame and glory)

What we just did is called *domain-adaptive pre-training* [2] by NLP researchers. We have *adapted* the model that was pre-trained on the general domain (general web-crawled data) to our target domain (news articles). Domain-adaptive pre-training can help a lot when the domain of an NLP task is not very similar to the domain of the pre-trained model's training corpus. Remember that we did news article classification on the AGNews dataset last assignment! Does our domain-adaptive pre-training improve performance?

1. Compare performance on news article classification of "off-the-shelf" pre-trained `distilroberta-base` that you used in assignment 4 vs. your domain-adapted model.
2. Which one performs better?
3. Are there any interesting findings?

**Submission:** Include your report and any code snippets in separate files in the ZIP file you submit for Task 1.2.

## 2 Task 2: Generative Open-book Question Answering

### 2.1 Background

**Question Answering.** In question answering (QA), our goal is to train models that can receive a question as input and provide the answer as output. In extractive question answering, in addition to the question, the model also receives an input text, namely context, and the objective is to *extract* the span of the subtext in the context that provides the answer to the question. This can be, for instance, done by a classification approach, where the goal is to classify each token in the context as either `begin_of_answer, end_of_answer` or `none`.

On the other hand, one can also solve question answering as a generative task, i.e., given the question and the context, generate the sequence of the answer text by predicting the most probable sequence of tokens. In this setting, the goal is to generate a correct answer to the question, regardless of whether the answer is exactly extracted from the context or not. We refer to this setting as the generative open-book question answering.

**T5 Model.** T5 [3] is yet another transformer model that enables us to perform many downstream tasks, e.g., question answering, text summarization and even translation. T5 has an encoder-decoder architecture that has been pre-trained to generally learn many sorts of text-to-text generation. For example, in the language translation setting from English to German, this text-to-text generation task becomes EnglishText-to-GermanText generation. In the summarization task, it becomes LongText-to-SummaryText generation. Similarly, in the question answering task using a context, it becomes QuestionContextText-to-AnswerText generation. T5 has been pre-trained using a very large amount of data for multiple supervised and unsupervised tasks. As usual, the cross-entropy loss is used to calculate the loss between the ground truth text and the generated text sequence. For more information, have look at this blog.

### 2.2 Task (60 points)

In this task, you will be experiencing a mini-journey of an NLP researcher working on a state-of-the-art generative generative question answering task. You will be using the state-of-the-art benchmark dataset QASPER[1] and employing T5 to solve the question answering task in an open-book scenario.

**Note:** This problem is an ongoing state-of-the-art research problem and even many researchers are not able to achieve high prediction performances. Remember that you are **not required** to reach a specific high performance goal for this task. The main goal is for you to experience a small end-to-end NLP research journey in the context of natural language text generation and get familiar with a scientific research scenario! :)

**QASPER:** The motivation for curating the QASPER dataset is that readers of scientific papers often read the paper with certain questions about "what dataset was used for experiments", "how long the training process required", "how much did the results improve", etc. QASPER has been proposed with the purpose of enabling the training of such question answering models. QASPER includes multiple sets of question–answer pairs on various scientific articles which are mostly related to the NLP research.

This dataset provides the full text for each article including the abstract section, introduction, methods, etc. For simplicity and to avoid the handling of long texts, in this task, you will be only working with the **abstracts** from each article in order to perform the QA task.

We suggest that you use the Kaggle notebooks for this task. In the Kaggle notebook, you can use a P100 GPU with 16GB of virtual RAM. This GPU should be enough for this task. We highly encourage you to use the HuggingFace framework for solving this task. Similar to Google Colab, after each `!pip install`, you need to restart the run-time.

Advanced tip: Kaggle also provides the 2xT4 16GB GPU setup, which allows you to use two GPUs with 16GB virtual RAM at the same time. Using DistributedDataParallel training, this will allow you to train (almost) twice as fast.

Your tasks are:

1. Load the QASPER dataset: `dataset = load_dataset("allenai/qasper")`

   The dataset comes with train and test splits. Print the first sample from the train split and familiarize yourself with the structure of this dataset. Notice the nested structure of the data where on the first level, you have the `id, title, abstract, qas` and other attributes. In the second level, e.g., under the `qas` attribute, you see for example `question, answers`. On the third level, e.g., under `answers`, you should be able to see `unanswerable, extractive_spans, free_form_answer` and other attributes.

   You will be mainly working with the `qas` and `abstract` features in this task.

2. **(6 points)** Perform an explanatory data analysis (EDA) on the train split of the dataset. Your EDA should include the distribution of number of questions per article (i.e., how many articles have $x$ number of questions, how many article have $y$ number of questions, etc), the average number of answers per question (notice that in the dataset, a question can have multiple answers associated with it), the number of free-form answers in this split, the number of extractive answers in the training split, the number of unanswerable questions, and the distribution of abstract lengths. You can use assignments 1 and 2 from the beginning of the semester to get inspired.

3. **(4 points)** Repeat step 2 for the test split.

4. **(10 points)** For the open-book QA task, T5 expects the inputs to be in the form of:

   ```
   question: {q} context: {c}
   ```

   For example, for the question "what are the main contributions in this paper?" from a paper with the abstract "This paper is proposing the S55-dummy model in order to solve question answering.", the final input text for T5 should be:

   "question: what the main contributions in this paper? context: This paper is proposing S5-dummy model in order to solve question answering."[1].

   In this task, we will be using the abstract sections of papers as contexts. Notice that we have multiple questions for each paper and therefore, each abstract. So, the same abstract must be used for each of the questions from the same article.
   Furthermore, some questions can have multiple answers associated to them. For simplicity, randomly select just one answer as the ground truth in these cases. Notice that for extractive questions, the answer is provided by the:

[1] These examples are synthetically generated and not real.

`sample['qas']['answers']['answer']['extractive_spans']`

.
In contrast, for the free-form questions, the answer is provided by the:

`sample['qas']['answers']['answer']['free_form_answer']`.

You can treat both cases simply as answers, regardless of extractive or free-form types.

In this step, you will be flattening the hierarchical structure of the dataset so that parallelism of the text pre-processing function on the dataset becomes possible. You can use the following structure for flattening. You can also propose other structures if you want. :)

Perform flattening for both train and test splits.

```
train = {'abstract': train_abstracts, 'question': train_questions,
         'answer': train_answers}
test = {'abstract': test_abstracts, 'question': test_questions,
        'answer': test_answers}

train_dataset = datasets.Dataset.from_dict(train)
test_dataset = datasets.Dataset.from_dict(test)

flattened_dataset = datasets.DatasetDict(
                    {'train': train_dataset,
                     'test': test_dataset})
```

Notice that the lengths of the abstracts collection and questions and answers should be the same, i.e.,:

```
assert len(train_abstracts) == len(train_questions)
       len(train_abstracts == len(train_answers)
```

The same condition applies for the test split. The goal is to have a 1:1:1 relationship between abstract:question:answer for simplicity.

5. **(2 points)** From this step, you should be only working with your flattened dataset. Randomly choose 10% of the train split to be used as the validation set.

6. **(10 points)** Define your pre-processing function. Set the maximum input length (i.e., concatenated question and abstract) to 128 and the maximum target length (i.e., answer sequence) to 32. Do these maximum sequence values result in truncating a lot of your sequences? Hint: Use the EDA on length distribution from the step 2 for answering this.

   Notice that since we are working with an encoder-decoder model here, you need to tokenize, truncate and pad the input sequence, i.e., `question: {question} context: {abstract}`, as well as the output sequence, i.e., answer.

   **Note:** Choice of 128 and 32 for sequence lengths is due to GPU resource limitations. Longer sequences might result in out of memory issues.

7. **(2 points)** Apply the pre-processing function to all of the data using the HuggingFace Dataset `map` function.

8. **(2 points)** Load the `google/t5-efficient-tiny` model with **pre-trained** weights. For the generative QA task, you need to use `T5ForConditionalGeneration`. Different versions of T5 checkpoints with different numbers of parameters are publicly available for you to use, e.g., this link provides all the pre-trained checkpoints provided by Google. Due to the GPU resource limitations on the Kaggle notebook, we encourage you to use the tiny model.

9. **(10 points)** Define your `Seq2SeqTrainingArguments`[2] with learning rate scheduling and weight decay. Notice that this task requires many more calculations as well as GPU memory in comparison to all the tasks you have experienced so far in this course. Therefore, you are required to carefully follow the instructions provided by HuggingFace in this link for efficient training. Otherwise, you might encounter `CUDA Out Of Memory` errors, which means that training your model requires more virtual RAM than you have on your notebook, i.e., 16GB if you're using a single GPU on Kaggle.

   Hint: You might even need to set your batch size to 1. But what can you do to still reach a larger *effective* batch size? The HuggingFace tips should help you with the answer!

10. **(10 points)** Define your `Seq2SeqTrainer`. Choose the best model on the validation set, i.e., the model achieving the best loss value. You do not have to reach a specific performance goal for this task. It is rather about building an understanding of how the text generation procedure happens using T5. However, a validation loss value higher than 2.5 after 3 epochs potentially means things are not working as intended!

    For logging, store the train and validation loss values for each step to your Weights & Biases `https://wandb.ai/site` profile and provide us with these plots. You can simply provide us screenshots of these plots.

    **Optional:** You can use the BLEU score for evaluating the sequence generation performance on the validation set. That means, you can define `compute_metrics` to return BLUE scores using the `evaluate` library from HuggingFace. Notice that BLEU from `evaluate` requires to strings as inputs. But the outcome of your model is a sequence of `token_id`. Therefore, you would need to decode them using your tokenizer and generate the text string first.

    **Note:** If you're using the Kaggle notebook, make sure to check out this tutorial on how to add your wandb API key to the Kaggle environment.

11. **(4 points)** As an example inference, select a single sample from the test split of your flattened dataset and use your model for the question answering task. Notice that the outcome of your model is a sequence of `token_id` and you need to use your tokenzier for decoding these ids, i.e., converting each id to a token using the vocabulary. You are free to choose any sample from the test split.

    Explain your observations. How does your model perform when answering? Does the output make sense? What do you think should be done for improving the prediction?

    **Note:** The goal here is that you start critical thinking about your results and try to analyze what your results mean [3]. Even if you get empty strings generated by your model, start thinking about potential issues, suspect what could be improved, what more does the model need for better predictions, etc. In research, we often times have an iterative process of analysing results and implementing for improving until we achieve reasonable observations! :)

**Submission:** For this task, each group is required to submit a ZIP file including:

- Your **runnable** Python code for this task with instructions for environment setup in a `README.md`. The `README.md` should also include the training command you used to obtain the best model.
- A PDF file which states your group members and an optional short paragraph summarizing your research journey. You can include the challenges you encountered during this task, your observations, your learnings, etc. The PDF should also include answers to questions 2, 3, 6, 10, and 11.

The name of the ZIP file must include the first name and last name of the group members following the pattern:

[2]`Seq2Seq` is usually short for the phrase Sequence-to-Sequence. In theory, any type of task that gets a text sequence as input and generates a text sequence as output can be solved by a Sequence-to-Sequence problem. In our open-book QA task, we are also inputting a text and generating a text and therefore, it is a Seq2Seq task. Using the `Seq2SeqTrainingArguments` from HuggigFance rather than the vanilla `TrainingArguments` gives you nice properties like `predict_with_generate` options.

[3]Similar to an NLP researcher! :)

**qa_{firstName1_secondName1+firstName2_secondName2+firstName3_secondName3}**.zip

For example, if your name is Hasso Plattner and you are submitting together with Peter Lustig and Albert Einstein, the file should be named:

**qa_hasso_plattner+peter_lustig+albert_einstein.zip**

## References

[1] P. Dasigi, K. Lo, I. Beltagy, A. Cohan, N. A. Smith, and M. Gardner. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4599–4610, 2021.

[2] S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith. Don't stop pretraining: Adapt language models to domains and tasks, 2020.

[3] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.