Homework 1: Language models (50 points)

The first homework focuses on the following skills: being able to work with simple formal exercises on language modeling, on understanding and being eable to extract properties and configurations of state-of-the-art language models and, finally, conducting language model training yourself!

Logistics

- submission deadline: May 12th 23:59 German time via Moodle
  - please upload a **SINGLE ZIP FILE named Surname_FirtName_HW1.zip** containing the .ipynb file of the notebook (if you solve it on Colab, you can go to File > download) and the json file.
- please solve and submit the homework **individually**!
- if you use Colab, to speed up the execution of the code on Colab (especially Exercise 3), you can use the available GPU (if Colab resources allow). For that, before executing your code, navigate to Runtime > Change runtime type > GPU > Save.

Exercise 1: Understanding language modeling (12 points)

Please answer the following exercises. Importantly, please reason step by step; i.e., where calculations are required, please provide intermediate steps of how you arrived at your solution. You do not need to write any code, just mathematical solutions.

1. [6pts] Consider the corpus C with the following sentences: $C=¿${"The cat sleeps", "The mouse sings", "The cat sleeps", "A dog sings"}.

   (a) Define the vocabulary V of this corpus (assuming by-word tokenization).

   (b) Pick one of the four sentences in C. Formulate the probability of that sentence in the form of the chain rule. Calculate the probability of each termn in the chain rule, given the corpus.

1. [4pts] We want to train a neural network that takes as input two numbers x_1, x_2, passes them through three hidden linear layers, each with 13 neurons, each followed by the ReLU activation function, and outputs three numbers y_1, y_2, y_3. Write down all weight matrices of this network with their dimensions. (Example: if one weight matrix has the dimensions 3x5, write $M_1 \in R^{3\times5}$)
2. [2pts] Consider the sequence: "Input: Some students trained each language model". Assuming that each word+space/punctuation corresponds to one token, consider the following token probabilities of this sequence under some trained language model: $p=[0.67, 0.91, 0.83, 0.40, 0.29, 0.58, 0.75]$. Compute the average surprisal of this sequence under that language model. [Note: in this class we always assume the base e for log, unless indicated otherwise. This is also usually the case throughout NLP.]

Answer Exercise 1: Understanding language modeling (12 points)

Note: The usage of BOS/EOS is not typical for theses models. Nonetheless both practices are correct.

1. First Possibility

    a. vocabulary V = {'the', 'cat', 'sleeps', 'mouse', 'sings','a', 'dog'}

    b. The cat sleeps: \begin{align*} P_{LM} = &|prod^n_{n=1}P_{LM}(w_i|w_{1:i-1})| | end{align*} $P_{LM}(the,cat,sleeps)=p(the)*p(cat \vee the)*p(sleeps \mid the,cat)$ \ begin{align*} P_{LM}(the,cat,sleeps)=& |frac{3}{12}\frac{2}{3} |frac{2}{2}= |frac{1}{4}\ frac{2}{3}=\frac{2}{12}= \frac{1}{6}= 0.167\ \end{align*} The mouse sings: |begin{align*} P_{LM} = \prod^n_{n=1}P_{LM}(w_i|w_{1:i-1}) \end{align*} $P_{LM}(the,mouse,sings)=p(the)*p(mouse \mid the)*p(sings \mid the,mouse)$ | begin{align*} &P_{LM}(the,mouse,sings)=\frac{3}{12} |frac{1}{3}\frac{1}{2}=\frac{1}{4} | frac{1}{3}\frac{1}{2}= \frac{1}{24}= 0.125\ \end{align*}
    A dog sings: |begin{align*} &P_{LM} = \prod^n_{n=1}P_{LM}(w_i|w_{1:i-1})\ \end{align*} $P_{LM}(a,dog,sings)=p(a)*p(dog \mid a)*p(sings \mid a,dog)$ |begin{align*} &P_{LM}(a, dog,sings)=\frac{1}{12} |frac{1}{1}\frac{1}{1}= \frac{1}{12}= 0.083\ \end{align*}

1. Second Possibility

    1. a. vocabulary V = {'','the', 'cat', 'sleeps', 'mouse', 'sings','a', 'dog'''}

       b. The cat sleeps: \begin{align*} P_{LM} = &|prod^n_{n=1}P_{LM}(w_i|w_{1:i-1})| | end{align*} P_{LM}¿p(|,the,cat, sleeps)$ |begin{align*} P_{LM}(, the,cat,sleeps,)=&\ frac{3}{4} |frac{2}{3}\frac{2}{2} |frac{2}{2}= |frac{6}{12} = |frac{1}{2}= 0.5| |end{align*}
       The mouse sings: \begin{align*} P_{LM} = |prod^n_{n=1}P_{LM}(w_i|w_{1:i-1}) | end{align*} P_{LM}¿p(|,the,mouse, sings,)$ |begin{align*} &P_{LM}(,the,mouse, sings,)=\frac{3}{4} |frac{1}{3}\frac{1}{1} |frac{1}{1}=|frac{3}{12}= |frac{1}{4} = 0.25| | end{align*}
       A dog sings: \begin{align*} &P_{LM} = |prod^n_{n=1}P_{LM}(w_i|w_{1:i-1})| | end{align*} P_{LM}¿p(sings|,a, dog, sings,)$ |begin{align*} &P_{LM}(,a, dog,sings,)=\ frac{1}{4} |frac{1}{1}\frac{1}{1} |frac{1}{1}= |frac{1}{4}= 0.25| |end{align*}

1.1 evaluation

- vocabulary correct = 2 points
- calculation correct = 4 points
- calculation incorrect, but P_{LM} formula correct = 2 points
- calculation and formula incorrect but idea understandable = 1 point

deduction:

- vocabulary with minor mistake = -1 point
- caculation correct for one alternative but formula of the other alterative = -1 point

---

a. input_neurons = 2
hidden_neurons = 13
output_neurons = 3
$M_1 \epsilon R^{2x13}$ input to hidden1
$b_1 \epsilon R^{1x13}$ weights1
$M_2 \epsilon R^{13x13}$ hidden1 to hidden2
$b_2 \epsilon R^{1x13}$ weights2
$M_3 \epsilon R^{13x13}$ hidden2 to hidden3
$b_3 \epsilon R^{1x13}$ weights3
$M_4 \epsilon R^{13x3}$ hidden3 to output
$b_4 \epsilon R^{1x13}$ weights4

1.2 evaluation

- Matrizes M_i from layer to layer correct = 3 points
- comment or definition of bias weights b_i = 1 points
- Matrizes incorrect, but correct right amount of matrizes = 2 points
- Amount of matrizes M_i incorrect, but matrizes layer correct = 2 points (if additionally $b_i += 1 point)

---

1. Sentence = 'Input: Some students trained each language model'

   $p = [0.67, 0.91, 0.83, 0.40, 0.29, 0.58, 0.75]$ \begin{align*} P_{LM} = &| prod^n_{n=1}P_{LM}(w_i|w_{1:i-1})= 0.67*0.91*0.83*0.40*0.29*0.58*0.75=0.0255| | end{align*}*

   $\text{Avg-Surprisal}_{LM}(w_{1:n}) = -\frac{1}{n} \log P_{LM}(w_{1:n})$

   $¿ -\frac{1}{7} \log(0.2553533346) = -\frac{1}{7} * -0.59 = 0.523$

1.3 evaluation

- calculation correct = 2 points

deduction:

- caluclation incorrect, but P_{LM} formula correct or calculation only minor mistake = 1 points

Exercise 2: Extracting LLM fingerprints (15 points)

For this task, your job is to extract the "fingerprint" of a state-of-the-art large language model from the paper. Specifically, you job is to:

- find the model that is assigned to your surname in the list **HW1_Model2Group_assignment.csv** (to be found on Moodle under topic 02). Please investigate the latest version of your model, unless the version is specified in the list.
- submit a json file with your responses in the following format (below is a partial example).

Note that, of course, it might be that some information is not available or that some categories are not applicable. The idea is, that, as a course we can create a fun website which will show a somewhat comprehensive graphical comparison of current language models and their configurations. Based on your collective json files, the lecturers will set up a front end at some point during the class.

IMPORTANT: Please email the lecturers by the homework deadline if you DO NOT consent that your json file is used for this idea.

```json
{
    "model_name": "GPT-35",
    "huggingface_model_id": "gpt35",
    "paper_url": "https://arxiv.org/abs/XXX",
    "tokenizer_type": "BPE",
    "vocabulary_size": "XXX",
    "architecture": "Mixture of transformer agents",
    "architecture_type": "decoder only",
    "architecture_quirks": [
        "sparse attention",
        "...",
    ],
    "parameters": "XXX",
    "finetuning_type": "RLHF",
    "training_data_cutoff": "2050",
    "number_training_tokens": "XXX",
    "pretraining_data_size": "1GB",
    "finetuning_data_size": "XXX",
    "training_data": [
        "Books corpus",
        "Twitter",
        "..."
    ],
    "finetuning_data": [
        "XXX",
        "XXX",
        "..."
    ],
    "access": "open",
    "summary": "A few sentences of what the model claims to be their
unique selling point / main contribution"
}

{'model_name': 'GPT-35',
 'huggingface_model_id': 'gpt35',
 'paper_url': 'https://arxiv.org/abs/XXX',
 'tokenizer_type': 'BPE',
 'vocabulary_size': 'XXX',
 'architecture': 'Mixture of transformer agents',
 'architecture_type': 'decoder only',
 'architecture_quirks': ['sparse attention', '...'],
 'parameters': 'XXX',
 'finetuning_type': 'RLHF',
 'training_data_cutoff': '2050',
 'number_training_tokens': 'XXX',
 'pretraining_data_size': '1GB',
 'finetuning_data_size': 'XXX',
 'training_data': ['Books corpus', 'Twitter', '...'],
 'finetuning_data': ['XXX', 'XXX', '...'],
 'access': 'open',
 'summary': 'A few sentences of what the model claims to be their
unique selling point / main contribution'}
```

List of models (maybe each student could even get their own model :'D). Mapping to names TBD.

- phi-2
- mixtral, mixtral-instruct
- mistral, mistral-instruct v2
- mamba
- jamba (yes i'm not joking)
- llama-2 suite
- llama-3 (8b, 70b)
- gpt-3
- gemini 1.5 (the whole suite is multimodal though)
- claude (there seems to only be a technical report, not sure how many details)
- palm-2
- Bloom (might be interesting because it's multilingual)
- Grok 1.5 (not sure there is an actual paper)
- Vicuna (blog not paper)
- Falcon 40b
- Gemma
- DBRX
- cmd-r+ (rag integrated, other provider (cohere))
- h2o danube
- BitNet
- JetMoE-8b

- qwen-1.5-MoE
- wizardLM

Exercise 3: Fine-tuning GPT-2 for QA (23 points)

The learning goal of this exercise is to practice fine-tuning a pretrained LM, GPT-2 small, for a particular task, namely commonsense question answering (QA). We will use a task-specific dataset, CommonsenseQA, that was introduced by Talmor et al. (2018). We will evaluate the performance of the model on our test split of the dataset over training to monitor whether the model's performance is improving and compare the performance of the base pretrained GPT-2 and the fine-tuned model. We will need to perform the following steps:

1. Prepare data according to steps described in sheet 1.1
    a. additionally to these steps, prepare a custom Dataset (like in sheet 2.3) that massages the dataset from the format that it is shipped in on HuggingFace into strings that can be used for training. Some of the procesing steps will happen in the Dataset.
2. Load the pretrained GPT-2 model
3. Set up training pipeline according to steps described in sheet 2.5
4. Run the training while tracking the losses
5. Save plot of losses for submission

Your tasks:

1. [19pts] Complete the code in the spots where there is a comment "#### YOUR CODE HERE ####". There are instructions in the comments as to what the code should implement. With you completed code, you should be able to let the training run without errors. Note that the point of the exercise is the implementation; we should not necessarily expect great performance of the fine-tuned model (and the actual performance will not be graded). Often there are several correct ways of implementing something. Anything that is correct will be accepted.
2. [4pts] Answer questions at the end of the execise.

```
# note: if you are on Colab, you might need to install some
requirements
# as we did in Sheet 1.1. Otherwise, don't forget to activate your
local environment

from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch
from torch.utils.data import DataLoader
from torch.utils.data import Dataset
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm

----------------------------------------------------------------------
-----
```

```
ModuleNotFoundError                       Traceback (most recent call
last)
<ipython-input-3-66d1a2d3ccf0> in <module>
      2 # as we did in Sheet 1.1. Otherwise, don't forget to activate
your local environment
      3
----> 4 from datasets import load_dataset
      5 from transformers import AutoTokenizer, AutoModelForCausalLM
      6 import torch

ModuleNotFoundError: No module named 'datasets'
```

```
# additioanlly, we need to install accelerate
# uncomment and run the following line on Colab or in your environment
# !pip install accelerate
# NOTE: in a notebook, reloading of the kernel might be required after
installation if you get dependency errors with the transformers
package

### 1. Prepare data with data prepping steps from sheet 1.1

# a. Acquiring data
# b. (minimally) exploring dataset
# c. cleaning / wrangling data (combines step 4 from sheet 1.1 and
step 1.1 above)
# d. splitting data into training and test set (we will not do any
hyperparam tuning)
# (we don't need further training set wrangling)
# e. tokenizing data and making sure it can be batched (i.e.,
conversted into 2d tensors)
# this will also happen in our custom Dataset class (common practice
when working with text data)

# downalod dataset from HF
dataset = load_dataset("tau/commonsense_qa")

# inspect dataset
print(dataset.keys())
# print a sample from the  dataset
### YOUR CODE HERE ####
print(dataset["train"][1])
```

Note that the test split does not have ground truth answer labels. Therefore, we will use the validation split as our test split.

```
# load tokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")
tokenizer.pad_token = tokenizer.eos_token
# set padding side to be left because we are doing causal LM
tokenizer.padding_side = "left"
```

```python
def massage_input_text(example):
    """
    Helper for converting input examples which have
    a separate qquestion, labels, answer options
    into a single string.

    Arguments
    ---------
    example: dict
        Sample input from the dataset which contains the
        question, answer labels (e.g. A, B, C, D),
        the answer options for the question, and which
        of the answers is correct.

    Returns
    -------
    input_text: str
        Formatted training text which contains the question,
        the forwatted answer options (e.g., 'A. <option 1> B. <option
2>' etc)
        and the ground truth answer.
    """
    # combine each label with its corresponding text
    answer_options_list = list(zip(
        example["choices"]["label"],
        example["choices"]["text"]
    ))
    # join each label and text with . and space
    ### YOUR CODE HERE ####
    answer_options = [f"{label}. {text}" for label, text in
answer_options_list]
    # join the list of options with spaces into single string
    ### YOUR CODE HERE ####
    answer_options_string = " ".join(answer_options)
    # combine question and answer options
    input_text = example["question"] + " " + answer_options_string
    # append the true answer with a new line, "Answer: " and the label
    input_text += "\nAnswer: " + example["answerKey"]

    return input_text

# process input texts of train and test sets
massaged_datasets = dataset.map(
    lambda example: {
        "text": massage_input_text(example)
    }
)

# inspect a sample from our preprocessed data
massaged_datasets["train"][0]
```

```
class CommonsenseQADataset(Dataset):
    """
    Custom dataset class for CommonsenseQA dataset.
    """

    def __init__(
            self,
            train_split,
            test_split,
            tokenizer,
            max_length=64,
            dataset_split="train",
        ) -> None:
        """
        fill me.
        """
        self.train_split = train_split['text']
        self.test_split = test_split['text']
        self.tokenizer = tokenizer
        self.max_length = max_length
        self.dataset_split = dataset_split

    def __len__(self):
        """
        Method returning the length of the training dataset.
        """
        ### YOUR CODE HERE ####
        return len(self.train_split)

    def __getitem__(self, idx):
        """
        Method returning a single training example.
        Note that it also tokenizes, truncates or pads the input text.
        Further, it creates a mask tensor for the input text which
        is used for causal masking in the transformer model.

        Arguments
        ---------
        idx: int
            Index of training sample to be retrieved from the data.

        Returns
        ---------
        tokenized_input: dict
            Dictionary with input_ids (torch.Tensor) and an
attention_mask
            (torch.Tensor).
        """        # retrieve a training sample at the specified index
idx
        # HINT: note that this might depend on self.dataset_split
        ### YOUR CODE HERE ####
        input_text = self.train_split[idx]
        tokenized_input = self.tokenizer(
            input_text,
            ### YOUR CODE HERE ####
            max_length=self.max_length,
            padding="max_length",
            truncation=True,
            return_tensors="pt"
        )
        tokenized_input["attention_mask"] =
(tokenized_input["input_ids"] != tokenizer.pad_token_id).long()
        return tokenized_input

# move to accelerated device
if torch.cuda.is_available():
    device = torch.device("cuda")
    print(f"Device: {device}")
elif torch.backends.mps.is_available():
    device = torch.device("mps")
    print(f"Device: {device}")
else:
    device = torch.device("cpu")
    print(f"Device: {device}")

# 2. init model

# load pretrained gpt2 for HF
### YOUR CODE HERE ####
model = AutoModelForCausalLM.from_pretrained("gpt2")
# print num of trainable parameters
model_size = sum(t.numel() for t in model.parameters())
print(f"GPT-2 size: {model_size/1000**2:.1f}M parameters")

# 3. set up configurations required for the training loop

# instantiate dataset
train_dataset = CommonsenseQADataset(
    ### YOUR CODE HERE ####
    massaged_datasets["train"],
    massaged_datasets["test"],
    tokenizer
)
# instantiate test dataset with the downloaded commonsense_qa data
test_dataset = CommonsenseQADataset(
    ### YOUR CODE HERE ####,
    massaged_datasets["train"],
    massaged_datasets["test"],
    tokenizer,
    dataset_split="test"
)
# create a DataLoader for the dataset
# the data loader will automatically batch the data
# and iteratively return training examples (question answer pairs) in
batches
dataloader = DataLoader(
    train_dataset,
    batch_size=32,
    shuffle=True
)
# create a DataLoader for the test dataset
# reason for separate data loader is that we want to
# be able to use a different index for retreiving the test batches
# we might also want to use a different batch size etc.
test_dataloader = DataLoader(
    test_dataset,
    batch_size=32,
    shuffle=True
)

# 4. run the training of the model
# Hint: for implementing the forward pass and loss computation,
carefully look at the exercise sheets
# and the links to examples in HF tutorials.

# put the model in training mode
model.train()
# move the model to the device (e.g. GPU)
model = model.to(device)

# trianing configutations
# feel free to play around with these
epochs  = 2
train_steps =  len(train_dataset) // 32
print("Number of training steps: ", train_steps)
# number of test steps to perform every 10 training steps
# (smaller that the entore test split for reasons of comp. time)
num_test_steps = 5

# define optimizer and learning rate
optimizer = torch.optim.AdamW(model.parameters(), lr=5e-4) ### YOUR
CODE HERE ###
# define some variables to accumulate the losses
losses = []
test_losses = []

# iterate over epochs
for e in range(epochs):
    # iterate over training steps
    for i in tqdm(range(train_steps)):
        # get a batch of data
        x = next(iter(dataloader))
        # move the data to the device (GPU)
        ### YOUR CODE HERE ###
        x = x.to(device)

        # forward pass through the model
        ### YOUR CODE HERE ###
        outputs = model(
            **x,
            labels=x["input_ids"]
        )
        # get the loss
        ### YOUR CODE HERE ###
        loss = outputs.loss
        # backward pass
        ### YOUR CODE HERE ###
        loss.backward()
        losses.append(loss.item())
        # update the parameters of the model
        ### YOUR CODE HERE ###
        optimizer.step()
        # zero out gradient for next step
        ### YOUR CODE HERE ###
        optimizer.zero_grad()

        # evaluate on test set every 10 steps
        if i % 10 == 0:
            print(f"Epoch {e}, step {i}, loss {loss.item()}")
            # TODO
            # track test loss for the evaluation iteration
            test_loss = 0
            for j in range(num_test_steps):
                # get test batch
                x_test = next(iter(test_dataloader))
                x_test = x_test.to(device)
                with torch.no_grad():
                    test_outputs = model(
                        **x_test,
                        labels=x_test["input_ids"]
                        ### YOUR CODE HERE ####
                    )
                test_loss += test_outputs.loss### YOUR CODE HERE ####

            test_losses.append(test_loss / num_test_steps)
            print("Test loss: ", test_loss/num_test_steps)

# Plot the fine-tuning loss
### YOUR CODE HERE ###
plt.plot(losses)
plt.xlabel("Training steps")
plt.ylabel("Loss")

# print a few predictions on the eval dataset to see what the model
predicts

# construct a list of questions without the ground truth label
# and compare prediction of the model with the ground truth

def construct_test_samples(example):
    """
    Helper for converting input examples which have
    a separate qquestion, labels, answer options
    into a single string for testing the model.

    Arguments
    ---------
    example: dict
        Sample input from the dataset which contains the
        question, answer labels (e.g. A, B, C, D),
        the answer options for the question, and which
        of the answers is correct.

    Returns
    -------
    input_text: str, str
        Tuple: Formatted test text which contains the question,
        the forwatted answer options (e.g., 'A. <option 1> B. <option
2>' etc);
        the ground truth answer label only.
    """

    answer_options_list = list(zip(
        example["choices"]["label"],
        example["choices"]["text"]
    ))
    # join each label and text with . and space
    ### YOUR CODE HERE ####
    answer_options = [f"{label}. {text}" for label, text in
answer_options_list]
    # join the list of options with spaces into single string
    ### YOUR CODE HERE ####
    answer_options_string = " ".join(answer_options)
    # combine question and answer options
    input_text = example["question"] + " " + answer_options_string
    # create the test input text which should be:
    # the input text, followed by the string "Answer: "
    # we don't need to append the ground truth answer since we are
creating test inputs
    # and the answer should be predicted.
    ### YOUR CODE HERE ####
    input_text += "\nAnswer: "

    return input_text, example["answerKey"]

test_samples = [construct_test_samples(dataset["validation"][i]) for i
in range(10)]
test_samples

# Test the model

# set it to evaluation mode
model.eval()

predictions = []
for sample in test_samples:
    input_text = sample[0]
    input_ids = tokenizer(input_text, return_tensors="pt").to(device)
    output = model.generate(
        input_ids.input_ids,
        attention_mask = input_ids.attention_mask,
        max_new_tokens=2,
        do_sample=True,
        temperature=0.4,
    )
    prediction = tokenizer.decode(output[0], skip_special_tokens=True)
    predictions.append((input_text, prediction, sample[1]))

print("Predictions of trained model ", predictions)
```

Questions:

1. Provide a brief description of the CommonsenseQA dataset. What kind of task was it developed for, what do the single columns contain? **Answer** The CommonsenseQA dataset contains common sense question, i.e. questions where finding the correct answer involved basic world knowledge. These are presented as a question, a question concept, a number of labelled possible answers, and the label of the correct answer.
2. What loss function is computed? Where is it implemented? **Answer** The loss function is Negative Log-Likelihood Loss (you can show this by print(loss), which will show grad_fn=)
3. Given your loss curve, do you think your model will perform well oon answering common sense questions? (there is no single right answer; you need to interpret your specific plot)
4. Inspect the predictions above. On how many test questions did the model predict the right answer? Compute the accuracy.

   $Accuracy = \frac{#correct\:predictions}{#total\:predictions}$