To get access to this week's code use the following link: https://classroom.github.com/a/tnxDFYZD

General constraints for submissions: Please adhere to these rules to make our and your life easier! We will deduct points if you fail to do so.

- Your code should work with *Python 3.8*.
- You should only fill out the *TODO-gaps* and not change anything else in the code.
- Add comments to your code, to help us understand your solution.
- Your code should adhere to the PEP8 style guide. We allow line lengths of up to 120.
- While working on the exercise, push all commits to the `dev` branch (details in assignment 1). Only push your final results to the `main` branch, where they will be automatically tested in the cloud. If you push to `main` more than 3 times per exercise, we will deduct points.
- All provided unit tests have to pass: In your *GitHub* repository navigate to *Actions* → your last commit → Autograding → education/autograding to see which tests have passed. The points in autograding only show the number of tests passed and have nothing to do with the points you get for the exercise.
- `for` loops can be slow in Python, use vectorized `numpy` operations wherever possible (see assignment 1 for an example).
- Submit *a single PDF* named `submission.pdf`. Include your matriculation numbers on the top of the sheet. Add the answers and solution paths to all non-coding questions in the exercise. Do not leave answers to any questions as comments in the code. You can use Latex with the student template (provided in exercise 1 / ILIAS) or do it by hand.
- Please help us to improve the exercises by filling out and submitting the `feedback.md` file.
- We do not tolerate plagiarism. If you copy from other teams or the internet, you will get 0 points. Further action will be taken against repeat offenders!
- Passing the exercises ($\geq 50\%$ in total) is a requirement for passing the course.

How to run the exercise and tests

- See the `setup.pdf` in exercise 1 / ILIAS for installation details.
- We always assume you run commands in the *root folder* of the exercise repository.
- If you use miniconda, do not forget to activate your environment with `conda activate mydlenv`
- Install the required packages with `pip install -r requirements.txt`
- Python files in the *root folder* of the repository contain the scripts to run the code.
- Python files in the `tests/` folder of the repository contain the tests that will be used to check your solution.
- Test everything at once with `python -m pytest`
- Run a single test with `python -m tests.test_something` (replace `something` with the test's name).
- To check your solution for the correct code style, run `pycodestyle --max-line-length 120 .`
- The scripts `runtests.sh` (Linux/Mac) or `runtests.bat` (Windows) can be used to run all the tests described above. If you are on Linux, you need execution rights to run `runtests.sh`.

This exercise focuses on *attention* and how it can be learned in deep neural networks. In particular, we will:

1. Visualize attention as learned implicitely by a deep CNN on an image classification task
2. Implement explicit attention and use it to solve a simple counting task
3. Complete a transformer implementation and apply it to a translation task

Note that effectively learning attention typically requires large models and a lot of data. To keep computational requirements for this exercise low, we will use pre-trained models in 1 and 3. Only in 2, we will learn attention from scratch, however here the task and model architecture are carefully chosen to allow attention to be learned efficiently.

Data Setup: This exercise requires auxiliary data that is not part of the starter code. Before starting, download *transformers_exercise_data.zip* and unpack it as `data` in the root folder of your project.

1. Warmup: The Jacobian, Implicit Attention, and Adversarial Examples

Before we start working with explicit attention and transformers, we will have a look at how a classical neural

network implicitly encodes attention. In particular, we will visualize how a pretrained ResNet implicitly focuses on certain parts of the input image.

1) [1 point] **Calculate the Jacobian:**
Todo: Fill the TODOs in `calculate_jacobian` (`lib/utils.py`).

Run `python -m tests.test_jacobian` to test your implementation.

Hint: Modify the `requires_grad` attribute of the input tensor to calculate its `grad` attribute in the backward pass.

2) [2 points] **Attention on Input:**
Todo: Fill the TODOs in `show_attention_on_input_image` (`lib/plot.py`) and run `plot_attention_on_input.py`. Include the outputs for the four suggested samples into the `submission.pdf`, describe what you see and briefly discuss why this makes sense or not.

3) [1 point] **Adversarial Examples:** In the Regularization lecture we saw that small changes (invisible to the human eye) in the input image can 'trick' a neural network to misclassify the input. Such adversarial examples can be generated by performing gradient ascent on the input, in the direction of the Jacobian.

Todo: Fill the TODOs in `show_adversarial_example` (`lib/plot.py`) and run `plot_adversarial_example.py` to do so for the three suggested samples and include the resulting figures into the `submission.pdf`.

Note: Try scaling the Jacobian when generating the adversarial examples to increase its contribution.

2. Learning to Count using Explicit Attention

We will now implement explicit attention and show how it can be used to learn to solve a simple counting task. In this task, the goal is to count the frequency with which certain digits appear in a given input sequence. For example, assume 3 possible digits (i.e., vocabulary size = 3), given an input sequence `[1, 0, 2, 1, 0, 1, 1, 0, 1, 2]`, the output should be `[3, 5, 2]`.

1) [2 points] **Scaled Dot-Product Attention:** Implement the attention function similar to the one described in *section 3.2.1* of the seminal paper "Attention is All You Need" (Vaswani et al., 2017), but apply the sigmoid instead of the softmax operation i.e.,

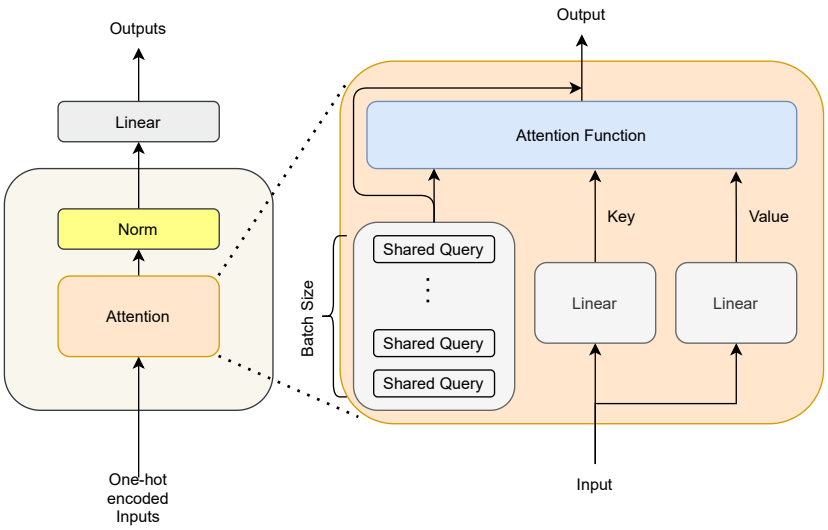
$$\text{attention(Q, K, V)} = \text{sigmoid}(\frac{QK^T}{\sqrt{d_k}})V$$

Todo: Fill the TODOs in `attention_function` (`lib/attention.py`)

Run `python -m tests.test_attention` to test your implementation.

Note: Next to the attention output, this function should also return the attention weights.

2) [3 points] **Counting Model:** Now, we will use this attention function in the implementation of a model for solving the counting task. The architecture of this model is depicted below.



Here, keys (K) and values (V) are determined as (different) linear combinations of the one-hot encoded input. The queries (Q) only depend on the output (i.e., the digit to be counted) and the query string for each digit is itself a trainable parameter and is the same for each element in the batch. The attention output is added to the query, normalized, and each augmented query is finally passed through a linear layer with an output for every possible count (i.e., [0, sequence length]).

Todo: Fill the TODOs in `CountingModel` (`lib/counting.py`)

Run `python -m tests.test_counting` to test your implementation.

3) **Training the Model:** Run `train.py` to train your network to minimize the `CrossEntropy` loss on randomly generated sequences of a fixed length of 10 and vocabulary size of 3.

4) [2 points] **Visualizing Explicit Attention:**

Plot a heatmap of the attention weights to visualize explicit attention.

Todo: Fill the TODOs in `plot_attention` (`lib/plot.py`)

Run `plot_attention.py` to visualize the attention.

Include the plot and briefly discuss your observations in your `submission.pdf`.

5) [1½ bonus points] **Model with Softmax:**

As already mentioned, in the counting task we deviated a little from the original attention function given in the seminal paper "Attention is All You Need" (Vaswani et al., 2017).

In this specific task applying the softmax, i.e.:

$$\text{attention(Q, K, V)} = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

would actually hurt the performance of the model.

Todo: Verify that the performance drops if you apply the softmax operation and explain why that is the case.

3. Neural Machine Translation

We will now apply the transformer model to a German to English translation task.

1) [4 points] **Transformer Model:** First, complete the implementation of the transformer in `lib/transformer.py`. Here, you are to implement the forward pass of the model by putting together the different components as depicted in *Figure 1* of (Vasvani et al., 2017).

   Todo: Fill the TODOs in `TransformerModel` (`lib/transformer.py`).

   Run `python -m tests.test_transformer` to test your implementation.

   Note: The provided starter code already initializes all components, you just need to combine them correctly.

   Note: The code uses learned positional encodings (original paper used predefined sinusoidal positional encodings).

   Note: The `MultiHeadAttention` module (`lib/transformer.py`) also implements the residual connection and subsequent normalization.

   Note: The feedforward module is an MLP with a single hidden layer with ReLU activation having a number of hidden units twice the embedding size.

   Note: The final softmax operation is not part of the model, but of the `CrossEntropy` loss.

2) [1 point] **Translation Task:**
   Test the pre-trained model using the script `python3 run_translation.py --test --line=7`. The `--line` flag specifies the desired sample sentence (here the 7th sentence). This takes a number between 1 and 209773. Note that the sentences might not all be meaningful. Repeat the test multiple times for different samples by changing the `--line` option.

   Todo: Choose your favorite sentence and add the `--plot` flag to the script to plot the corresponding attention heatmaps (e.g. `python3 run_translation.py --test --line=7 --plot`). Include the plots in the `submission.pdf` file and interpret the results.

4. [1 bonus point] Code Style

On each exercise sheet, we will also be using `pycodestyle` to adhere to a common Python standard. `pycodestyle` checks your Python code against some common style conventions in PEP 8 and reports any deviations with specific error codes.

Your code will be automatically evaluated on submission (on push to `main`). You can run `pycodestyle --max-line-length=120 .` to manually evaluate your code before submission.

One bonus point will be awarded if there are no code style errors detected in your code by the `pycodestyle --max-line-length=120 .` command.

Translation task adapted from attention-primer-pytorch repository.

5. [1 bonus point] **Feedback**

Todo: Please give us feedback by filling out the `feedback.md` file.

- Major Problems?
- Helpful?
- Duration (hours)? For this, please follow the instructions in the `feedback.md` file.
- Other feedback?

This assignment is due on 18.12.2024 (23:59 CET). Submit your solution for the tasks by uploading (`git push`) the PDF, txt file(s) and your code to your group's repository. The PDF has to include the name of the submitter(s). Teams of at most 3 students are allowed.