Assignment 4 – Transformers

CSCI-LING 5832 – Spring 2025

Due 03/26/2025

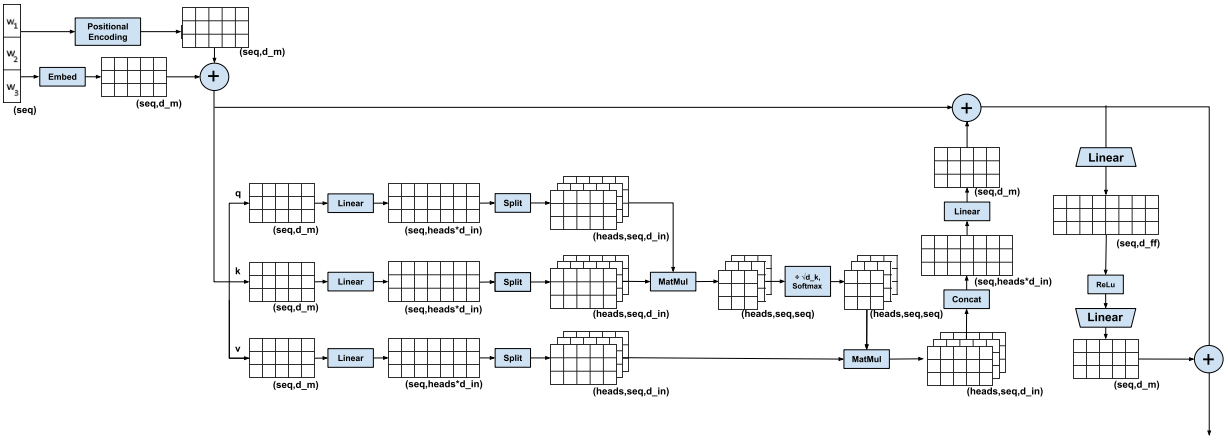
Disclaimer: All code turned in is expected to be python. You are given the choice of making a script file (`*.py`) or a jupyter notebook (`*.ipynb`). Whichever route you choose, please ensure your final submission is clean and well-organized – enough so as to be quickly evaluated by your TAs.

We suggest using a Notebook on this assignment. If you have any difficulties setting up Jupyter, feel free to reach out to a TA at an office hours for help.

Allowed Packages: In this assignment, you are not-allowed to use any off-the-shelf implementations of attention mechanisms, such as `nn.TransformerEncoder`, `nn.TransformerDecoder`, etc. The assignment can be completed using only the functions we have imported and Tensor manipulations (`squeeze`, `unsqueeze`, `view`, `split`, `chunk`, `concat`). If you desire to use others, feel free to ask on Piazza or at Office Hours.

Submission Format: You are expected to return your code (as `*.ipynb` or `*.py`, ideally as 1 file), and a seperate report file (pdf) with the ***Deliverables*** listed after every part.

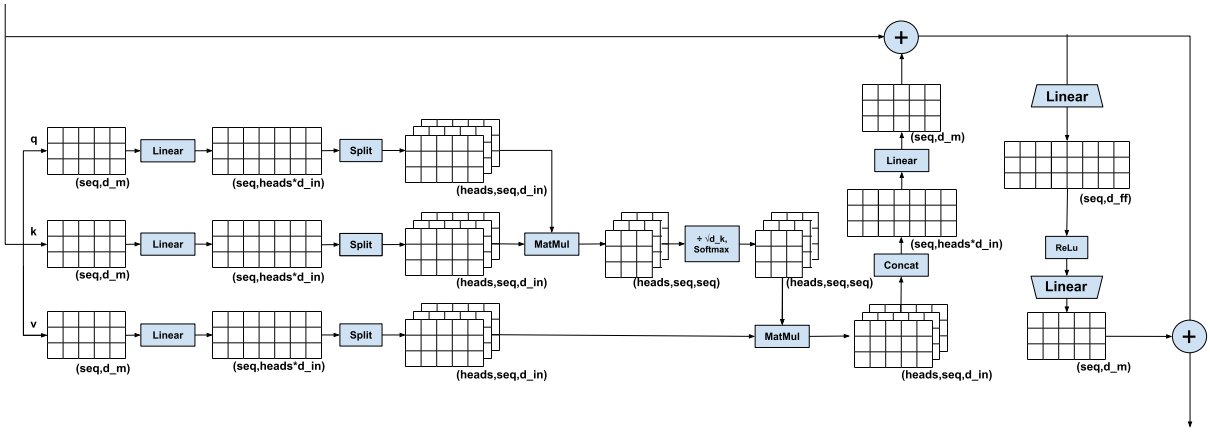
The Transformer Architecture



Part 1: Multi-head Attention and the Encoder Layer

In this section, you will implement a transformer encoder layer and store the attention maps. We will attempt to guide you through the implementation through the starter code and visualizations; for more information on the architecture we recommend the following resources: *The Illustrated Transformer*, *Jurafsky-Martin Ch.9*, and *Attention is All You Need*. Note that our model is somewhat simplified; we use absolute positional encodings and no LayerNorms. We also do not require any attempts at batching or masking inputs.
Tips:

- We recommend you start with a single head, and make sure to check the dimensions of your Tensors with `.shape` after each calculation.
- Make sure to store a deep copy of your residuals, rather than references.



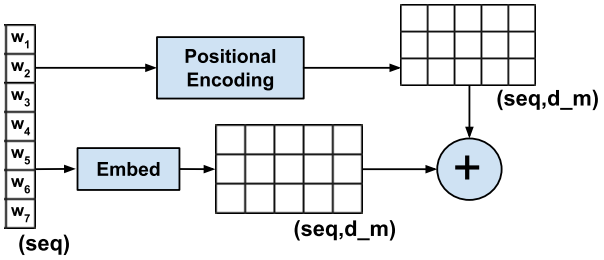
- The attention map(s) are the $softmax(\frac{QK^T}{\sqrt{d_k}})$ term(s).
  - We recommend you cache this as a layer-property rather than include it during inference, as the latter strategies complicates transformers with more than one layer.

Part 1 Deliverable: Return the outputs of the single-head, and 2-head test cases as described in the Notebook and Python file.

Part 2: Embeddings, Positional Encodings, and the Module

Next, we will be creating the Transformer model base.
Your task is to implement the embeddings and the forward pass.

Embeddings



Each token's embedding should be the sum of two embeddings:

1. the word embedding that embeds the vocabulary in a `d_model`-dim space.
2. the absolute positional encoding that embeds each index in a `d_model`-dim space. Since all the inputs in this assignment will be of length `max_seq_len`, you can consider it a mapping from the `max_seq_len`-dim indices to `d_model`, and embed each index using a range.

Forward Pass

1. Apply the embeddings
2. Apply the `nn.Sequential` Encoder(s).
   (a) Addditionally, iterate through `nn.Sequential` to retrieve the cached maps for each layer

3. Apply the simplified decoder layer

4. Convert the outputs to log probabilities over the tokens, and return them along with the maps

Part 2 Deliverable: Provide the embedding (word embedding plus positional encoding) for [1,1,2,2,3], using `d_model`=16, `max_seq_len`=5.

Part 3: Task and Dataset Processing

The dataset for this transformer implementation will be letter-counting. Given a string, your model should predict how many times that character has occurred previously. To simplify the computation, this will be split into 3 bins, $n = 0$, $n = 1$, and $n \geq 2$

Specifically for we be considering two settings:
1) Task 1 - counts of the letter *before* the current index
2) Task 2 - counts of the letter *before* and *after* the current index

We provide helper functions for formatting the tasks, your job is to write a function `indexer` that takes the 20 character strings and converts them to `LongTensor`s of IDs.

Read from `lettercounting-train.txt`, `letttercounting-dev.txt`. For both the train and eval set, collect an array for the strings, the ID tensors, and the gold standard outputs for both tasks.

Part 3 Deliverable: Provide the ID tensor and task tensors for the string `"ed by rank and file"`.

Part 4: Training Loop and Evaluation

Now that we have both our model and our dataset, it is time to train and evaluate on our tasks.

The following code blocks provide a helper function for visualizing your attention maps (so you can see it learning), as well as loss functions / optimizers / learning rate schedulers that may come in handy.

To complete this section, you must provide us with models that achieve >90% eval accuracy on each task. If implemented properly, this can be done with 1 layer and 1 head/layer, however, we encourage you to experiment.
In your solution, provide us with the following:

- train losses logged at 100-250 steps
- eval losses logged every 5-10 train loss updates
- for each eval update, the accuracy and macro-F1 scores of the train/eval.
  - this is best done accumulating arrays of the predictions and model outputs throughout the training loop, and then calls the to the scikit-learn metrics.

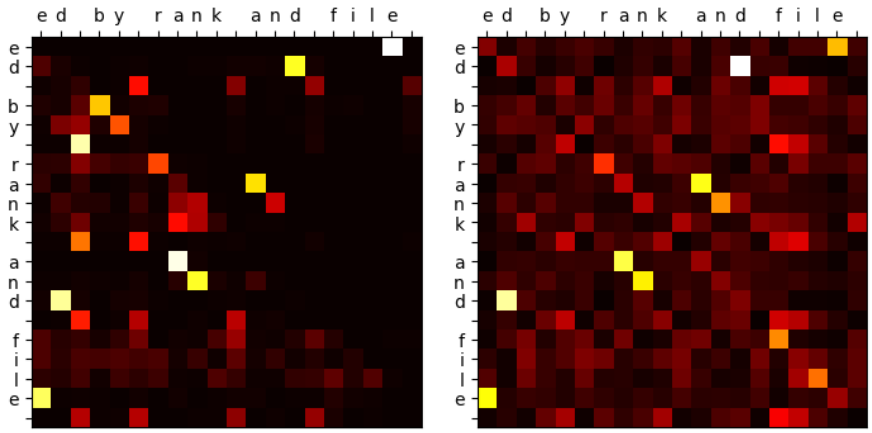
You can return these in any form you would like (table, graph, etc), as long as they are immediately interpretable (for example, normalizing the train/eval loss for number of examples would be necessary).

Additionally, describe any issues you had with selecting on optimal learning rate, loss function, etc.

Part 4 Deliverable: Train/eval losses logged over time, as well as accuracy/macro-F1. Passage describing difficulties with hyperparameter selection / loss functions / optimizers.

Part 5: Attention Map Visualization

Lastly we will examine the computation performed by the model(s) you've trained using its attention maps. The following are 1-layer, 1-head models trained on Task 1 (the before task), and Task 2 (the before/after task) respectively. They both achieve accuracy ∼95%.



5.1) Hypothesize on how these attention maps reflect the computation done by their associated models.

Specifically address how each handles the tokens ['e', 'a', 'n', 'd', ' '], how they are similar, and how/why they maybe different.

5.2) Train a multi-layer (>4), single-head model on one of the tasks.

Achieve an eval accuracy of at least 80%, and plot the attention maps below on the string "ed by rank and file". Hypothesize on how each layer contributes to the classification.

5.3) Train a multi-head model on one of the tasks.

Achieve an eval accuracy of at least 80%, and plot the attention maps below on the string "ed by rank and file". Hypothesize on how each head contributes to the classification.

Part 5 Deliverable: Answers to 5.1/5.2/5.3, and attention maps for 5.2/5.3.

Rubric

Base Points Item

- 25 Attention Mechanism - (Part 1 Deliverable: Single-head)
- 10 Multi-head attention - (Part 1 Deliverable: Multi-head)
- 10 Embedding, Positional encoding - (Part 2 Deliverable)
- 10 Data processing - (Part 3 Deliverable)
- 15 Task 1 - 90% accuracy and visible training/eval info - (Part 4 Deliverable: Task 1 losses/Acc/F1, any issues with training)
- 15 Task 2 - 90% accuracy and visible training/eval info - (Part 4 Deliverable: Task 2 losses/Acc/F1, any issues with training)
- 5 Attention Maps and visualization (Part 5.1 Deliverable: Analysis)
- 5 Multi-layer Attention Maps - (Part 5.2 Deliverable: Analysis and Attn. Map)
- 5 Multi-head Attention Maps - (Part 5.3 Deliverable: Analysis and Attn. Map)

Point Multipliers Criteria

- 1 All expected functionality is present. All deliverables are answered in the report pdf.
- 0.75 Minor errors not impacting general functionality. Output may slightly deviate from what is expected. All deliverables are have an answer, although lacking in some regard.
- 0.5 Errors demonstrating a lack of understanding. Some functionality missing. Deliverables were attempted.
- 0.25 Significant errors affecting the functionality of the item. Deliverables were may be incomplete.
- 0 No work present or code does not run. No report or answers in the notebook.

Acknlowedgements

Acknowledgement This homework was adapted from CS388, taught by Greg Durrett at University of Texas-Austin, and CS685, taught by Mohit Iyyer, then at University of Massachusetts Amherst. The visualizations take inspiration from a twitter post by Mark Riedl of Georgia Tech