Computer Vision II, Fall 2023
Assignment 2

Prof. Jingya Wang

November 16, 2023

Acknowledgement

1. Deadline: 2023/12/26 23:59:00. Late Policy please refers to the course slides.
2. Giving your report in English, report in Chinese is not accepted. Besides, handwritten homework is not accepted and we highly recommend you to use LaTeX. LaTeX template has been upload to Blackboard.
3. Please use the code templates (`https://epan.shanghaitech.edu.cn/l/7F2af0`).
4. Please submit your assignment in Gradescope with PDF format. Registration code: ZWP2JB.
5. Please upload your code zip to ShanghaiTech cloud disk. `https://epan.shanghaitech.edu.cn/l/wFKUdg`. All source files and readme should be included but remember to remove the datasets. Your zip should be named as CS272_NAME_ID_hw2.zip.
6. Plagiarism or cheat is strictly prohibited. Do not share your assignment or code! No fake solution is allowed! Make sure that your codes can run and are consistent with your solutions

Image Captioning with Transformers (30 points)

We will be implementing the different pieces of a Transformer decoder (Transformers), and train it for image captioning on a subset of the COCO dataset.

- **Setup:** Run the following command to extract COCO data, in the `transformer_captioning/datasets` folder : `./get_coco_captioning.sh`
- **Question:** Follow the instructions in the `README.md` file in the `transformer_captioning` folder to complete the implementation of the transformer decoder.
- **Deliverables:** After implementing all parts, use run.py for training the full model. The code will log plots to `plots`. Extract plots and paste them into the appropriate section below.
- **Expected results:** These are expected training losses after 100 epochs. Do not change the seed in run.py.
  - 2-heads, 2-layers, lr 1e-4: Final loss ≤ 1
  - 4-heads, 6-layers, lr 1e-4: Final loss ≤ 0.3
  - 4-heads, 6-layers, lr 2e-4: Final loss ≤ 0.05

Visual Question Answering (70 points)

Download dataset: Run `./download_data.sh` to download and unzip everything. The command downloads the train and validation data from https://visualqa.org/vqa_v1_download.html(we only need 'Real Images', not 'Abstract Scenes'). We're using VQA v1.0 Open-Ended. If for any reason the above script does not work for you, download the data directly from the website. We need the annotations, the questions and the images for both the training and validation sets (see what the script is trying to download for details).

Task1: Understanding VQA (20 points)

VQA is often addressed as a classification problem where we map image-question pairs to a pool of predefined answers. To create this pool of answers, a naive way is to collect all different answer strings of the training set and treat them as separate class. However, this creates a very long tail of classes and only makes the problem harder and the inference slower computationally. To account for this, we sort the answer by frequency and keep the 5216 most frequent ones (5216 is a hyperparameter used by the first VQA papers). To ensure fair pre-processing and filtering, we have included the code that sorts the answers and creates the answer-to-class map. We also include a 5217th class that corresponds to 'other'. All rare answers are mapped to 'other'. This makes VQA a 5217-way classification problem. Note that the answers are phrases, not necessarily single words. For example a valid answer can be 'riding a bike'.

Based on the above description:

1.1: Why do you think that it's computationally costly to use all answers as separate classes instead of keeping only the most frequent ones?

1.2: Complete the $\_\_init\_\_$ method of the dataset class for VQA in `vqa_dataset.py`. Specifically, initialize the VQA API and anything you need from that.

1.3: Implement the $\_\_len\_\_$ method of the VQADataset class. Should the size of the dataset be equal to the number of images, questions or the answers?

1.4: Complete the $\_\_getitem\_\_$ method. You need to (1) figure out what is the idx-th item of the dataset; (2) load the question string; (3) fill the image preprocessing part. The $\_\_getitem\_\_$ method returns the question and list of answers, the preprocessed (normalized) image, as well as the original image for visualization purposes.

Additionally, the VQA dataset contains 10 answers per question. These are usually synonyms/paraphrases (or 'other', if they're very rare). Our dataset class considers all of them and returns a binary vector indicating the validity of each answer (field `answers`). This means that `item['answers']` is a binary vector of length 5217.

Task 2: Building a pipeline for VQA (30 points)

To build a learning pipeline we need i) a dataset class (we have already built it!), ii) a model that absorbs the data and returns some predictions, iii) a training pipeline that optimizes these predictions and iv) metrics that evaluate the predictions.

Model As you've already figured out, a VQA model for this assignment is simply a classifier over 5217 classes. The first baseline is very simple. We featurize the image using a frozen pre-trained ResNet18. For that, we feed the image to ResNet18 except for the last layer which gives the classification logits. This gives us an 1-d feature vector for the image. We also featurize the question into an 1-d vector using a frozen pre-trained RoBERTa, a Transformer-based language model. Finally, we concatenate the two representations and feed to a linear layer. The parameters of this linear layer are trainable. The code for loading the pre-trained models is given. The forward pass of the language model is also given. Based on the above description:

2.1: What should be the output dimension of the trainable linear layer? Complete the corresponding part in the $\_\_init\_\_$ method of `BaselineNet` in `models.py`

2.2: Implement `compute_vis_feats` that featurizes the image (it can be implemented as an one-liner!).

2.3: Implement the forward pass of `BaselineNet`. Use `compute_vis_feats` and `compute_text_feats`.

Training loop We provide skeleton code in `main.py`. We use Adam optimizer, although you can experiment with different optimizers and hyperparameter values. As mentioned earlier, the VQA dataset contains 10 synonym/-paraphrased/duplicate answers per question. A standard softmax classification would create competition among such answers. To avoid that, we treat all of them as correct.

2.4: Implement the loss call and the optimization code in the `train_test_loop` method.

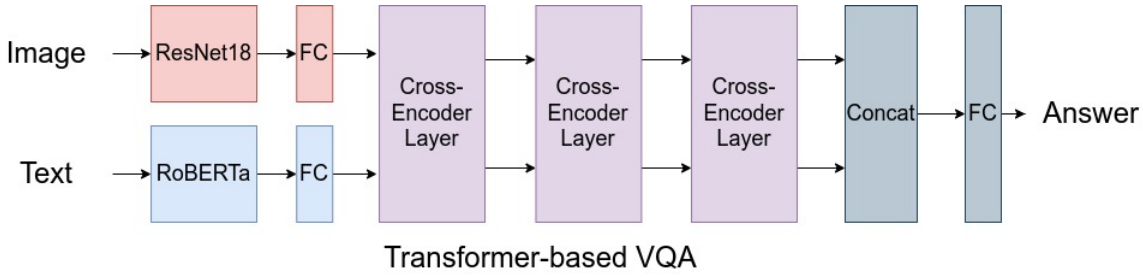
Evalution The prediction is considered accurate if the most confident answer is correct, i.e. if it is one of the multiple correct answers. We provide the code to compute accuracy.

2.5: Complete the `train_test_loop` method to monitor the performance in Tensorboard. Plot the loss and accuracy and include these in your report. Additionally show multiple image-question pairs (at least 3 successful and 3 failure cases) with the respective answers (predicted and ground-truth). For the accuracy, you can either plot it after every step or every epoch. For the image-question pairs, just show results from your best model, not at intermediate epochs. If an image-question pair has more than one ground-truth answers, randomly select one to show. The image-question pair visualizations don't have to be on Tensorboard, you can use any visualization tool you want.

Now, we are ready to train/test the model! Aim for a validation accuracy of 50%, but anything above 45% is fine. You're free to use different hyperparameters but not to change the architecture at this point.

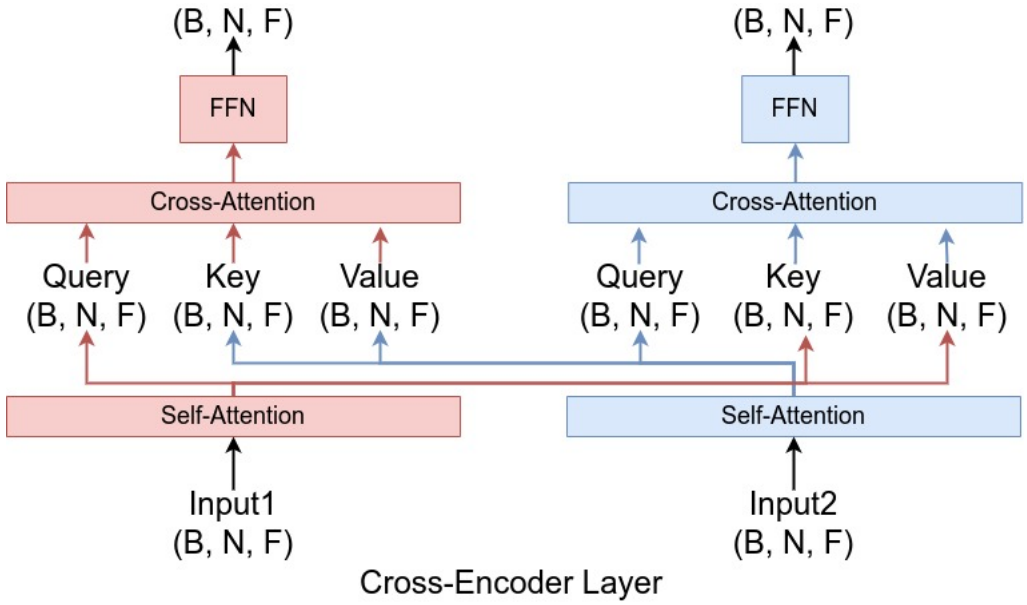
Task3: Transformer Network (20 points)

The baseline model has obvious limitations: it condenses the whole image (question) into an 1-d vector and there's no interaction between the visual and textual features. We are now going to enhance this model with attention layers, using Transformers. Here is the model we are going to implement:



Transformer-based VQA

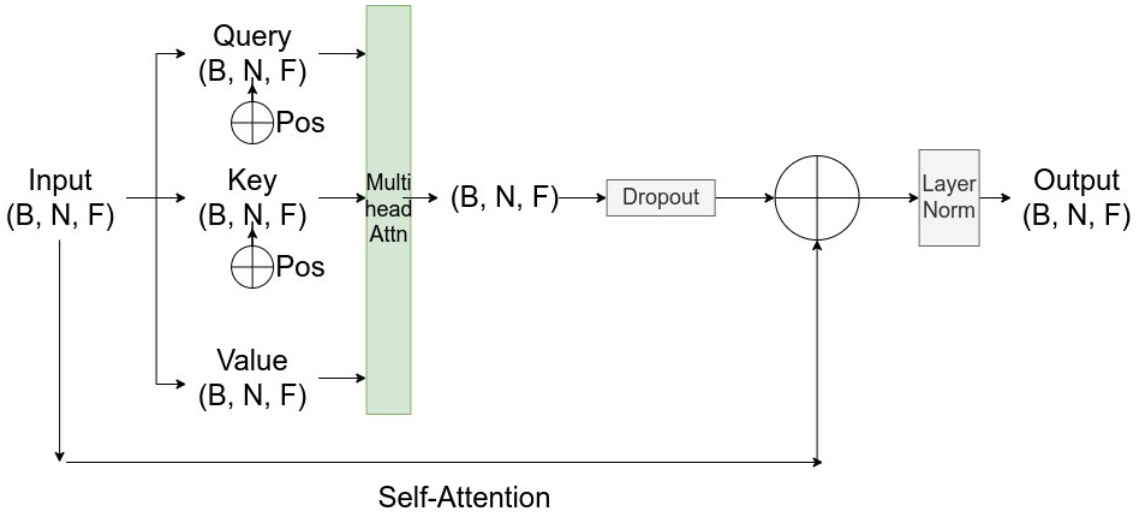
After encoding each modality separately and projecting them to have the same feature dimension, we apply three cross-encoding layers. The refined visual/text features are then mean pooled, concatenated and fed to a linear layer that predict confidence scores over the possible answers. This pipeline is already implemented in `models.py` as `TransformerNet`. However, the cross-encoder layer is not implemented. In the following, B is the batch size, N the sequence length (e.g. number of visual or language tokens) and F the feature dimension. Make sure that your model supports any batch size. A high-level diagram of the cross-encoder layer can be seen here:



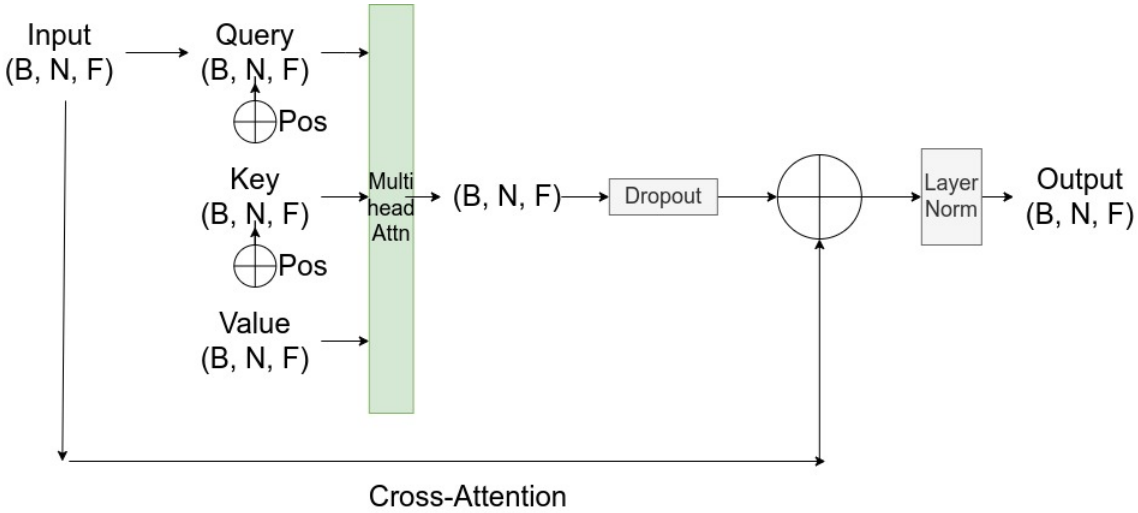
Cross-Encoder Layer

We are going to implement this layer step-by-step (see TODOs in `models.py`).

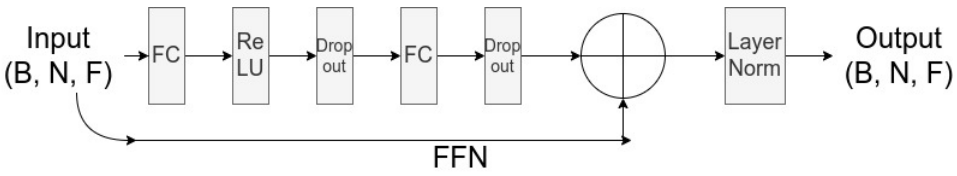
First, the self-attention layer allows interaction among the different tokens of a single modality:



Next, the cross-attention layer allows interaction among different modalities:



Pay attention to the query/key/value now as they come from different modality!
Lastly, the feed-forward network is just an MLP (see code for exact dimensions):



Keep in mind that the input to and the output of a cross-encoder layer have identical shape.
For the visual stream we also use positional embeddings to denote the position of each patch. This is a generalization of the embeddings introduced in "Attention is all you Need" for images.
3.1: Complete the `CrossAttentionLayer`.

Hints: The current `__init__` method contains the names of all layers that you will need. We've implemented one of the self-attention layers for reference. Build the others based on that and the comments. `nn.MultiheadAttention` directly implements multi-head attention. Make sure to initialize it and call it with appropriate arguments. This is a common source of bugs. Remember: the `key_padding_mask` is for keys as its name says and you need to use it. Refer to the figures above to structure the forward method.

3.2: Train the model and report your result. Performance should excess 60% on the validation set.