Computer Science 401
St. George Campus

10 February 2024
University of Toronto

Homework Assignment #2
Due: Friday, 8 March 2024 at 23h59m (11.59 PM),

Neural Machine Translation (MT) using Transformers

Email: `csc401-2024-01-a2@cs.toronto.edu`. Please prefix your subject with `[CSC401_W24-A2]`.
All accessibility and extension requests should be directed to `csc401-2024-01@cs.toronto.edu`.
Lead TAs: Julia Watson and Arvid Frydenlund.
Instructors: Gerald Penn, Sean Robertson and Raeid Saqur.

Building the Transformer from Scratch

It is quite a cliche to say that the transformer architecture has revolutionized technology. It is the building block that fuelled the recent headline innovations such as ChatGPT. Despite all those people who claimed to work in a large language model, only a very selected and brilliant few understand and are familiar with its internal workings. And you — who came from the prestigious and northern place called the University of Toronto — *must* carry on the century-old tradition to be able to create a transformer-based language model using only pen and paper.

Organization You will build the transformer model from beginning to end, and train it to do some basic but effective machine translation tasks using the Canadian Hansards data. In §1, we will guide you through the process of implementing all the **building blocks** of a transformer model. In §2, you will use the building blocks to put together the transformer **architecture**. §3 discusses the **greedy and beam search** for decoded target sentence generation. In §4, you'll **train and evaluate** the model. Finally, in §5, you'll use the trained mode to do some real machine translation and write-up your **analysis** report.

Goal By the end of this assignment, you will acquire low-level understanding of the transformer architecture and implementation techniques of the entire data processing → training → evaluation pipeline of a functioning AI application.

Starter Code Unlike A1, the starter code of this assignment is distributed through MarkUs. The training data is located at `/u/cs401/A2/data/Hansard` on `teach.cs`.

We use Python version 3.10.13 on `teach.cs`. That is, just run everything with the default `python3` command. You may need to add `srun` commands to request computational resources — please follow the instructions in the following sections to proceed. You shouldn't need to set up a new virtual environment or install any packages on `teach.cs`. You can work on the assignment on your local machine, but you must make sure that your code works on `teach.cs`. Any test cases failed due to incompatibility issues will not receive any partial marks.

Marking Scheme Please see the `A2_rubric.pdf` file for detailed breakdown of the marking scheme.

1 Transformer Building Blocks and Components [12 Marks]

Let's start from the three types of building blocks of a transformer: the layer norm, multi-head attention and the feed-forward modules (a.k.a. the MLP weights).

LayerNorm The normalization layer computes the following. Given an input representation $\mathbf{h}$, the normalization layer computes its mean μ and the standard deviation σ. Then, it outputs the normalized features.

$$\mathbf{h} \leftarrow \frac{\gamma(\mathbf{h} - \mu)}{\sigma + \varepsilon} + \beta \tag{1}$$

Using the instructions, please complete the `LayerNorm.forward` method.

FeedForwardLayer The feed-forward layer is a two-layer fully connected feed-forward network. As shown in the following equation, the input representation $\mathbf{h}$ is fed through two layers of fully connected layers. Dropout is applied after each layer, and ReLU is the activation function.

$$\begin{aligned} \mathbf{h} &\leftarrow \text{dropout}(\text{ReLU}(\mathbf{W}_1\mathbf{h} + \mathbf{b}_1)) \\ \mathbf{h} &\leftarrow \text{dropout}(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2) \end{aligned} \tag{2}$$

Using the instructions, please complete the `FeedForwardLayer.forward` method.

MultiHeadAttention Finally, you need to implement the most complicated but important component of the transformer architecture: the multi-head attention module. For the base case where there's only $H = 1$ head, the attention score is calculated using the regular cross-attention algorithm:

$$\text{dropout}(\text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}))\ \mathbf{V} \tag{3}$$

Using the instructions, please complete the `MultiHeadAttention.attention` method.

Then, you need to implement the part where the query, key and value are split into H heads, and then pass them through the regular cross-attention algorithm you have just implemented. Next, you should combine the results. Don't forget to apply the linear combination and dropout when you output the final attended representations.

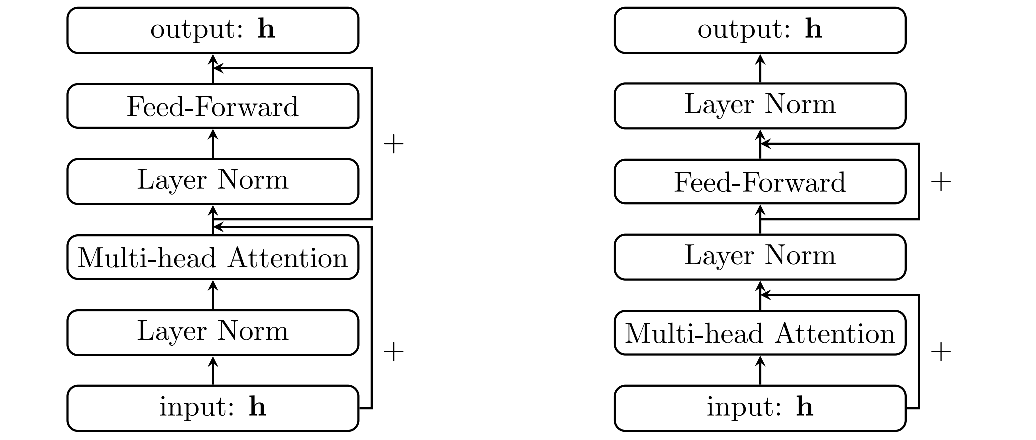
Using the instructions, please complete the `MultiHeadAttention.forward` method.

2 Putting the Architecture (Enc-Dec) Together [20 Marks]

OK, now we have the building blocks, let's put everything together and create the full transformer model. We will start from a single transformer encoder layer and a single decoder layer. Next, we build the complete encoder and the complete decoder together by stacking the layers together. Finally, we connect the encoder with the decoder and complete the final transformer encoder–decoder model.

TransformerEncoderLayer You need to implement two types of encoder layers. Pre-layer normalization (Figure 1a), as its name suggests, applies layer normalization before the representation is fed into the next module. On the other hand, post-layer normalization (Figure 1b) applies layer normalization after the representation is fed into the representation.

Using the instructions, please complete the `pre_layer_norm_forward` and the `post_layer_norm_forward` methods of the `TransformerEncoderLayer` class.



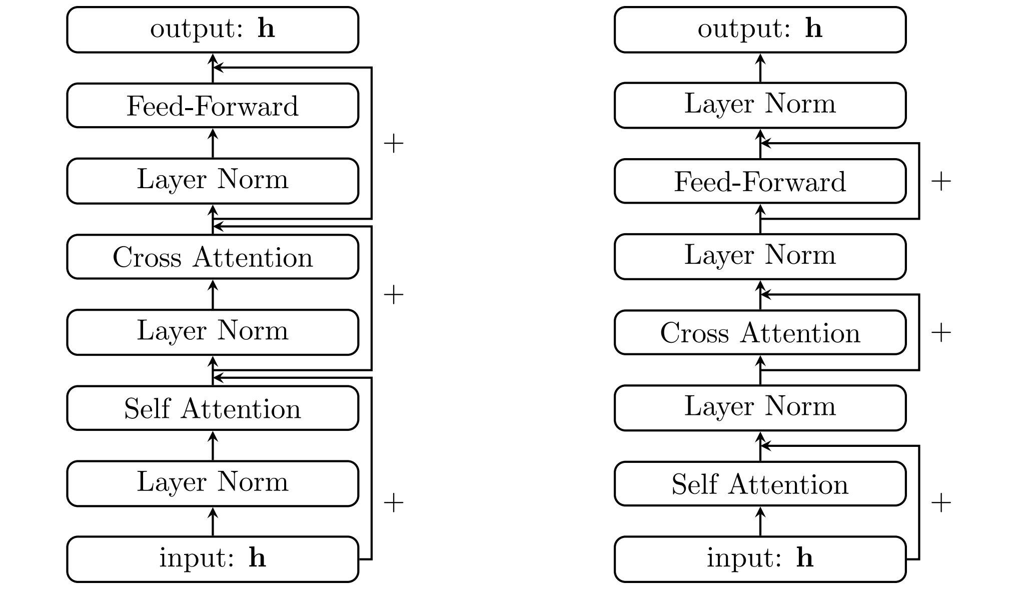
(a) Pre-layer normalization for encoders. (b) Post-layer normalization for encoders.

Figure 1: Two types of `TransformerDecoderLayer`.

TransformerEncoder You don't need to implement the encoder class. The starter code contains the implementation. Nonetheless, it will be helpful to read it, as it will a good reference for the following tasks.

TransformerDecoderLayer Again, you need to implement both pre- and post-layer normalization. Recall from lecture, there are two multi-head attention blocks for decoders. The first one is a self-attention block and the second one is a cross-attention block.

Using the instructions, please complete the `pre_layer_norm_forward` and the `post_layer_norm_forward` methods of the `TransformerDecoderLayer` class.



(a) Pre-layer normalization for decoders. (b) Post-layer normalization for decoders.

Figure 2: Two types of `TransformerEncoderLayer`.

TransformerDecoder Similar to `TransformerEncoder`, you should pass the input through all the decoder layers. Make sure to add the LayerNorm module in the correct place depending if the model is pre- or post-layer normalization. Finally, don't forget to use the logit projection module on the final output.

Using the instructions, please complete the `TransformerDecoder.forward` method.

TransformerEncoderDecoder After making sure that the encoder and the decoder are both built properly, it's time to put them together. You need to implement the following methods:

- The method `create_pad_mask` is the helper function to pad a sequence to a specific length.
- The method `create_causal_mask` is the helper function to create a causal (upper) triangular mask.
- After finishing the two helper methods, you can implement the `forward` method that connect all the dots. In particular, you first create all the appropriate masks for the inputs. Then, you feed them through the encoder. And, finally, you can get the final result by feeding everything through the decoder.

3 MT with Transformers: Greedy and Beam-Search [20 Marks]

The `a2_transformer_model.py` file contains all the required functions to complete and detailed instructions (with hints). Here we list the high-level methods/functions that you need to complete.

3.1 Greedy Decode

Let's warm up by implementing the greedy algorithm. At each decoding step, compute the (log) probability over all the possible tokens. Then, choose the output with the highest probability and repeat the process until all the sequences in the current mini-batch terminate.

Using the instructions, please complete the `TransformerEncoderDecoder.greedy_decode` method.

3.2 Beam Search

Now, it's time for perhaps the hardest part of the assignment – beam search. But don't worry, we have broken everything down into smaller, and much simpler chunks. We will guide you step by step to complete the entire algorithm.

Beam search is initiated by a call to the `TransformerEncoderDecoder.beam_search_decode` method call. Recall from the lecture that its job is to generate *partial translations* (or, *hypotheses*) from the source tokens during the decoding phase. So, the `beam_search_decode` method gets called whenever you are trying to generate decoded translations (e.g. from `TransformerRunner.[translate, compute_average_bleu_over_dataset]` etc.).

Complete the following functions in the `TransformerEncoderDecoder` class:

1. `initialize_beams_for_beam_search`: This function will initialize the beam search by taking the first decoder step and using the top-k outputs to initialize the beams.
2. `expand_encoder_for_beam_search`: Beam search will process 'batches' of size 'batch_size * k' so we need to expand the encoder outputs so that we can process the beams in parallel. (*Tip*: You should call this from within the preceding function).
3. `repeat_and_reshape_for_beam_search`: It's a relatively simple expand and reshaping function. See how it's called from the `.beam_search_decode` method and read the instructions in function comments. (*Tip*: Review Torch.Tensor.expand).
4. `pad_and_score_sequence_for_beam_search`: This function will pad the sequences with eos and seq_log_probs with corresponding zeros. It will then get the score of each sequence by summing

the log probabilities. See how it's called from the `.beam_search_decode` method and read the instructions in function comments.

5. `finalize_beams_for_beam_search`: Finally, this functions as its name - it will take a list of top beams of length batch_size, where each element is a tensor of some length and return a padded tensor of the top beams. It's the final return statement of the `.beam_search_decode` method – see how the return is consumed by the calling functions (e.g. 'translate').

4 MT with Transformers: Training and Testing [20 Marks]

4.1 Calculating BLEU scores

Modify `a2_bleu_score.py` to be able to calculate BLEU scores on single reference and candidate strings. We will be using the definition of BLEU scores from the lecture slides:

$$BLEU = BP_C \times (p_1 p_2 \ldots p_n)^{(1/n)}$$

To do this, you will need to implement the functions `grouper(...)`, `n_gram_precision(...)`, `brevity_penalty(...)`, and `BLEU_score(...)`. Make sure to **carefully** follow the doc strings of each function. Do not re-implement functionality that is clearly performed by some other function.

Your functions will operate on sequences (e.g., lists) of tokens. These tokens could be the words themselves (strings) or an integer ID corresponding to the words. Your code should be agnostic to the type of token used, though you can assume that both the reference and candidate sequences will use tokens of the same type.

Please scale the BLEU score by 100.

4.2 The Training Loop

Before you start working on this part of the assignment, have a good look at the `train` function. It describes how the training loop works. For each epoch, we first train the model using all the data from the training set. Then, we evaluate the model's performance upon the completion of the epoch. We repeat the process until we reach the specified maximum epoch number. For this part of the assignment, you will need to implement the following methods:

`train_for_epoch` The training of one epoch contains seven steps:

1. We iterate through the training dataloader to obtain the current mini-batch.
2. Then, we send the data tensors to the appropriate devices (CPU or GPU).
3. Call `train_input_target_split` to prepare the data for teacher forcing training.
4. Feed the data through the transformer model and collect the logits.
5. Compute the loss value using the loss function.
6. Call `loss.backward()` to compute the gradients.
7. Call `train_step_optimizer_and_scheduler` to update the model using the optimizer, and step the scheduler.
   Note: You should handle gradient accumulation (using the `accum_iter` parameter) for full marks.

`train_input_target_split` This method splits target tokens into input and target for maximum likelihood training (teacher forcing).

`train_step_optimizer_and_scheduler` This method steps the optimizer, zeros out the gradient, and steps scheduler.

`compute_batch_total_bleu` This function computes the total BLEU score for each n-gram level in n-gram levels over elements in a batch. **Note**: You need to clean up the sequences by removing **ALL** special tokens (`sos_idx`, `eos_idx`, `pad_idx`).

4.3 Run Model, Run!

OK, enough coding. Let's actually do some model training and deployment. In this part of the assignment, you will (1) train models with different settings, (2) evaluate those models on the hold-out test set, and (3) deploy the model and do some actual translation.

4.3.1 Training the Models

Debugging and Development Similar to A1, we provide a `--tiny-preset` option to make your model train only on a tiny sliver of the dataset. You should use this option with a smaller toy model for debugging during your development phase. Training this toy model using the CPU on `teach.cs` will take approximately 10 seconds to complete. The following command is just an example. You should modify the command to test different components of your code.

```
srun -p csc401 --pty \
    python3 a2_main.py train \
        model_test.pt \
        --tiny-preset \
        --source-lang f \
        --with-transformer \
        --encoder-num-hidden-layers 2 \
        --word-embedding-size 20 \
        --transformer-ff-size 31 \
        --with-post-layer-norm \
        --batch-size 5 \
        --gradient-accumulation 2 \
        --skip-eval 0 \
        --epochs 2 \
        --device cpu
```

The `srun -p csc401 --pty` part of the command requests a compute node on teach.cs. This will provide you with dedicated CPUs and reduce congestion during peak times. The `--pty` flag enables pseudo-terminal mode, which allows your breakpoints to work properly. However, if you want to run the assignment locally, you should remove it.

Training the Final Models with GPUs After ensuring that your code works properly, you should train your model using both the pre- and post-layer normalization settings. Each epoch of training should take approximately 2 minutes. However, this time may vary depending on GPU availability.

Note: You are not required to train the model with post-layer normalization. However, you must ensure that post-layer normalization is implemented correctly. The correctness of both pre- and post-layer normalization will have equal importance during the evaluation of the code. For the analysis and training, only the pre-layer normalization setting is required.

```
# Train the model using pre-layer-norm
srun -p csc401 --gres gpu \
    python3 -u a2_main.py train \
    transformer_model.pt \
    --device cuda
```

Then, include a printout of the the training logs in `analysis.pdf`. You have the option to use WandB for this part of the assignment, see Appendix B for more information.

4.3.2 Evaluate the Models

Now, evaluate the models using the following commands.

```
# Test the model using pre-layer-norm
srun -p csc401 --gres gpu \
    python3 -u a2_main.py test \
    transformer_model.pt \
    --device cuda
```

5 Analysis [8 Marks]

You are all done! Now, that you've completed all the parts needed for **training** your transformer end-to-end and **testing** it, you're ready to finish this assignment with some **analysis**. Report the evaluation results (from preceding section) in your latex report file: `analysis.pdf`.

Is there a difference between BLEU-3 and BLEU-4? What do you think could be the reason behind the differences? Write your answer in `analysis.pdf`.

5.1 Let's Translate Some Sentences!

In the `TransformerRunner.translate` method, you are tasked with processing a "raw" input sentence through several stages to obtain its translation. You need to (1) tokenize the sentence, (2) convert tokens into ordinal ids, (3) feed the ids into your model, and (4) finally, convert the output of the model into an actual sentence. You can load your trained model with the following command.[1]

```
srun -p csc401 --pty python3 a2_main.py interact transformer_model.pt
```

An interactive Python prompt will start and your can start translating with the function `translate`. You can then utilize the model by interacting with it in the prompt as shown.

```
Trained model from path Your Model.pt loaded as the object ‘model‘
Python 3.10.13 (main, Sep 19 2023, 12:09:15) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> translate("Mon nom est Frank.")
'<s> frank is frank frank frank </s>'
```

For this part of the assignment, you will translate a few sentences from French to English using your model, a fine-tuned pre-trained transformer model (T5 MT model or Bart MT model), and a large, established model (Google Translate or ChatGPT). First, you need to translate the following eight sentences[2] into English using your model:

[1]It's OK to run the model with the greedy decoder if you haven't completed the implementation of beam search. You won't incur any penalty for doing so. To enable greedy decoding, simply add a `--greedy` flag at the end of the command.

[2]French accent marks (é,ç,à,ô,ï) are removed in the Canadian Hansards dataset and, therefore, you should too. Use *francaise*, *Universite* and *etudiants* instead of *française*, *Université* and *étudiants*.

- Voila des mesures qui favorisent la famille canadienne.
- Je voudrais aussi signaler aux deputes qu'ils peuvent maintenant s'avancer pour voter.
- Trudeau embauche un cabinet de consultants pour examiner la dependance excessive du gouvernement a l'egard des cabinets de consultants.
- Pierre demande s'il vous plait s'il peut s'attribuer le merite d'avoir supprime 600 emplois a la SRC.
- La France a remporte la coupe du monde 2018.
- J'avais a peine de l'eau a boire pour huit jours.
- Toronto est une ville du Canada.
- Les etudiants de l'Universite de Toronto sont excellents.

Then, translate the same sentences into English using your choice of the T5 MT model or the Bart MT model. Next, translate the same eight sentences into English using your choice of Google Translate or ChatGPT.

In Section 2, "Translation Analysis," of `analysis.pdf`, list all the translations and answer the following questions.

1. Describe the **overall** quality of the three types of models. Which one is the best and which one is the worst?
2. What attributes and factors of the models do you think play a role in determining the quality?
3. Now, let's observe the quality of your model's translations for individual sentences. Which sentences does your model translate better, and which does it translate worse? Can you identify a pattern? Describe the pattern of quality across different types of sentences.
4. What about the fine-tuned pre-trained model and Google Translate/ChatGPT's quality for individual sentences? Does the previous pattern still persist? Why?

Submission Requirements

This assignment is submitted electronically via MarkUs. You should submit a total of five ("5") required files as follows:

1. Your code: `a2_transformer_model.py`, `a2_transformer_runner.py`, and `a2_bleu_score.py`.
2. Your analysis: `analysis.pdf`. N.B. a *latex* template for this file has been provided to you in the starter code as `a2_report.zip`.
3. `ID.txt`: Provide pertinent information as per the `ID.txt` template on course's website, including the statement:

   > "By submitting this file, I declare that my electronic submission is my own work, and is in accordance with the University of Toronto Code of Behaviour on Academic Matters and the Code of Student Conduct, as well as the collaboration policies of this course."

You do not need to hand in any files other than those specified above.

Appendices

A Canadian Hansards

The main corpus for this assignment comes from the official records (*Hansards*) of the 36th Canadian Parliament, including debates from both the House of Representatives and the Senate. This corpus is available at `/u/cs401/A2/data/Hansard/` and has been split into `Training/` and `Testing/` directories.

This data set consists of pairs of corresponding files (`*.e` is the English equivalent of the French `*.f`) in which every line is a sentence. Here, sentence alignment has already been performed for you. That is, the n^{th} sentence in one file corresponds to the n^{th} sentence in its corresponding file (e.g., line n in `fubar.e` is aligned with line n in `fubar.f`). Note that this data only consists of sentence pairs; many-to-one, many-to-many, and one-to-many alignments are not included.

B Visualizing and logging training with WandB

You have the option to use '*Weights and Biases*' [W&B] to visualize and log model training. Go to the W&B site[3] and sign-up, then create a new project space named: '`csc401-W24-a2`'. We refer to your W&B *username* as `$WB_USERNAME` hereafter. Then, add `--viz-wandb $WB_USERNAME` flag to your model training commands (3.3.1).

C Suggestions

C.1 Check Piazza regularly

Updates to this assignment as well as additional assistance outside tutorials will be primarily distributed via Piazza (`https://piazza.com/class/lnlv5iq4kgria`). **It is your responsibility to check Piazza regularly for updates.**

C.2 Run cluster code early and at irregular times

Because GPU resources are shared with your peers, your `srun` job may end up on a weaker machine (or even postponed until the resources are available) if too many students are training at once. To help balance resource usage over time, we recommend you finish this assignment as early as possible. You might find that your peers are more likely to run code at certain times in the day. To check how many jobs are currently queued or running on our partition, please run `squeue -p csc401`.

If you decide to run your models right before the assignment deadline, please be aware that we will be unable to request more resources or run your code sooner. **We will not grant extensions for this reason.**

[3] https://wandb.ai/