Assignment 5: Sequence-to-Sequence Learning with Transformers for Arithmetic

Course: Introduction to NLP

Deadline: 23rd April, 2025 | 23:59

1 General Instructions

1. The assignment must be implemented in Python using PyTorch.
2. While you are allowed to use PyTorch modules to aid your implementation, you should not directly use the encoder and decoder blocks from PyTorch.
3. Your grade will reflect the correctness of your implementation, the thoughtfulness and depth of your analysis, and the overall clarity of your submission.
4. This assignment should be feasible on platforms with limited GPU resources (e.g., Google Colab, Kaggle). Consider this constraint when designing your model and experiments.
5. Start early. We won't be able to grant any deadline extensions. (This time for real, because endsems start right after that).

2 Task Overview: Teaching Transformers Arithmetic

The goal of this assignment is to build, train, and analyze an Encoder-Decoder Transformer model capable of performing a basic arithmetic operations (addition and subtraction) on numbers represented as character sequences. For instance, input `"123+45"` should ideally produce output `"168"`. This involves sequence-to-sequence modeling, understanding the Transformer architecture, and critically analyzing the model's behavior.

3 Part 1: Data Generation and Preprocessing

You will need to create your own dataset for this task.

1. Develop a method to synthetically generate arithmetic problems (input strings) and their corresponding solutions (target strings). Focus on two primary operations (addition and subtraction).

2. Your generation process should allow control over factors like the length of numbers involved. Consider how to create representative training, validation and testing. Determine appropriate dataset sizes for effective training and evaluation within compute constraints.

3. Make sure to test your model on a generalization set with more number of digits than what it was trained on.

4. Ensure your generated data covers relevant edge cases for the chosen operation (e.g., carrying in addition, borrowing in subtraction).

4 Part 2: Transformer Model Implementation

Implement an Encoder-Decoder Transformer model.

1. **Architecture:** Construct the model based on the standard Transformer architecture ("Attention Is All You Need"). You are allowed to use PyTorch modules like `MultiHeadAttention` for this, however do not directly use the PyTorch Encoder and Decoder blocks. Your transformer should include:
   (a) Embedding layers and Positional Encoding.
   (b) Multi-Head Attention mechanisms (self-attention in encoder and decoder, cross-attention in decoder).
   (c) Position-wise Feed-Forward Networks.
   (d) Layer Normalization and Residual Connections.
   (e) Stacking of Encoder and Decoder layers.
   (f) An appropriate output layer for generating token probabilities.

2. **Configuration:** Choose and justify key hyperparameters for your model (e.g., embedding dimensions, number of layers, attention heads, feed-forward sizes). Your choices should be reasonable given the task complexity and computational constraints.

5 Part 3: Training

Develop the procedure to train your Transformer model.

1. **Loss Function:** Select and implement a suitable loss function for sequence generation, ensuring that padding tokens do not contribute to the loss.

2. **Training Loop:** Implement the main training loop, including forward pass, loss computation, backward pass, and optimizer steps.

3. **Evaluation:** Define and track relevant evaluation metrics during training and testing. Exact Match Accuracy (entire sequence correct) is highly recommended. Consider other metrics like character-level accuracy or perplexity (you should report 2 other metrics along with Exact Match)

4. **Inference:** Implement a method for generating output sequences from the trained model during evaluation using greedy decoding.

5. **Model Saving:** Implement logic to save and load model checkpoints, potentially based on validation performance.

6 Part 4: Analysis and Report

Analyze the performance and behavior of your trained model. This is a critical component requiring thoughtful investigation.

1. **Quantitative Performance:** Report the final performance of your model on your test set(s) using the metrics defined in Part 3.

2. **Generalization:** Investigate how well your model generalizes, for example, to inputs longer or structured differently than those seen during training. Discuss the observed behavior.

3. **Error Analysis:**

   (a) Examine incorrect predictions. Identify and categorize common types of errors.

   (b) Analyze if errors correlate with specific input characteristics (e.g., input length, presence of carries, specific digits).

4. **Ablation/Sensitivity Study:** Investigate the impact of two specific design choices (e.g., a hyperparameter, architectural components, tokenization methods, positional encoding methods, anything that you consider an appropriate ablation method) on model performance or behavior.

5. **Discussion:** Synthesize your findings. Critically discuss whether the model appears to have learned a robust arithmetic procedure. What are its limitations? How might its approach differ from human computation?

7 Submission Format

Submit a single `.zip` file named `<roll_number>_assignment5.zip`. It should contain:

1. **Source Code:** Logically organized and well-commented Python scripts or Jupyter Notebooks covering data handling, model definition, training, and analysis.

2. **Trained Model:** Saved weights of your best model. If large, provide a link to your model.

3. **Report (`report.pdf`):** A concise report detailing your approach, design choices, implementation specifics, results (including figures/tables from your analysis), and the discussion requested in Part 4. Justify significant design decisions.

4. **README (`README.md`):** Instructions on setting up the environment (e.g., dependencies), running your code (data generation, training, analysis), file structure overview, and any other relevant information for the evaluator.

8 Grading Rubric (Tentative)

Evaluation will focus on correctness, demonstrated understanding, analytical depth, and clarity.

1. Data Handling (Generation, Preprocessing, Loading) (10 points)
2. Transformer Model Implementation (Correctness, Design Choices) (20 points)
3. Training and Evaluation Implementation (Correctness, Metrics) (5 points)
4. Analysis and Report (Depth, Insight, Justification, Clarity) (15 points)
5. Viva (50 points)

9 Resources

Consult relevant resources for understanding the Transformer architecture and sequence-to-sequence modeling. Key starting points include:

1. **Attention Is All You Need (Original Paper):** https://arxiv.org/abs/1706.03762
2. **Illustrated Transformer:** https://jalammar.github.io/illustrated-transformer/
3. **The Annotated Transformer:** http://nlp.seas.harvard.edu/2018/04/03/attention.html