iNLP A5: Seq-2-Seq Learning with Transformers for Arithmetic

[Naitik Kariwal – 2023201044]

Code explanation:

1. Data Preparation

The dataset used in this project consists of synthetically generated arithmetic expressions and their corresponding results. Each input string is a simple arithmetic operation (+,-), such as "12 + 9", and the target string is the result, e.g., "21".

The key steps in data preparation include:

- **Synthetic Generation**: A custom function was used to generate a large number of arithmetic examples for both training and evaluation. This approach ensures full control over the variety and complexity of examples. Edge cases like overflow, borrowing in subtraction, zero as operand and zero as result were added (constitutes **~5%**).
- **Tokenization**: All input and output characters (including digits, operators, and special tokens like <sos>, <eos>, and <pad>) were tokenized at the character level. A vocabulary dictionary maps each unique character to an integer token.
- **Dataset and Dataloader Setup**: Each sample was converted into input and output sequences with proper start and end tokens. These sequences were padded to a fixed length.

  A total of **2,00,000 sample**s were generated with **60-20-20** split among train-dev-test sets.

---

2. Model Architecture and Major Components

The model is a standard Transformer-based sequence-to-sequence architecture, adapted for character-level arithmetic tasks. It consists of an encoder and decoder with the following components:

- **Input Embeddings**: Both the input (source) and output (target) sequences are passed through embedding layers that map token IDs to dense vectors of a fixed dimension (***d_model***).
- **Positional Encoding**: To allow the model to capture the order of tokens, sinusoidal positional encodings are added to the input embeddings. These help the model distinguish between sequences like "12 + 9" and "9 + 12".
- **Multi-Head Self-Attention**: Both encoder and decoder layers include multi-head self-attention mechanisms. The encoder uses self-attention over the input sequence, while the decoder includes both masked self-attention (to prevent attending to future tokens) and cross-attention with the encoder outputs.

- **Feedforward Network (FFN)**: Each encoder and decoder block contains a position-wise feedforward network. This increases the model's capacity to capture complex patterns.
- **Layer Normalization and Residual Connections**: These techniques are used after each major component (attention and FFN) to stabilize training and improve gradient flow.
- **Hyperparameters**: The baseline configuration includes a model dimension (***d_model***) of 256, 4 layers, 8 attention heads, a feedforward dimension (***d_ff***) of 1024, and a maximum sequence length of 20. Also ***lambda_acc*** was set at 0.5.

---

3. Loss Function

The model uses a custom loss function that combines traditional cross-entropy loss with a surrogate for character-level accuracy.

- **Cross-Entropy Loss**: This is the main loss component. It compares the predicted token probabilities with the target tokens and ignores padded positions using the ignore_index parameter.
- **Surrogate Accuracy Component**: To encourage the model to assign higher probabilities to correct tokens, we calculate the average probability mass assigned to the ground truth tokens. This is treated as a soft approximation of character accuracy.
- **Mixed Loss**: The final loss is a weighted sum of the cross-entropy loss and a penalty based on how far the surrogate accuracy is from 1. A hyperparameter (***lambda_acc***) controls the balance between these two components.

This mixed loss provides a more informative gradient, especially in early training when exact matches are rare, helping the model learn more effectively.

Results & Analysis:

1. The results from model training for 10 epochs on test and unseen (containing longer sequences) datasets were:

```
--- Final Test Results ---
Loss: 0.0024 | Perplexity: 1.00
Exact Match: 88.2% | Char Accuracy: 95.0%
```

```
--- Final Unseen Results ---
Loss: 6.1058 | Perplexity: 345.03
Exact Match: 27.0% | Char Accuracy: 33.5%
```

- On the in-distribution Test Set, we achieve 88.2% exact-match and 95.0% character-accuracy (perplexity ≈1.00).
- On the out-of-distribution Unseen Set (longer, 4–5-digit inputs), exact-match plummets to 27.0% and char-accuracy to 33.5% (perplexity ≈345).

Observations:

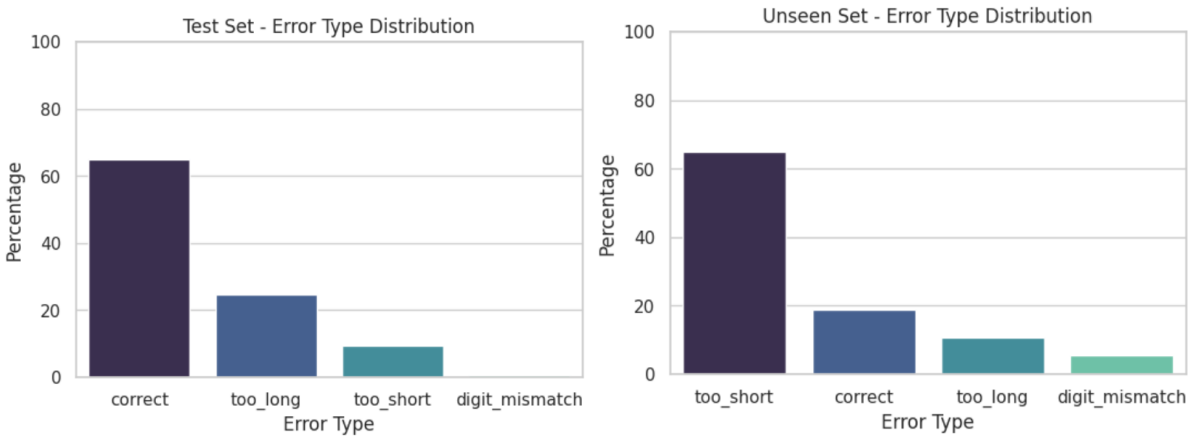
- The model has essentially memorized patterns for up to 3-digit addition/subtraction but fails to extend that to longer sequences.
- The huge spike in perplexity on Unseen suggests the model is confidently wrong—its learned token distributions don't cover the unseen cases.

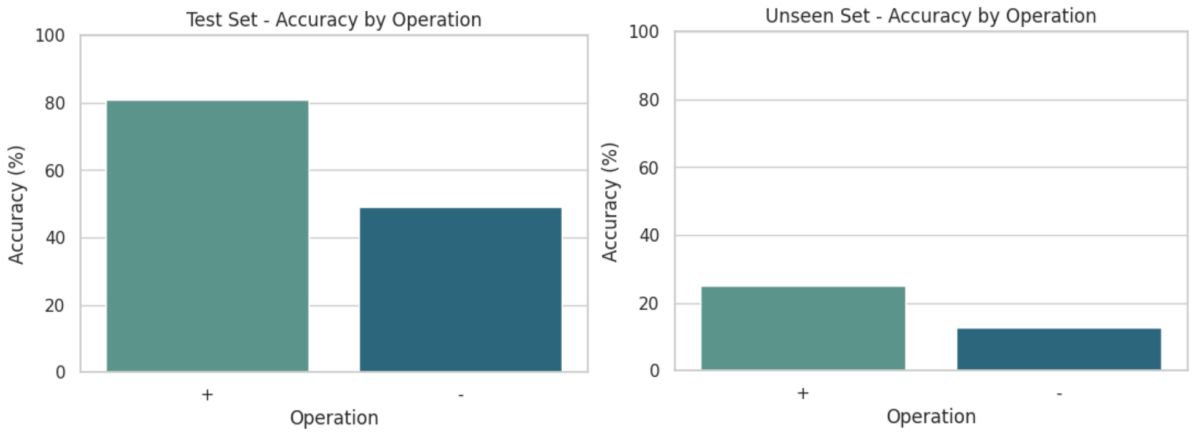
2. Following are some samples from test set and their predictions:

```
Input : 701-689
Target: 12 Pred : 1
---
Input : 420+746
Target: 1166 Pred : 116
---
Input : 864+608
Target: 1472 Pred : 147
---
Input : 565-668
Target: -103 Pred : -103
---
Input : 265-281
Target: -16 Pred : -16
---
Input : 111-580
Target: -469 Pred : -469
```

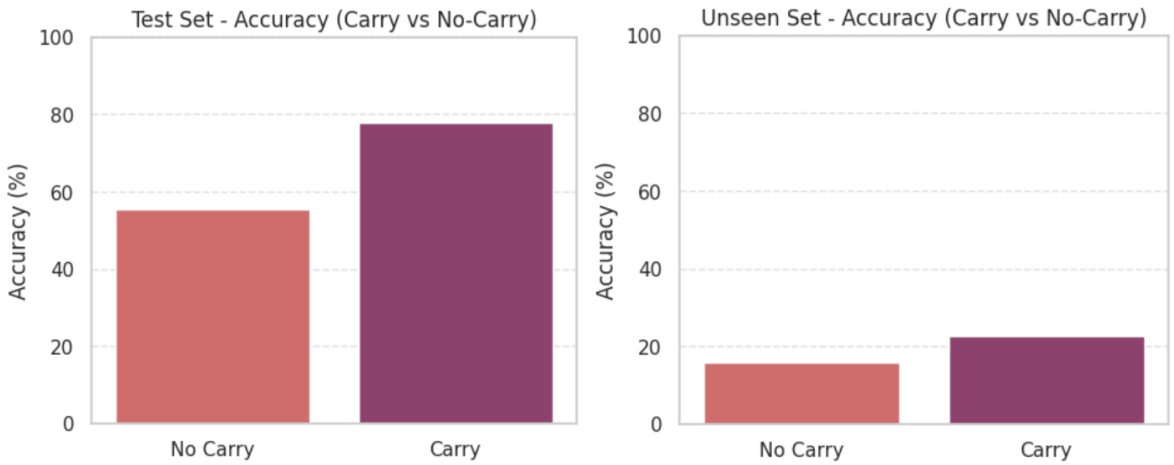
Because the Transformer was only trained on up to 3-digit examples, it never learned the iterative "carry" or "borrow" procedure in a length-agnostic way. Instead, it learned "if length = 3 and pattern X, output Y."

3. Error Analysis:

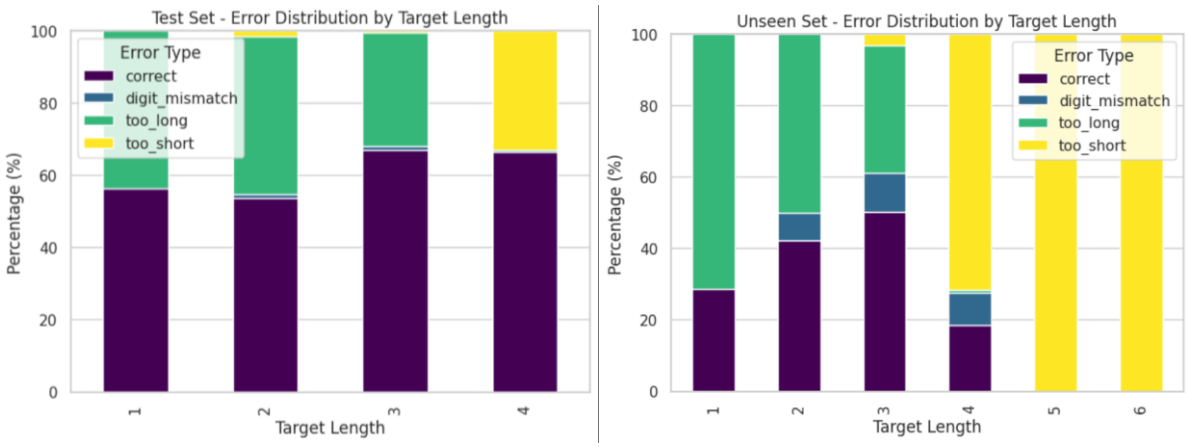




- Test: addition → 81.0% accuracy; subtraction → 49.2%
- Unseen: addition → 25.1%; subtraction → 12.8%
- *Inference:* Subtraction is systematically harder (likely because "borrow" logic is more complex than "carry").



- Test: with carry → 78.0%; without carry → 55.4%
- Unseen: with carry → 22.6%; without carry → 15.8%
- *Surprising finding:* On Test the model actually does better on carrying cases—this likely reflects that many addition examples require carry and the model has specialized on those frequent patterns.



- Test: accuracy dips for very short targets (length 1–2) because the model often outputs extra padding (too_long), and for length-4 results it often **truncates** (too_short ~32.9%).
- Unseen: performance degrades sharply as length grows—**100% too-short** for lengths 5–6, since the model never learned to emit that many digits.

4. Ablation Study:
   For ablation study, we used two distinct configurations:
   - **Fewer Heads**: {'d_model': 256, **'num_heads': 4, 'num_layers': 3**, 'd_ff': 1024}
   - **Shallower Model**: {'d_model': 256, **'num_heads': 8, 'num_layers': 1**, 'd_ff': 1024}

	Variant	ExactMatch	CharAccuracy	Perplexity
0	Fewer Heads	0.84550	0.927451	1.006913
1	Shallower Model	0.74855	0.917013	1.216301

   - **Fewer Attention Heads** (8→4, d_model reduced to 256) costs about **3.6 pp** exact-match and **2.3 pp** char-accuracy.
   - **Shallower Model** (4→1 layer) costs nearly **13.3 pp** exact-match and **3.8 pp** char-accuracy, and raises perplexity significantly.

   Inference: Depth and multi-head capacity are both critical. A single-layer decoder/encoder dramatically reduces the model's ability to track carry/borrow interactions; halving the attention heads has a milder but still meaningful impact.

[Link to Data and Model]