# Report

## 1. Model Analysis

**Model architecture:**

Embedding layer: Converts the input token id list into fixed-size vectors (embedding_dim).

LSTM layer: Uses torch.nn.LSTM.

Fully connected layer: Used for classification with num_classes output neurons.

**Loss reduction process:**

Define the loss function (cross-entropy loss).

Define the optimizer (Adam).

Training process iteration:

Pass the data into the model for prediction.

Calculate the loss between the prediction and the actual label.

Calculate gradients using the backpropagation algorithm.

Use the optimizer to update the model's parameters to minimize loss.

## 2. Dataset Analysis

Validation results: The validation results were like the training results, both improving with more training iterations.

Test results: Due to the distribution of the training and validation datasets, the model performed poorly in general cases (such as addition and subtraction of three numbers in the range 0-99).

The detailed validation and test results can be seen in the images that follow.

I designed two datasets with special distributions:

Dataset1: 50,000 samples of three single-digit (0-9) addition/subtraction + 50,000 samples of three two-digit (10-99) addition/subtraction.

Dataset2: 1,000,000 samples of three two-digit (10-99) addition/subtraction.

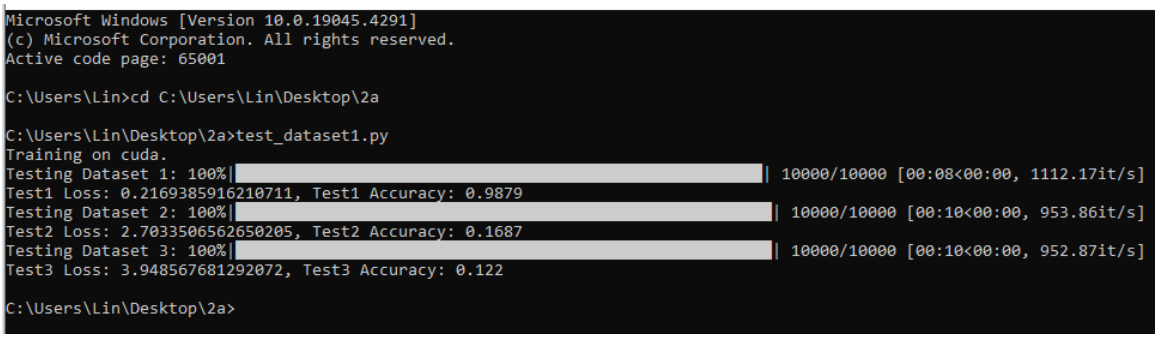
The model trained on Dataset1 was tested on test datasets 1-3:

Test dataset1: 10,000 samples of three single-digit (0-9) addition/subtraction.

Test dataset2: 10,000 samples of three two-digit (10-99) addition/subtraction.

Test dataset3: 10,000 samples of addition/subtraction of three numbers in the range 0-99.

Results are shown in the images.




I observed:

(1) The model performed well on Test dataset1 but poorly on Test datasets 2 and 3. I suspect this is because the range of numbers in three two-digit addition/subtraction and three-number range 0-99 addition/subtraction is larger than in three single-digit addition/subtraction, making the computation more complex. The model requires more training to effectively learn this complexity. If the model does not have enough time or data to adapt to this complexity, its performance may suffer in such cases.

(2) The model performed worse on Test dataset3 compared to Test dataset2. I suspect this is because the model struggled to handle addition and subtraction tasks involving both single-digit and two-digit numbers, as well as tasks involving three single-digit numbers (0-9). This is likely because the model did not encounter such scenarios during training.

(3) Even when the training accuracy is high, the model's performance may fall short of expectations in general cases (such as with Test dataset3) if the data presents a special distribution. This highlights the importance of proper data collection.

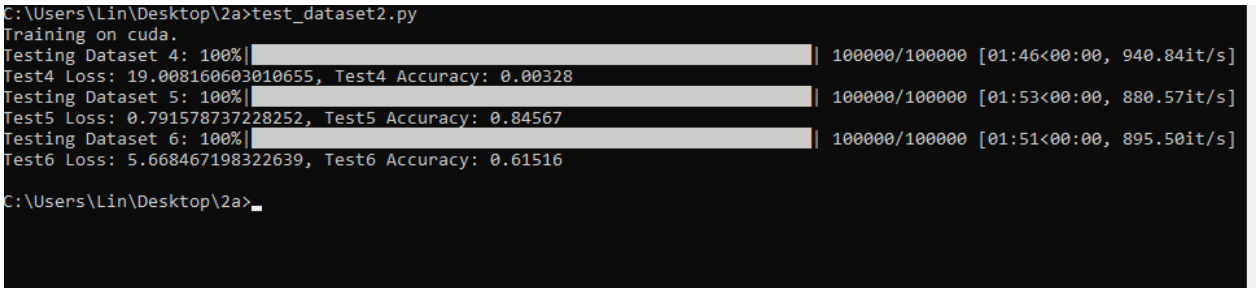
The model trained on Dataset2 was tested on test datasets 4-6:

Test dataset4: 100,000 samples of three single-digit (0-9) addition/subtraction.

Test dataset5: 100,000 samples of three two-digit (10-99) addition/subtraction.

Test dataset6: 100,000 samples of addition/subtraction of three numbers in the range 0-99.

Results are shown in the images.




I observed:

(1) The model performed poorly on Test dataset4 because the model did not encounter such situations during training. This also shows that even if the model learns the more difficult two-digit (10-99) addition/subtraction tasks, it does not guarantee good performance on simpler single-digit (0-9) addition/subtraction tasks.

(2) The model performed well on Test dataset5 because the training data also consisted of two-digit (10-99) addition/subtraction tasks. However, the model's performance on Test dataset6 was worse than on Test dataset5. This might be because the model cannot effectively handle addition/subtraction tasks that include both single-digit and two-digit numbers, as well as three single-digit numbers (0-9). The model did not encounter such situations during training.

(3) Even if the training accuracy is high, the model's performance may not meet expectations in general situations (Test dataset6). Therefore, data collection is important.

## 3. Discussion

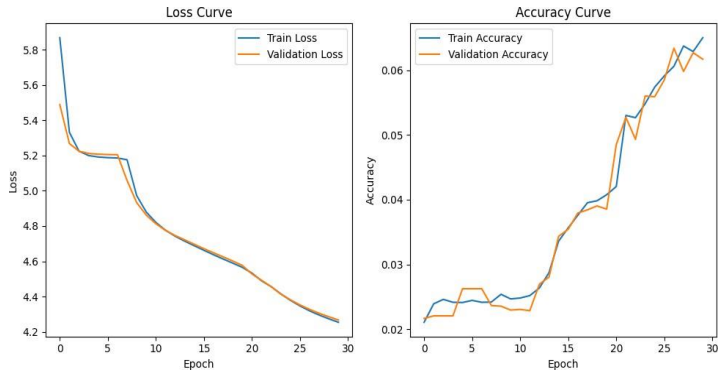
Note: The following uses Dataset1 as the train and validation dataset.

**Learning rate:**

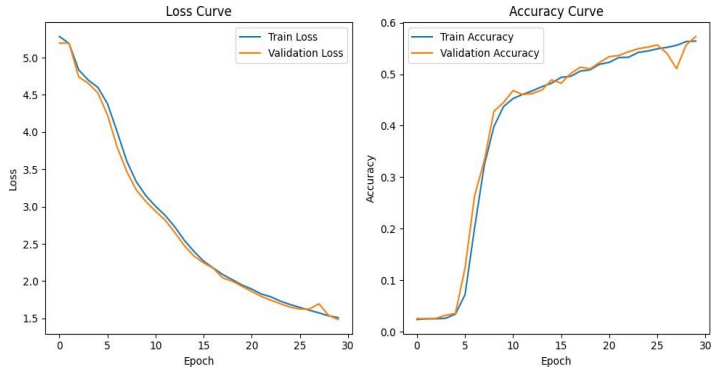
Too high learning rate: Can lead to large parameter updates, causing the model to oscillate back and forth in the parameter space (e.g., 2a_100lr) and possibly fail to converge to the optimal solution. In this case, the loss function may show significant fluctuations during the training process.

Too low learning rate: The parameter updates become very small, causing the model to converge too slowly (e.g., 2a_0.1lr) and possibly get stuck in local optima or flat regions.

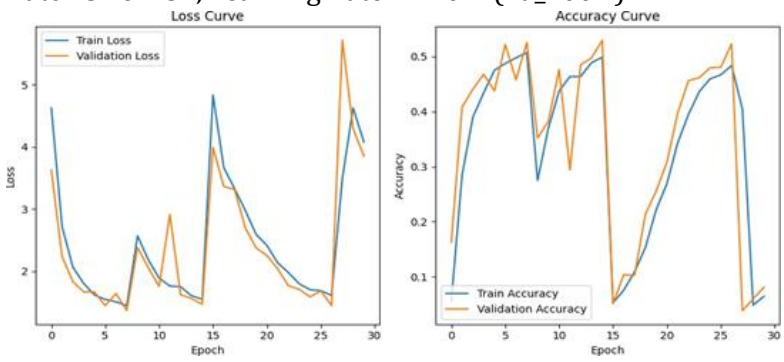
Batch Size = 32, Learning Rate = 1e-5 (2a_0.1lr)




Batch Size = 32, Learning Rate = 1e-4 (main)




Batch Size = 32, Learning Rate == 1e-2 (2a_100lr)

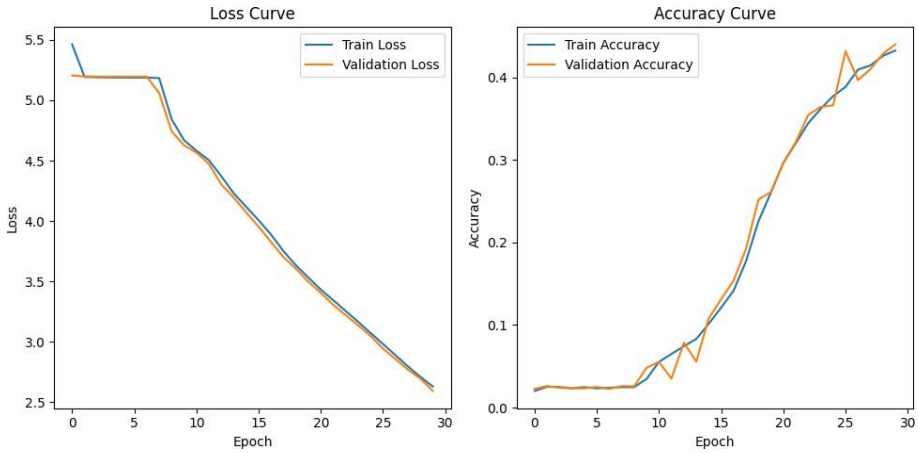



**Batch size:**

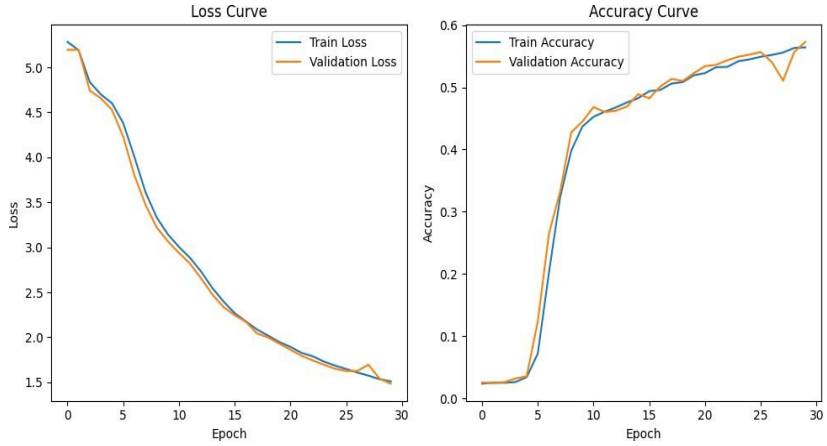
Small batch size: Slow training speed, hard to converge, better generalization ability (Small batch size introduces more randomness, helping the model escape local optima and improve generalization. However, too small a batch size may cause severe gradient oscillations, hindering convergence, as seen in 2a_quarterbs where oscillations were more significant).

Large batch size: Fast training speed, easy to converge, worse generalization ability, requires more epochs to reach the same accuracy (This might be because the number of parameter updates per epoch decreases, as seen in 2a_4BS, which likely requires more epochs to achieve accuracy above 0.5).

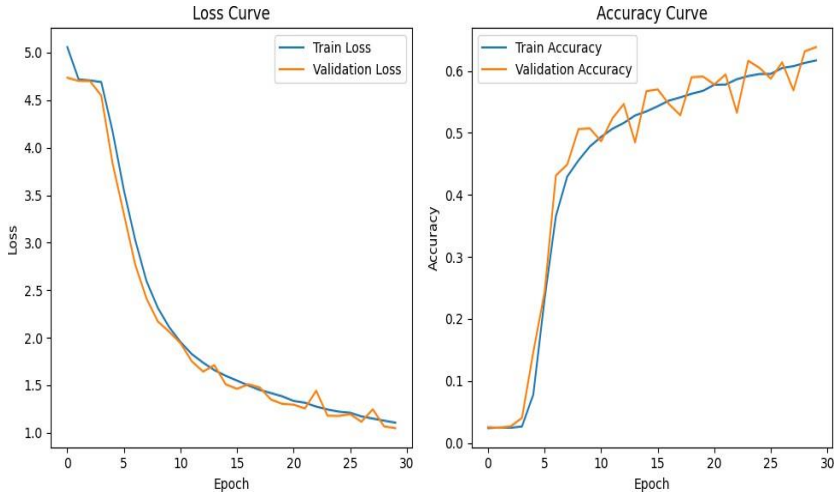
BATCH_SIZE = 128, LEARNING_RATE = 1e-4 (2a_4BS)




BATCH_SIZE = 32, LEARNING_RATE = 1e-4 (main)




BATCH_SIZE = 8, LEARNING_RATE = 1e-4 (2a_quarterbs)



I used LSTM (Long Short-Term Memory), which I believe is suitable for this task due to the following features:

(1) Memory units: LSTM has memory units that store and retrieve past information. This allows it to handle long sequences of data and maintain appropriate memory of previous information in the sequence.

(2) Gated units: LSTM uses gated units to control the behavior of the memory unit, including forget gates, input gates, and output gates. These gating mechanisms enable LSTM to learn when to forget, read, and write information.

(3) Long-term dependencies: LSTM is designed to handle long-term dependencies effectively, meaning it can remember and utilize information from far distances in the sequence, mitigating the "vanishing gradient" problem.

## 4. Bonus

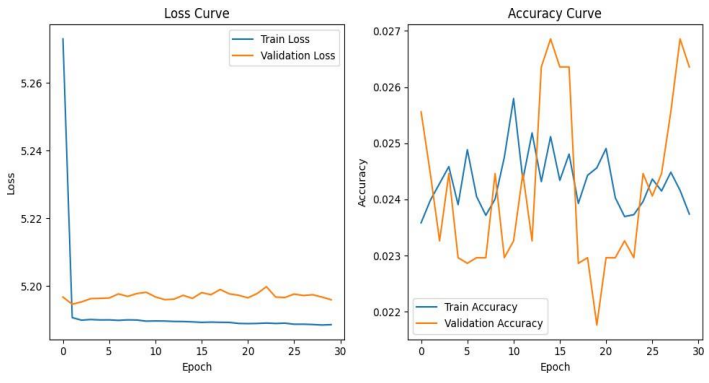
RNN: Short-term memory capability, gradient vanishing is common, fastest training speed.

GRU: Medium-term memory capability, less frequent gradient vanishing, moderate training speed.

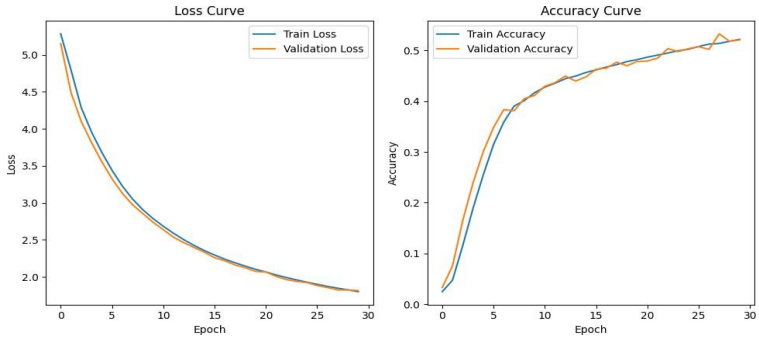
LSTM: Long-term memory capability, least gradient vanishing, slowest training speed.

The results show that with the same hyperparameter settings, the performance of GRU and LSTM is similar, but RNN performs poorly. I guess this is because RNN's memory capability is limited, so it may have difficulty learning complex relationships and may require more data to learn patterns in the dataset.

BATCH_SIZE = 32, LEARNING_RATE = 1e-4 (2a_RNN)



BATCH_SIZE = 32, LEARNING_RATE = 1e-4 (2a_GRU)




BATCH_SIZE = 32, LEARNING_RATE = 1e-4 (main)