# NLP HW2 Report

## Environment

| Running environment | Colab |
| --- | --- |
| Python version | Colab |

## Discussions

**What impact does using different learning rates have on model training?**

If too high

* Risk of overshooting optimal parameters
* Could possibly fail to converge (keep overshooting)
* Unstable training

If too low

* Converges slowly (more epochs)
* Could possibly get stuck in local minima

Hence, need to find a sweet spot -> balance between learning speed and stability

**If you use RNN or GRU instead of LSTM, what will happen to the quality of your answer generation? Why?**

RNN

* Poor ability to capture long-term dependencies
* May perform poorly when dealing with longer arithmetic sequence
* Suffers from vanishing/exploding gradients -> bad memory

GRU

* Simpler and faster architecture with fewer parameters
* May work well for relatively long sequences
* But in terms of very long arithmetic sequences, LSTM still outperforms GRU (better memory)

LSTM

* Perform better with long-term dependencies
* Have memory cells to help maintain information over many time steps
* However, LSTM have high computational and memory cost

Hence, in terms of dealing with long and complex arithmetic sequences, the LSTM model is in my opinion the most suitable.

**If we construct an evaluation set using three-digit numbers while the training set is constructed from two-digit numbers, what will happen to the quality of your answer generation?**

The model may fail to carry numbers correctly when doing arithmetic as it has never seen and understands larger number patterns, hence possibly lowering accuracy.

**If some numbers never appear in your training data, what will happen to your answer generation?**

* If certain numbers are unseen, it will have random/untrained embedding -> high error
* And LSTM will propagate this error -> higher error
* Or model may substitute unseen digit with most similar seen digit
* For example: If '7' is unseen, might output "12+6=18" or "12+8=20"

**Why do we need gradient clipping during training?**

* LSTMs are likely to have gradient explosion
* Long and complex arithmetic sequences can lead to unstable updates
* Gradient clipping can help prevent this and also improves convergence

## Insights

**Two LSTM layers**

* Two LSTM layers to extract features
* First layer might learn basic digit patterns
* Second layer likely learns arithmetic operations
* More layers may be unnecessary

**Character-level processing**

* In this homework, we split the data into characters (e.g. ‘123’ -> ‘1’, ‘2’, ‘3’)
* This is method is simple and handles numbers naturally
* However, it may lack understanding of larger numbers
* Poor understanding of number inheritance and needs to learn place value system
* Could probably split data in a higher level to better handle large numbers

**Training Speed/Performance**

* Switch to suitable device at at model training stage with model.to(device) and data.to(device) is important for leveraging hardware acceleration for speedup
* GPU is more capable of dealing with parallel processing tasks such as handling matrix operations in model training part
* model.train() mode: uses dropout and batch normalization to improve model robustness and convergence
* model.eval() mode: disable dropout, batch normalization and gradient computing. This saves computational resources for faster inference

## References

<https://pytorch.org/docs/stable/index.html>

<https://pytorch.org/tutorials/>

<https://pytorch.org/tutorials/beginner/basics/data_tutorial.html>

<https://stackoverflow.com/questions/63061779/pytorch-when-do-i-need-to-use-todevice-on-a-model-or-tensor>

<https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html#torch.nn.CrossEntropyLoss>

Use of ChatGPT for python syntaxes and pytorch library