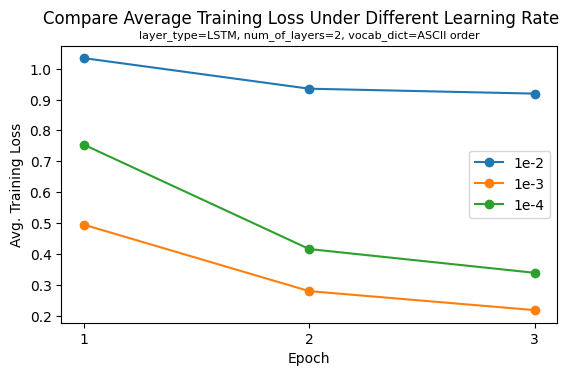
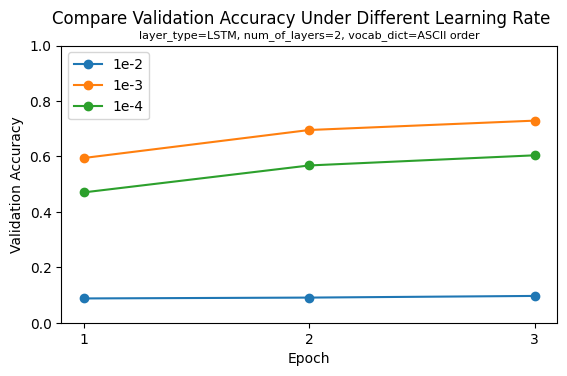
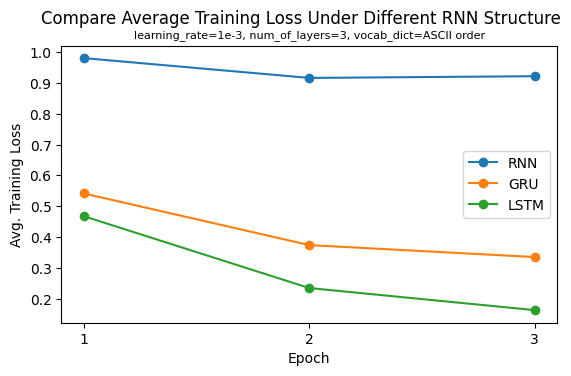
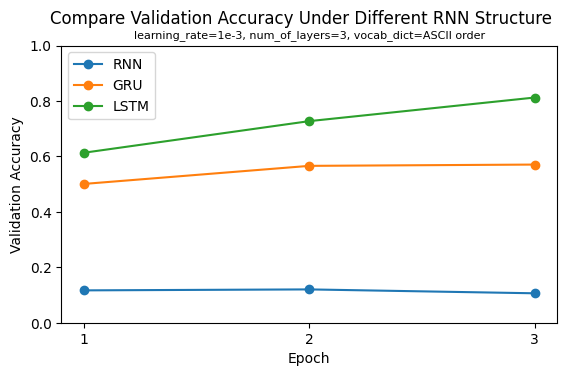
**11310CS563100 NLP Assignment 2: Arithmetic as a Language Report**

NTHU 110020007 施淙綸

* What impact does using different learning rates have on model training?
  + 一般而言，learning rate越高代表學習的速度越快速，反之。但實際上會遇到過高的learning rate導致模型無法正常收斂，在多項式「山谷」兩側反覆跳動而無法到達「谷底」。而過低的learning rate雖然能正常到達谷底，但抵達的時間會隨著降低的learning rate越來越長。作業使用的Adam optimize其實就有針對learning rate做一定程度隨iters增加而衰減，並提供momentum的能力，避免落在local answer而非global answer。
  + 為了在這份作業的情況下回答這個問題，我紀錄並觀察learning rate分別等於1e-2、1e-3及1e-4時，模型的訓練情況與表現：



* + 由上圖可知，在最多三代的情況下，越小的learning rate未必提供最大的Accuracy和Average Loss。因此，找到最恰當的learning rate很重要，在這個訓練任務中，1e-3是一個合適的數值，從圖中可以發現該數值能得到最高的Validation Accuracy和最低的Training Loss。至於抵達谷底的時間在此大致正比於Epoch數量，也就是說理論上學習率1e-4可能在更多代後達到和1e-3相同或更佳的效果，但由於時間考量沒有特別蒐集數據。
* If you use RNN or GRU instead of LSTM, what will happen to the quality of your answer generation? Why?
  + 相比傳統RNN，由於LSTM在提供額外結構上優化（像是forget gates、input gates與output gates等），讓LSTM能夠提供更佳的前後關係擷取能力。因此，LSTM相比RNN理論上能夠提供更佳的模型效果，且一定程度上避免梯度消失的情況發生。
  + 而GRU則是基於LSTM的架構，刪減了部分gate來減少訓練時消耗的運算資源，讓訓練時長相對較短，同時保留避免梯度消失的特性。因此，GRU相比LSTM準確率理論上相對較低，且訓練時間較短；相比RNN，能提供更佳的模型效果。[[1]](#footnote-1)
  + 為了驗證在這個任務中模型表現是否和理論相同，我紀錄並觀察在其餘變量相同時，使用RNN、GRU和LSTM時，模型的訓練情況與表現：



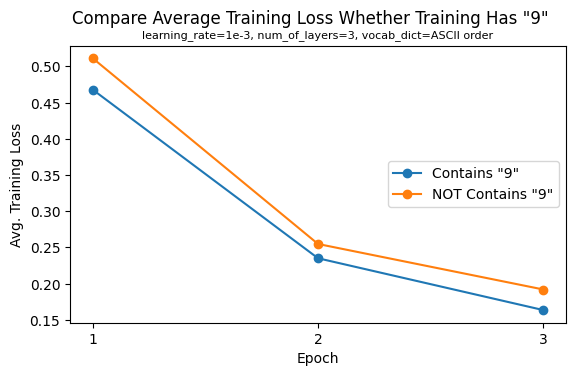
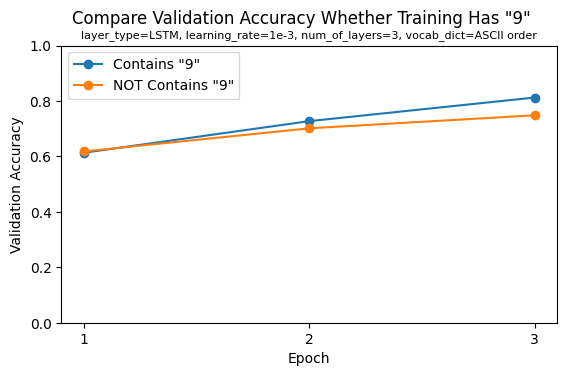
* + 由上圖可知，三者在Validation Accuracy和Average Training Loss的表現都符合先前的預期，也就是LSTM優於GRU優於RNN。至於總共訓練三代，平均一代訓練時間紀錄在下表，整體時長趨勢符合預期（LSTM > GRU > RNN）。但由於訓練時並非只進行訓練任務，同時有在使用電腦的其他功能（如看影片、瀏覽網頁和玩遊戲等），故數據僅供參考：

|  |  |  |  |
| --- | --- | --- | --- |
|  | RNN | GRU | LSTM |
| 平均一代訓練時長 | 7分55秒 | 9分54秒 | 10分42秒 |

* 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?
  + 由於沒有使用三位數進行訓練，可以預期的表現不會很好。因為對於模型而言，他學習到的pattern中並沒有三位數的情況，回答的長度和處理運算會以一、二位數的樣態為主。由於時間關係，只有進行推測而沒有生成大量的數據測試，但手動觀察幾個結果如下：

|  |  |  |  |
| --- | --- | --- | --- |
| Epoch=3, layer\_type=LSTM, num\_of\_layer=3, learning\_rate=1e-3, vocab\_dict=ASCII | | | |
| ID | Input | Output | Expected |
| 1 | 123+456= | 81 | 579 |
| 2 | 123\*456= | 7140 | 56088 |
| 3 | 333\*333= | 12799 | 110899 |
| 4 | 984\*232= | 636 | 228288 |
| 5 | (9+232)\*500= | 1400 | 120500 |
| 6 | (9-232)\*500= | -1000 | -111500 |
| 7 | (22+68)\*900= | 420 | 81000 |
| 8 | (22-68)\*900= | -160 | 41400 |
| 9 | (9+232)\*(22+68)= | 1485 | 21690 |
| 10 | (9+232)\*(22-68)= | -279 | -11086 |
| 11 | (9-232)\*(22+68)= | 147 | -20070 |
| 12 | (9-232)\*(22-68)= | -465 | 10258 |

* + 從上表可以發現一些有趣的觀察，可以猜測一些模型學習到的pattern：
    - ID 1~12：模型輸出 <eos> token的時刻會和兩位數能夠算出來的相似，也就是傾向比較短的答案。
    - ID 5~8：模型在遇到乘00結尾的算式時，輸出後方都有一些0，或許這就是他學到「乘以有幾個0結尾的數字，答案結尾也會有0」了。
    - ID 5,6 & ID 7,8：兩個例子都顯示，如果前面運算出現較小的數字減去較大的數字，也就是輸入負號的出現都會反映在輸出的負號。
    - ID 9~12：承上，但這個例子中，是否出現負號的正確率僅有50%，因此猜測模型確實有學到什麼時候輸出可能會出現負號，但並不是都是正確的。也不確定是否是因為三位數字影響到模型的回答。
  + 如果繼續觀察應該還能發現不少模型的行為規律，但礙於時間、篇幅只能僅止於此。
* If some numbers never appear in your training data, what will happen to your answer generation?
  + 和上一題的情況相似，沒有出現過的資料型態會大幅影響到模型的準確率。因此可以合理預期有數字從來沒有出現在訓練集中，訓練出來的模型在相關的回答不會太好。我訓練一個從來沒看過 “9” 的模型，並觀察Validation包含 “9” 的所有題目表現：



* + 由上圖可知，除了第一代的Validation Accuracy沒看過 “9” 的比較高以外，其他不論在Validation Accuracy和Average Training Loss看過的都表現的比沒看過的好。而隨著代數提高，兩者在Validation Accuracy的差距越來越大，可以猜測兩個模型訓練的方向已經漸漸不同了。由下表的數據可以進一步分析：

|  |  |  |  |
| --- | --- | --- | --- |
| Epoch=3, layer\_type=LSTM, num\_of\_layer=3, learning\_rate=1e-3, vocab\_dict=ASCII | | | |
|  | | Training including “9” | Training excluding “9” |
| Validation including “9” | # matched | 76964 | 739 |
| Total | 101460 | 101460 |
| **Accuracy** | **84.603%** | **0.728%** |
| Validation excluding “9” | # matched | 136879 | 121069 |
| Total | 161790 | 161790 |
| **Accuracy** | **75.856%** | **74.831%** |

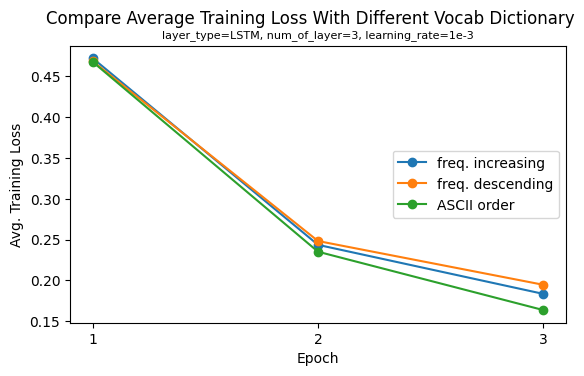
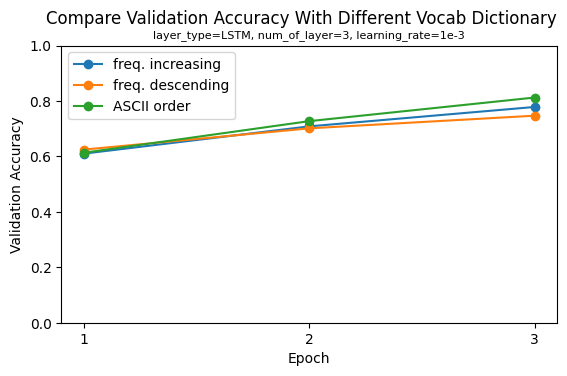
* + 由上表可以發現，沒看過 “9” 的模型在有包含 “9” 的Validation Data表現極差，而在沒有 “9” 的Validation Data表現和有看過的模型相近。比較意外的是，有看過 “9” 的模型在有“9” 的Validation Data表現特別好，不知道確切原因，也有可能是Dataset的影響。
* Why do we need gradient clipping during training?
  + 由於RNN和LSTM等模型的特性，如果沒有gradient clipping時常會發生梯度爆炸的情況，導致梯度在某些情況下不穩定、更新時爆衝等問題[[2]](#footnote-2) [[3]](#footnote-3)。因此，為了讓模型訓練過程更佳穩定，加入gradient clipping可以將梯度控制在一定範圍內，能減少一定量的訓練時長與訓練錯誤終止的機會[[4]](#footnote-4)。LSTM解決梯度消失的問題，而gradient clipping解決了梯度爆炸的問題，至此，常見不良的訓練情況已經被解決了。
* 不同的LSTM層數，對於模型表現是否有影響？
  + 一般而言，層數越多，模型更能夠學習到該語言的pattern，模型的表現應該越好。但層數過高有時候會過度學習training data的內容，導致overfit發生。我紀錄並觀察在其餘變量相同時，使用2、3和4層LSTM疊加時，模型的訓練情況與表現：

一張含有 文字, 螢幕擷取畫面, 行, 圖表 的圖片

自動產生的描述一張含有 文字, 螢幕擷取畫面, 行, 圖表 的圖片

自動產生的描述

* + 從上圖可以發現，疊加越多層LSTM，模型訓練出來的表現越好，目前疊加至四層都能穩定的增加Validation Accuracy，代表現在還沒有overfit發生，理論上還可以在增加更多層直到模型表現下降為止。
* 不同的Vocabulary Dictionary建立的方式，對於模型表現是否有影響？
  + 我一開始挑選用何種方式建立 Vocabulary Dictionary時，覺得依照ASCII的順序比較單純，在encode的情況下還保有一定的可讀性（因為數字都按順序），所以簡單使用各token的ASCII大小來建立字典。但後來查詢其他資料時發現，有些人依照token出現頻率高低來建立字典，於是我紀錄並觀察在其餘變量相同時，依照出現頻率高低或ASCII順序建立字典，模型的訓練情況與表現：



* + 由上圖可以發現，單純的使用ASCII建立字典表現的最好。但我認為這不應該造成太大影響，這裡觀察到的結果應該只能當作參考，是否單純只是巧合就需要其他資料，礙於作業性質就不再做更多討論。
* 遭遇到的困難 & Reflections
  + 在剛開始的時候，因為沒有太多使用Pytorch的經驗，所以在理解各部件的交互效果花了很多時間，尤其是在高維度的情況之下更為複雜。印象最深的是計算cross entropy的時候因為logits的維度和需要的剛好顛倒，在知道需要transpose之後仍然來回輸出很多次，最後才慢慢理解整體的作用方式。
  + 把所有TODOs其餘部分寫完其實沒有花太多時間，實際跑的時候發現loss會異常的降低，根據以前的經驗可能會是learning rate或input的問題，但試驗多次之後還是沒有改善，來回修改超久還是沒有頭緒，最後求助助教才弄懂label和data的關係。還好也算因禍得福，對整體的機制了解程度比先前深了許多，也驚訝RNN和LSTM的力量。之後的流程就很順利的跑完所有需要的數據，相比上次作業，這次訓練時長短了很多，能夠做更多的嘗試，至於後來在繪製、整理這些數據的圖也是很繁瑣的過程。
  + 為了避免等待時間過久和未知的斷線問題，我使用自己的電腦進行訓練，而非Google Colab，訓練環境如下：

|  |  |  |  |
| --- | --- | --- | --- |
|  | System | CPU | Python version |
| Running environment | Windows 11 Version 23H2 (KB5044285) | 11th Gen Intel® Core™ i7-11370H @ 3.30GHz | CPython 3.11.1 |

※註：本作業部分程式碼使用ChatGPT-4o生成，部分程式碼使用GitHub Copilot協助撰寫，區塊皆於檔案中註解。

1. Rice Yang (2022, Dec 8). *RNN, LSTM, GRU之間的原理與差異, 關於 Recurrent Neural Networks 的小小筆記.* Medium. <https://u9534056.medium.com/rnn-lstm-gru%E4%B9%8B%E9%96%93%E7%9A%84%E5%8E%9F%E7%90%86%E8%88%87%E5%B7%AE%E7%95%B0-23eba88afa1e> [↑](#footnote-ref-1)
2. ZacharyGz (2024, Jan 5). *梯度裁剪（Gradient Clipping）.* CSDN. <https://blog.csdn.net/ZacharyGz/article/details/135410610> [↑](#footnote-ref-2)
3. Aayush Bajaj (2024, Sep 13). *Understanding Gradient Clipping (and How It Can Fix Exploding Gradients Problem).* neptune.ai. <https://neptune.ai/blog/understanding-gradient-clipping-and-how-it-can-fix-exploding-gradients-problem> [↑](#footnote-ref-3)
4. Open Chat (2024, Jan 7). *梯度裁剪与其他剪裁技术的结合：更高效的模型优化.* 稀土掘金. <https://juejin.cn/post/7321047444476755995> [↑](#footnote-ref-4)