**2024 NTHU Natural Language Processing Assignment 2: Arithmetic as a language NTU** r12631055

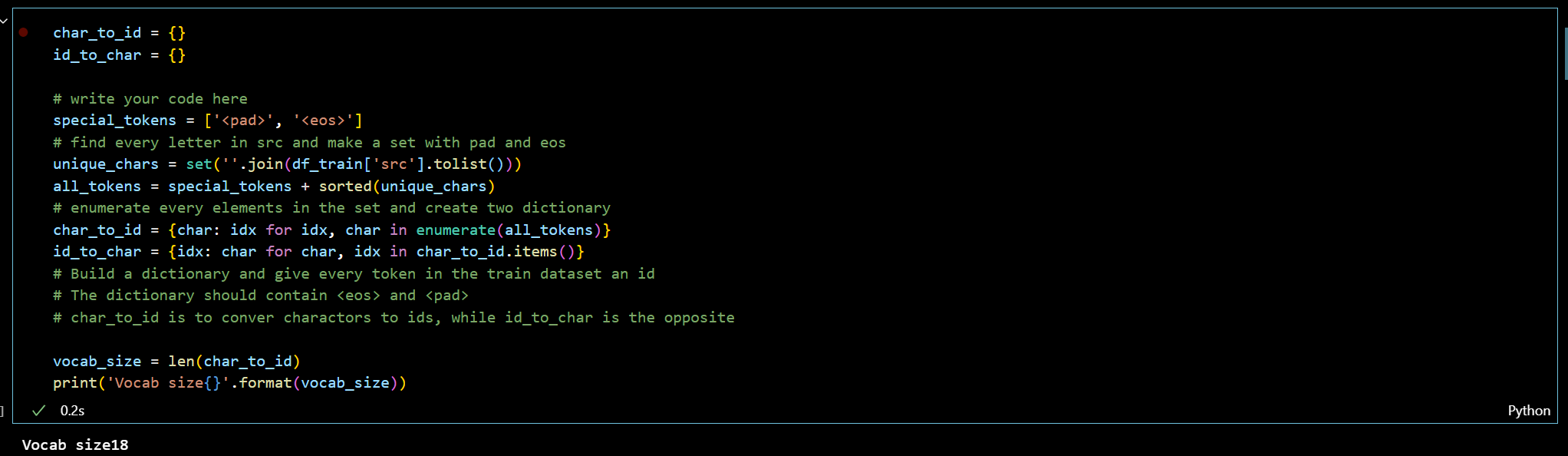
Running environment : CPU：AMD Ryzen 5955WX

GPU：NVIDIA GeForce RTX 4090

System：Ubuntu 22.04.3 LTS

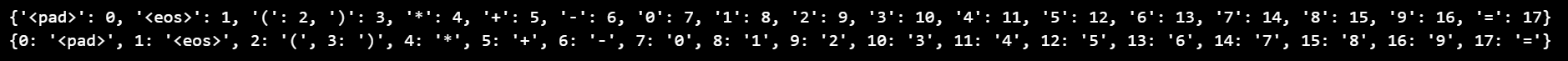
Python version : Python 3.10.12

**TODO1**



根據raw data格式以及要求的輸出dictionary格式來設計程式：

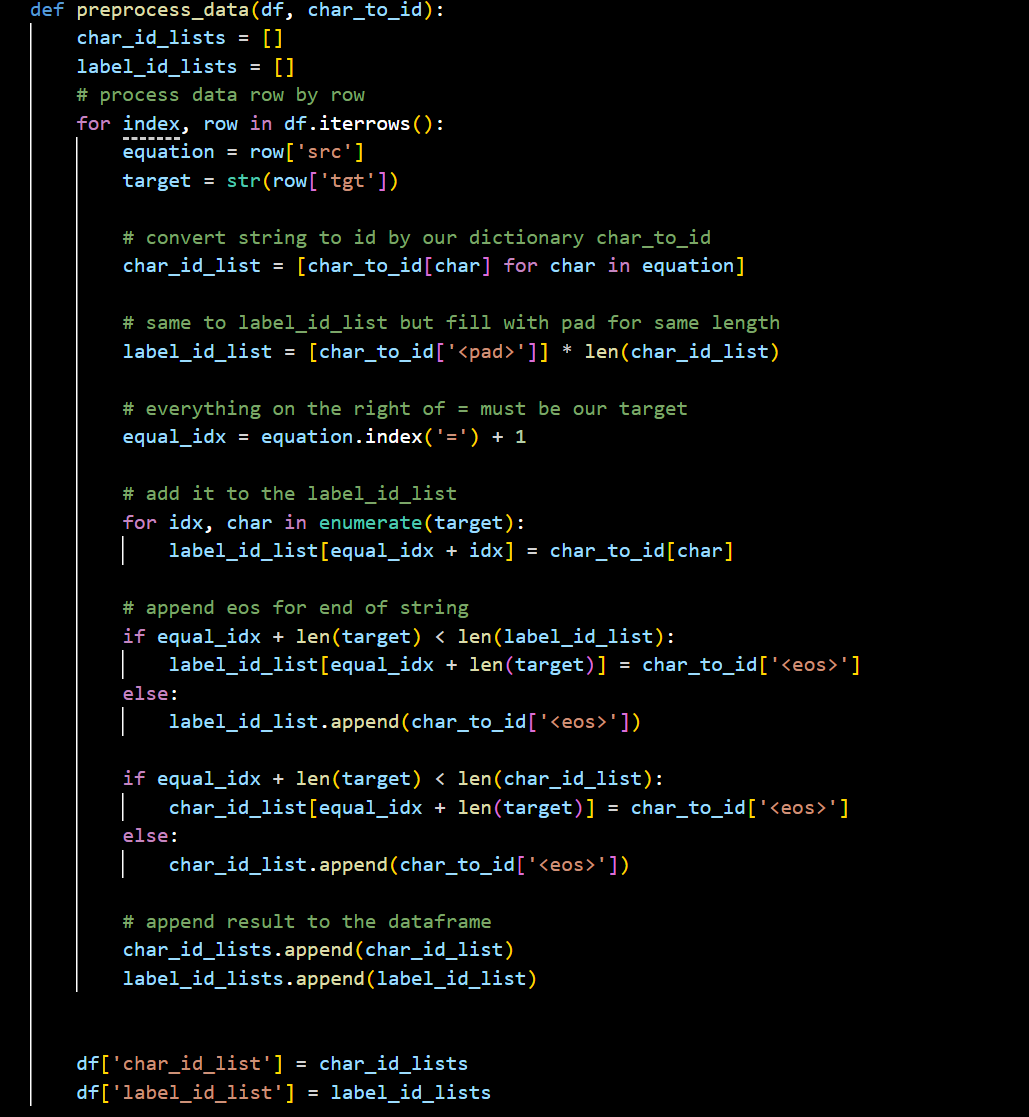
首先將特殊字元<pad>, <eos>建成集合，然後用set()找出src裡面的所有字元後合併成新的集合，至於id\_to\_char則只需將key和value值倒過來即可建立。



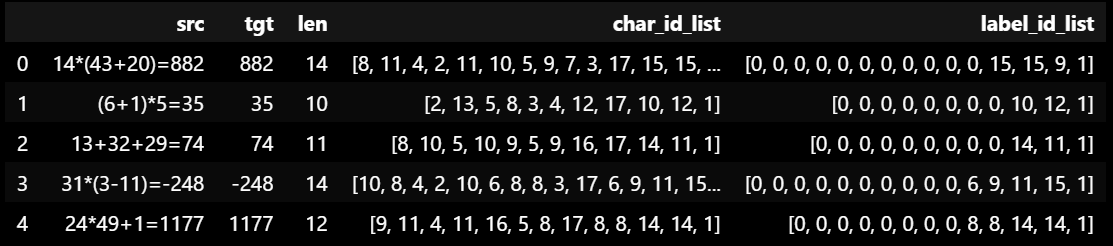
最終結果如上，經過多次的反覆嘗試，發現有經過sorted基本上編成順序都相同，等號’=’通常總是會落在最後一位：17的地方。這點可能和標準答案不太一樣。

**TODO2**

接下來要進行Data Pre-processing，我們需要把Data給分割成Input和Ground Truth(Label)，並且將Label的大小調整和Input相同，並且將等號前的數值全都換成padding：

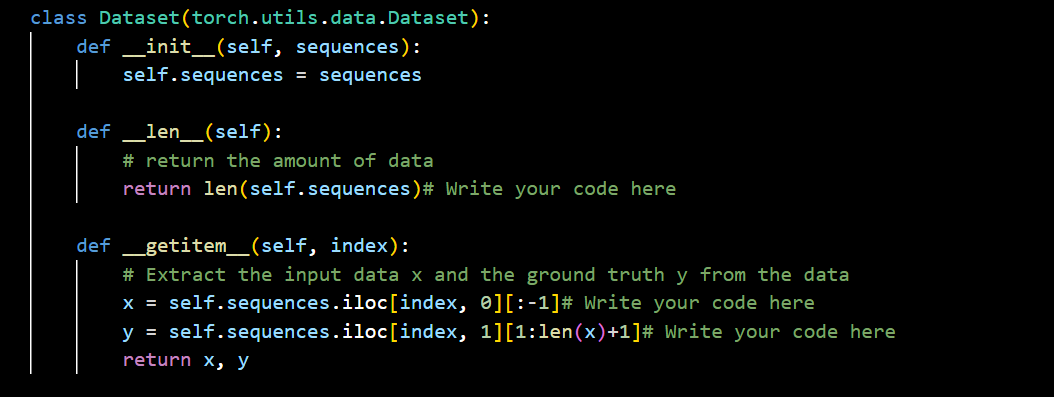


過程如註解所示：先把字串轉成ID，Label的製作也相同，只是加上padding：先找出等號位置在加工，最後在結尾全部加上end of string標記。執行時間大約1分鐘左右，最終結果如下：



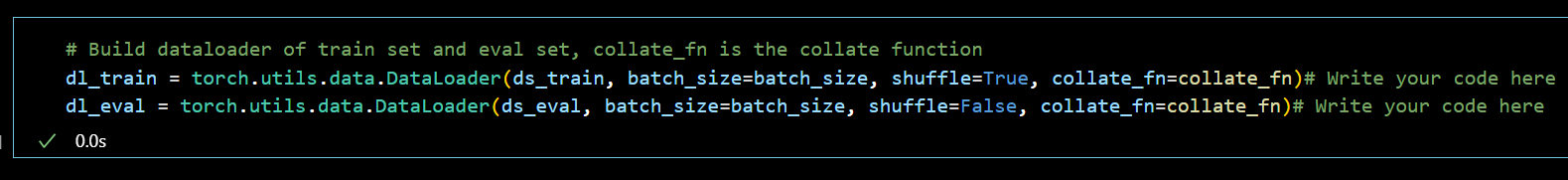
**TODO3**

再來要進行Data Batching，主要是在完成\_\_len\_\_和\_\_getitem\_\_：



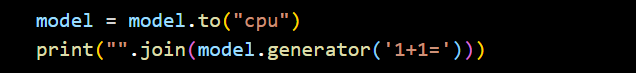
\_\_len\_\_比較容易，對input進行長度測量，至於\_\_getitem\_\_則比較麻煩，在課程內容有提到，我們是要建立一個 Language Model，其最佳化的型式是透過自迴歸，所以Input不需要最後一格並且Output需要經過向右偏移一格這樣處理流程下，因此將在Input不取最後一格的x設為長度基準，將y取到與x相同長度。

接下來在變數名稱的提示下，按照DataLoader的Document來完成各項參數，準備好要輸入進模型的Data：

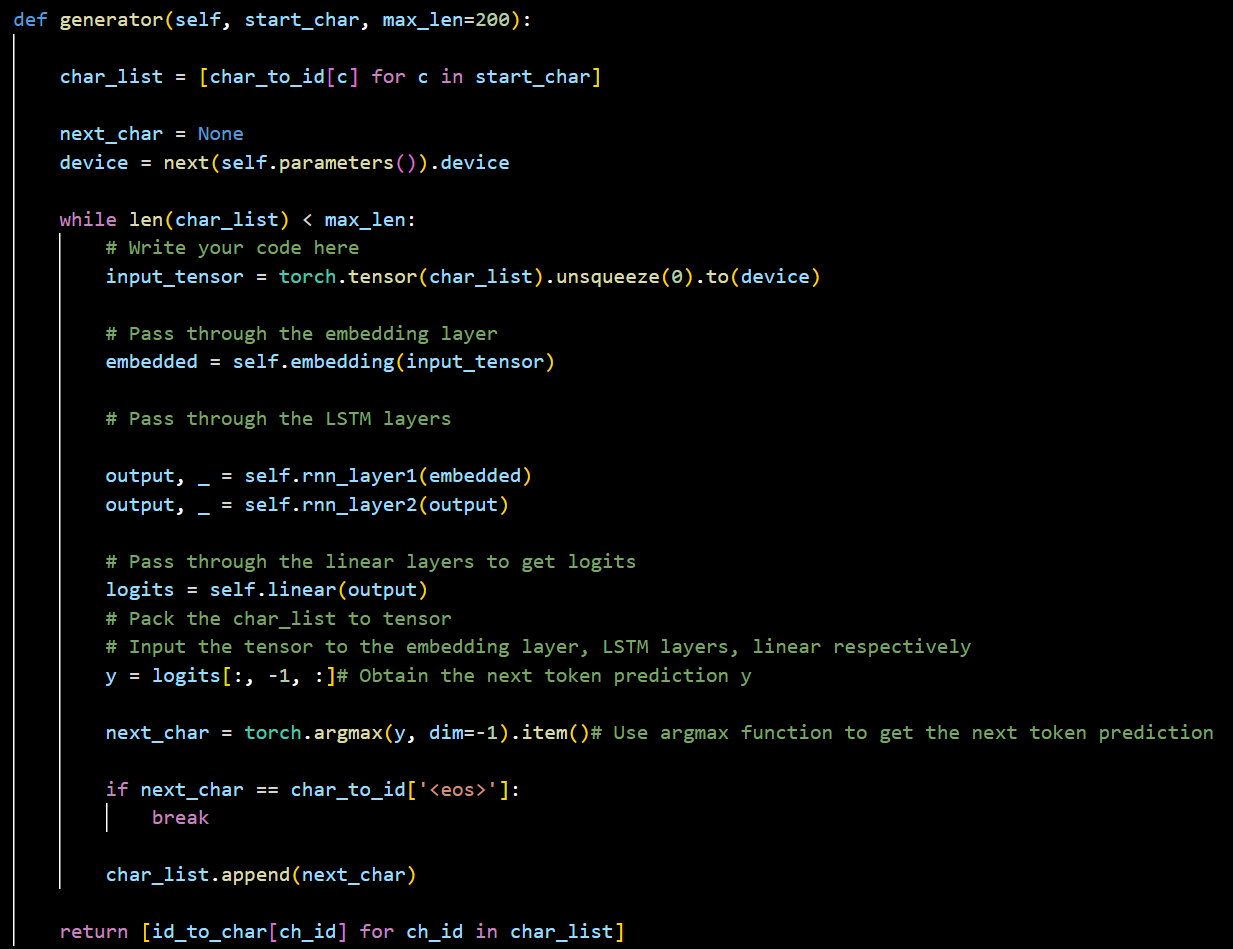


**TODO4**

接下來要準備好LSTM的模型，在這裡我們主要是要完成generator的部分，首先觀察到程式範例的最後：



由上圖可以知道，generator 的輸入為類似 "1+1="的字串，因此會需要將此字串轉為先前字典中的id：



接著根據範例的提示，似乎是要用個while迴圈來控制生成長度。

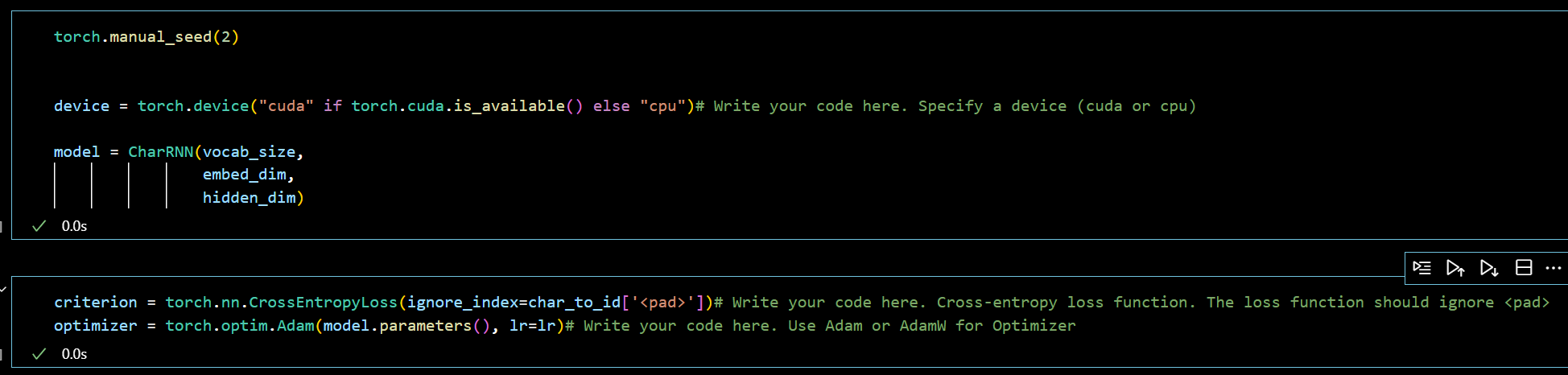
緊接著根據註解上的指示，需要將Input包裝成tensor，並且要陸續通過後續各層，其中unsqueeze的目標是將shape（維度）從(length,) 變為 (1, length)，而因為我們有使用GPU，所以多加了device。之後傳給embedding層，經過兩層LSTM後到全連接層（Linear），

最後得到每一步的預測分布，我們在這邊只採最後一步，最後使用argmax，透過函數的最大值尋找該目標參數，dim=-1表示在最後一個維度（即vocab\_size維度，在我們這次資料集中是18）上找到最大值的參數，從而確定哪個字符的預測分數最高。最後將之append到char\_list便大功告成，而最終遇到eos便會停下。

在device的部分，由於我們有實驗室主機的GPU，但由於怕資源被搶占，所以還是列了CPU做備案，在計算Loss的部分，由於本作業的任務類型是分類（Classification）任務，因此使用Cross Entropy來計算。同時因為在訓練資料中有大量占用長度的padding，因此在當遇到padding時要忽略而不計分。

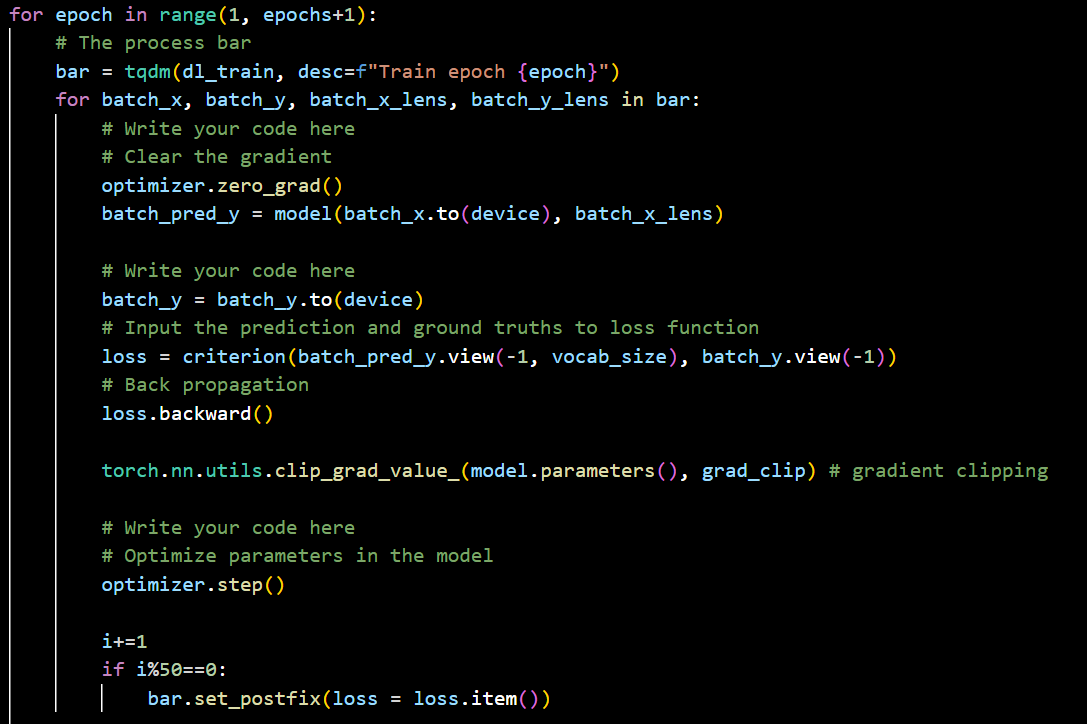
最後在Optimizer的部分則選擇簡單的Adam；其中各項參數值基本按照範例提供的參數值，或者是直接參照官方Document來填寫。

而最終結果如下：

****

**TODO5**

最後來到訓練與評估的階段，首先要先完成Training的部分，按照註解所提示，首先將Loop中的梯度歸零，然後將訓練完的預測放入batch\_pred\_y，接著按照上方的提示與batch\_y進行比較並計算Cross Entropy，然後透過反向傳播傳給模型參數，其中，需要進行一下梯度裁剪來防止梯度爆炸問題。最後再透過Optimizer對模型參數進行最佳化，最終程式碼如下：

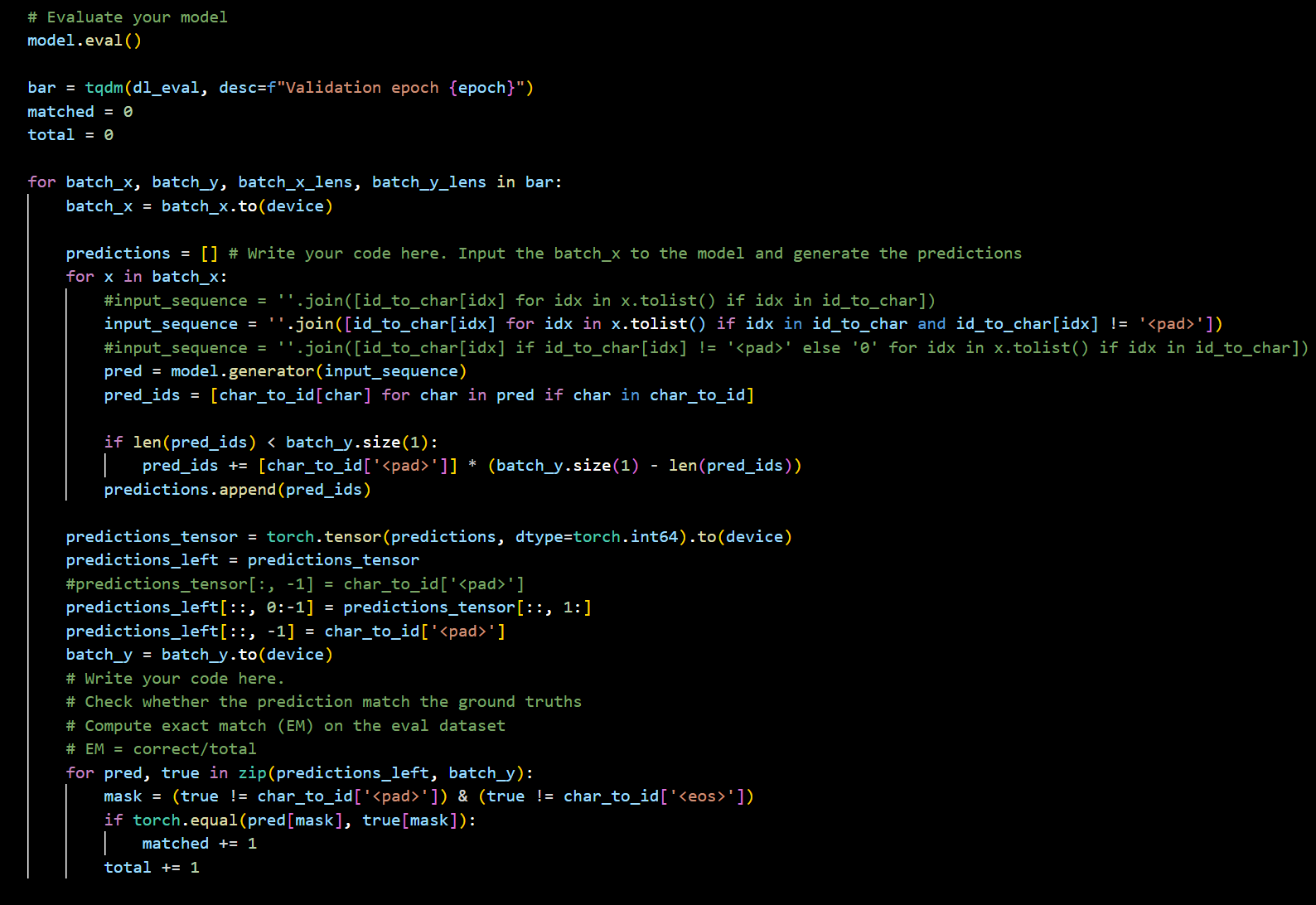


其中在計算Loss的時候一直出現錯誤，在將報錯丟進chatGPT詢問後才發現需要加入view()，這是PyTorch用於重塑Tensor的函數。

而重塑Tensor是為了將整個批次和序列中的所有預測和Label展平（或者說壓平）成一維，以便計算Loss。

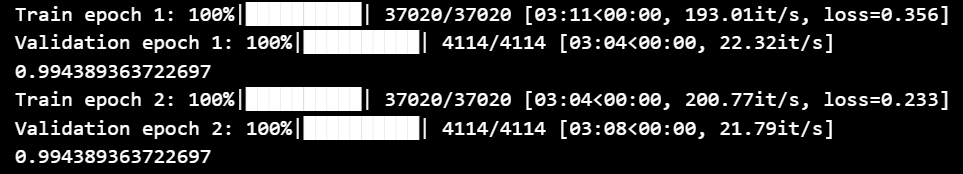
**TODO6**

接續來到評估的部分，這個部分花費了比較多時間才克服，首先模型進入評估模式，然後照範例程式碼最下面Generation的格式按批次來輸入Evaluation data，接著得到預測值pred，將pred中的每個字符轉換回對應的ID，並將這些ID存放在pred\_ids中。然後檢查 pred\_ids的長度是否小於batch\_y的序列長度。如果pred\_ids的長度小於目標序列長度，則在pred\_ids末尾補上<pad>的ID，直到它與目標序列長度相等。最後再把pred\_ids加到predictions列表中。

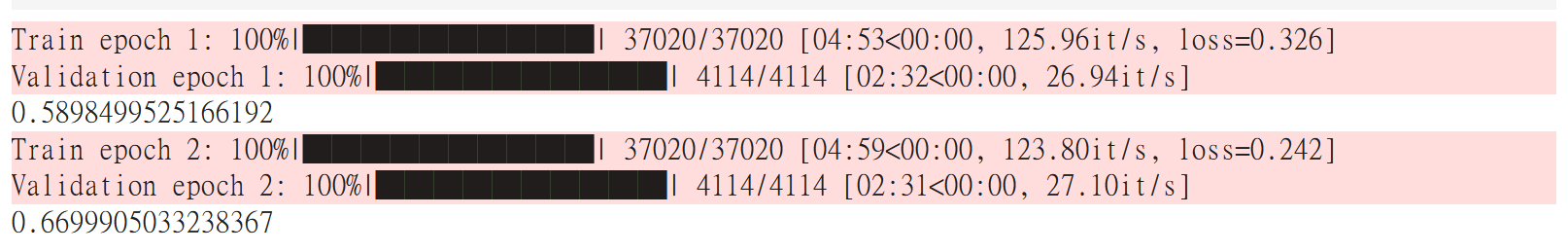


最後將predictions轉換為一個PyTorch Tensor，因為Label是有往左移動一格的，所以我們也將predictions\_tensor每行所有元素向左移一格，最後得到predictions\_left可以來跟ground truth進行比較。

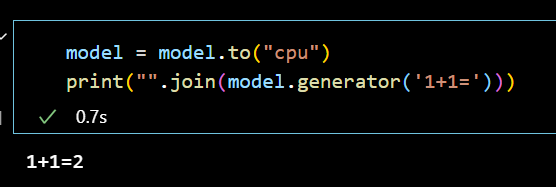
將預測結果跟batch\_y一一比較，使用mask將padding的地方遮住不做比較，最後得到的結果如下：



以Colab的CPU計算一個epoch大概需要兩個小時，T4 TPU則大概3分鐘，在我的實驗室主機上運算也差不多時間。可以看到在訓練階段的loss分別來到0.356和0.233，照理來說對照範例程式碼，



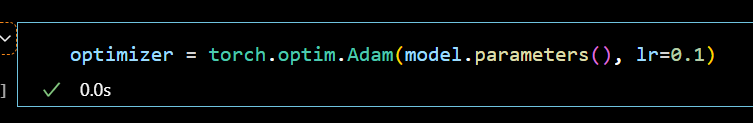
預測的準確率應該差不多是60%左右。然而實際的預測準確率結果卻高達99%，訓練時的loss對於這樣的結果，我個人覺得如果不是模型的學習效率太好，那麼就是我可能在哪個環節步驟搞錯了，後者的可能性大概比較高。



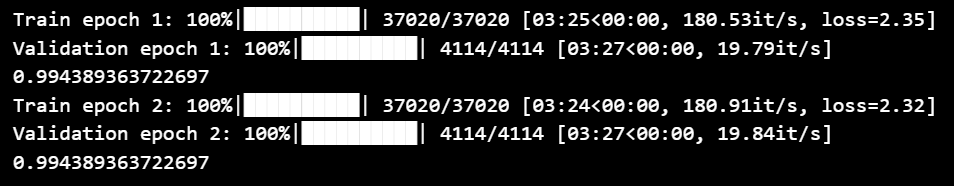
最後在generation的部分，也有成功預測出結果，可以說訓練本身是正確的，但可能在evaluation的某個環節出現了問題。

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

有嘗試將learning rate調整成0.1進行訓練：



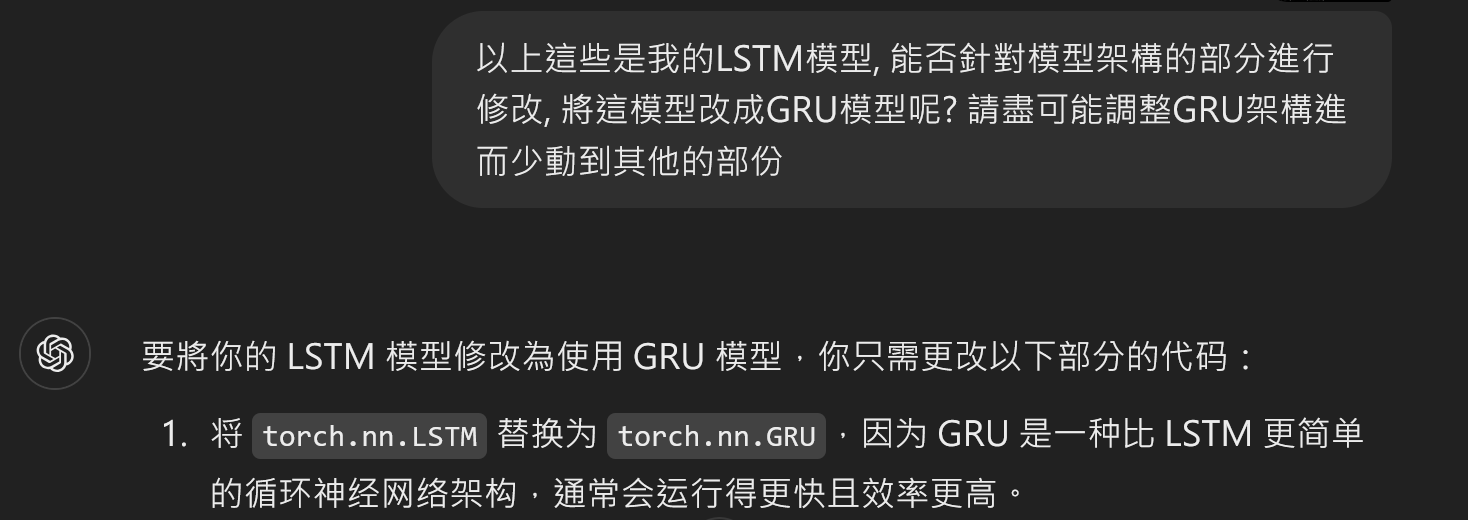
而其結果如下：

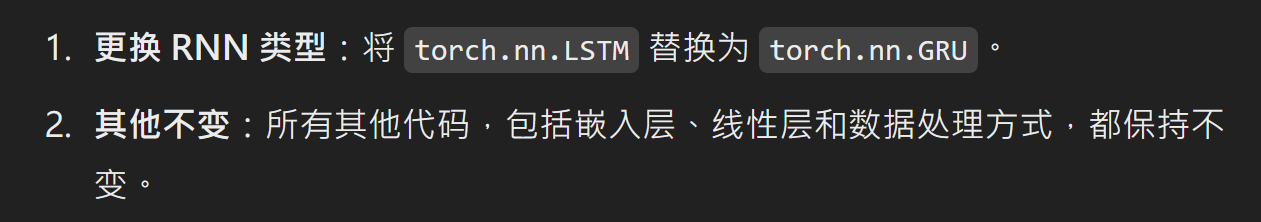


雖然就預測結果看不出端倪，但可以看到在Loss的部分分別是2.35和2.32。似乎使用不同的learning對模型訓練有顯著影響，主要體現在收斂速度和模型整體性能上，可能因為每一步向Local Loss最小值邁進的步伐較大，訓練過程可能變得不穩定，導致模型越過最佳解甚至發散。此外也可能會在最小值附近游走，無法穩定下來。

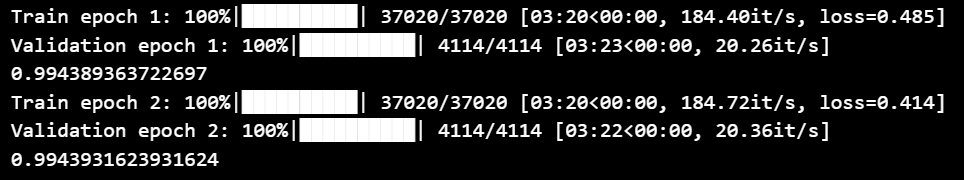
● If you use RNN or GRU instead of LSTM, what will happen to the quality of your answer generation? Why?

因為對pytorch的模型不了解，因此向chatGPT詢問如何修改模型：





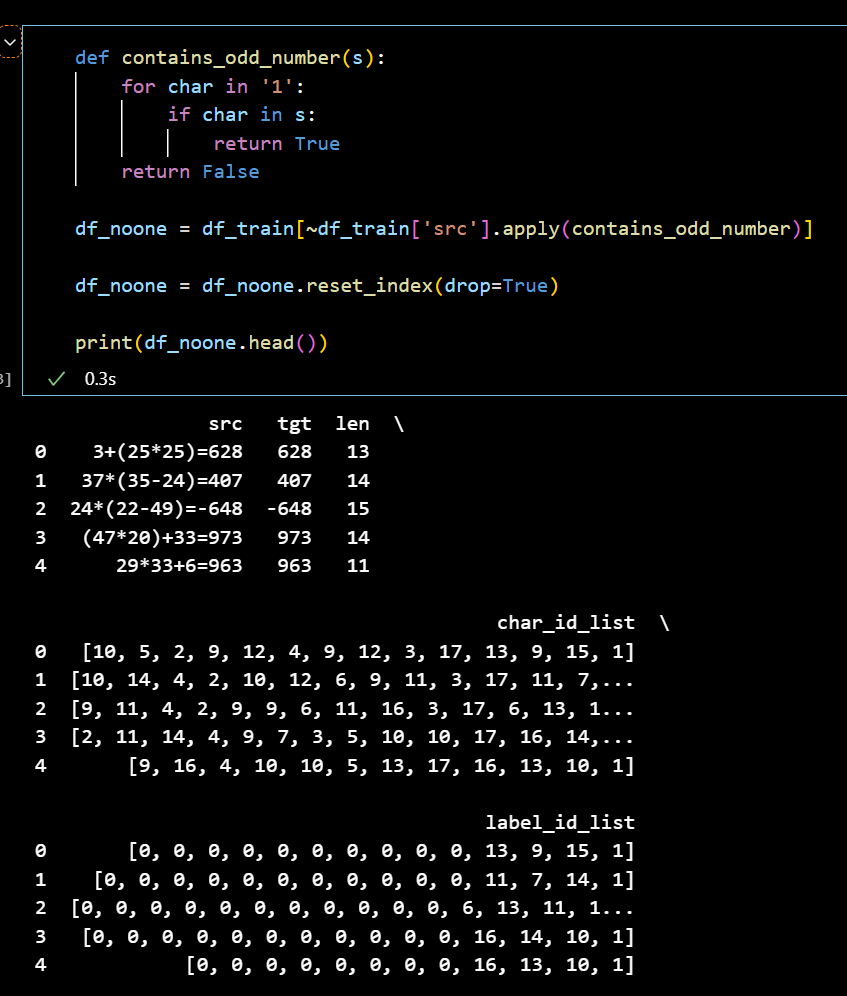
在進行一些調整後，訓練及測試以後結果如下：



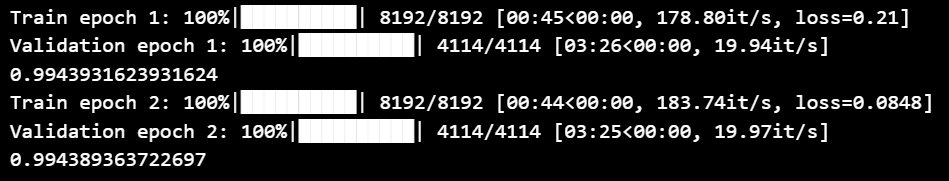
可以看到Loss分別是0.485和0.414，基本上稍微遜色於LSTM，可以預期生成預測的準確率可能會降低。可能是因為RNN和GRU比LSTM結構更簡單，是最基礎的回歸神經網路模型，因此它們不如LSTM擅長處理長期依賴問題，尤其是在長序列中會遇到梯度消失問題。因此對於需要長期記憶的任務，LSTM可能會表現得更好。

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

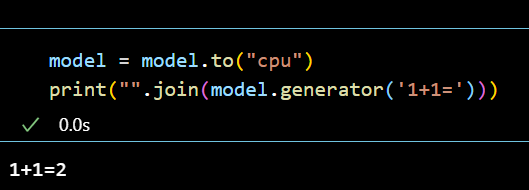
我們針對資料集做篩選，把所有包含’1’的字串都排除掉進而形成新的dataset：



進行過訓練以後結果如下：



在訓練方面反而是非常順利，說不定有overfitting，很難看得出來預測結果如何，但在預測方面反而有成功預測：



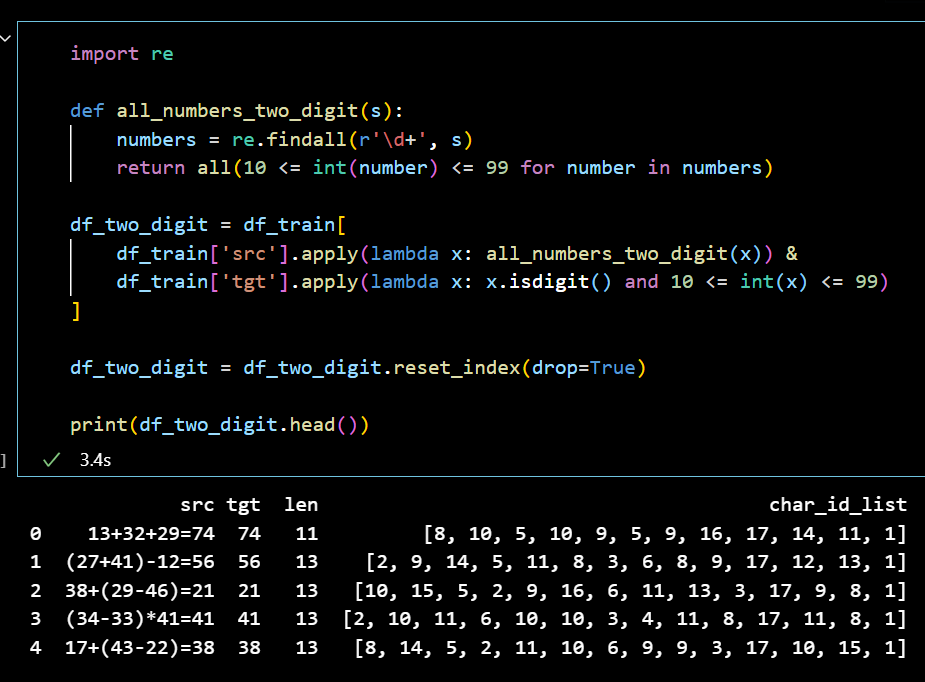
理論上有些數字從未出現在訓練資料中，在生成預測時，模型對這些數字的處理可能會較差。由於模型缺乏這些數字的學習經驗，無法準確地預測或生成涉及這些數字的結果。

● If we construct an evaluation set using three-digit numbers while the

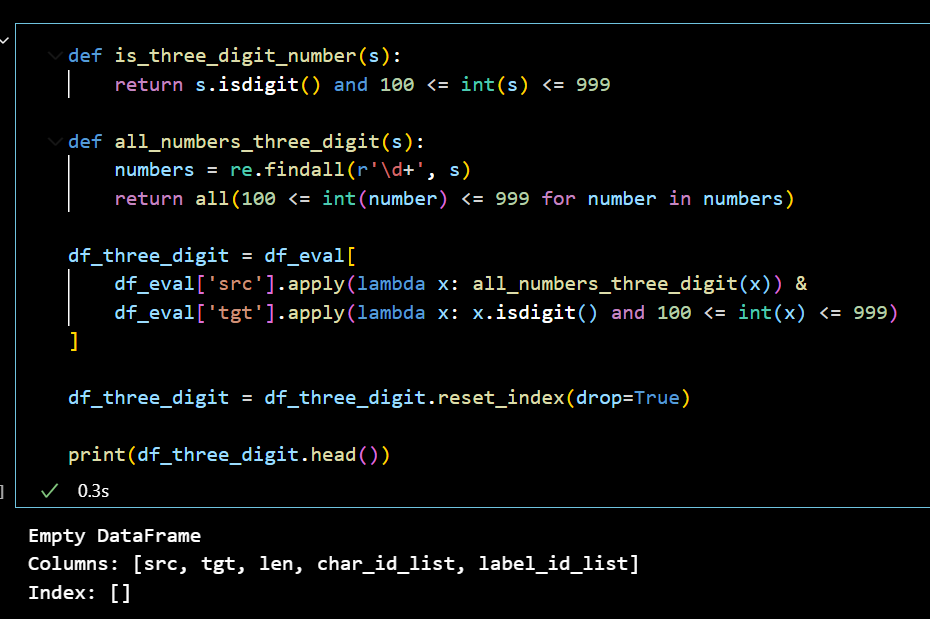
training set is constructed from two-digit numbers, what will happen to

the quality of your answer generation?

同樣有分別對訓練集和測試集進行篩選，在訓練集方面沒問題：



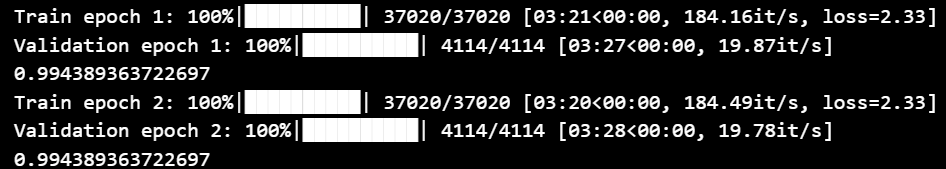
但在測試集，要全都是三位數字太困難，沒有任何一筆data符合：



因此很可惜沒有辦法做訓練來進行測試。但在理論上如果預測集由三位數構成，而訓練集由兩位數構成，預測的準確率可能會下降。因為模型在訓練期間未見過三位數的資料，而像RNN這樣對於位數敏感的模型習慣的都是二位數的格式，因此它可能無法預測更複雜的三位數資料。

● Why do we need gradient clipping during training?

進行沒有gradient clipping的訓練，其結果如下：



Loss都是2.33，說明訓練時Loss收斂得相當糟糕，而這可能是我們需要進行clipping來防止梯度爆炸。當梯度變過大時，參數更新幅度也會過大，導致訓練不穩。可見梯度裁剪有助於穩定訓練過程。