

GAI Project 2.a Arithmetic text generation

F74101254 資訊系 張暉俊

● Analysis

1. Model analysis

```
class CharRNN(torch.nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim):
        super(CharRNN, self).__init__()

        # Embedding層
        self.embedding = torch.nn.Embedding(num_embeddings=vocab_size,
                                             embedding_dim=embed_dim,
                                             padding_idx=tokenizer.cal_to_id['<pad>'])

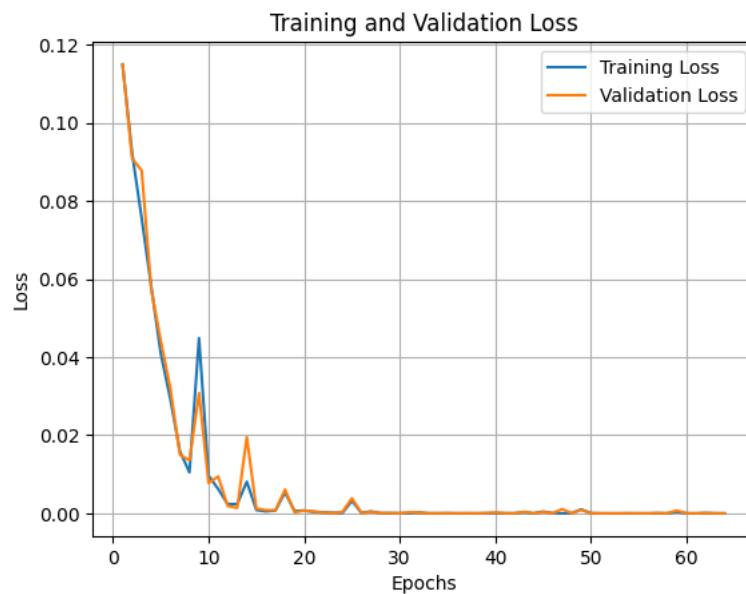
        # RNN層
        self.rnn_layer1 = torch.nn.LSTM(input_size=embed_dim,
                                         hidden_size=hidden_dim,
                                         batch_first=True)

        self.rnn_layer2 = torch.nn.LSTM(input_size=hidden_dim,
                                         hidden_size=hidden_dim,
                                         batch_first=True)

        # output層
        self.linear = torch.nn.Sequential(torch.nn.Linear(in_features=hidden_dim,
                                                           out_features=hidden_dim),
                                           torch.nn.ReLU(),
                                           torch.nn.Linear(in_features=hidden_dim,
                                                           out_features=vocab_size))
```

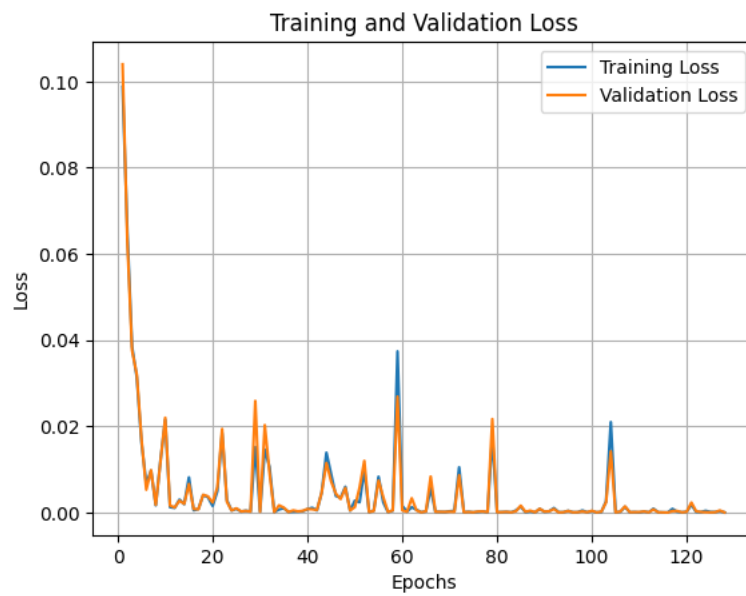
在模型的設計上，採用跟助教撰寫的程式碼相同的架構，一層的 embedding 層，兩層的 RNN 層以及一層 output 層，在實作上使用的是以 LSTM 這個模型為主。

loss rate 的部分則是會隨著超參數設定的不同而有細微的變動，但總結來說會在 epoch = 30 左右的時候來到最低數值，隨著 epoch 持續加大，他的 loss rate 則會上下起伏不定，再慢慢降回去。



```
batch_size = 512
epochs = 64
embed_dim = 256
hidden_dim = 256
lr = 0.001
grad_clip = 1
max_length = 25
vocab_size = len(tokenizer.cal_to_id)
```

上圖訓練結果之超參數，其資料數量為 512,000 筆



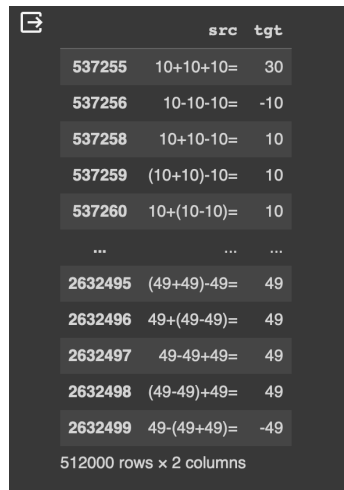
加大到 epoch = 128 的 loss rate 結果圖，其超參數除 epochs 外皆相同

2. Dataset analysis

我總共生成了四份 dataset

A. filtered_plus_minus.csv

取出含 3 個 2 位數字之加減運算



	src	tgt
537255	10+10+10=	30
537256	10-10-10=	-10
537258	10+10-10=	10
537259	(10+10)-10=	10
537260	10+(10-10)=	10
...
2632495	(49+49)-49=	49
2632496	49+(49-49)=	49
2632497	49-49+49=	49
2632498	(49-49)+49=	49
2632499	49-(49+49)=	-49

51200 rows x 2 columns

B. filtered_PlusAndMinus_random.csv

取值為集中值，為 10 到 25 以內各個數的加減運算

```
# 取集中值 (10 - 25) 的二位數字運算
def check_PlusAndMinus_random(row):
    # 檢查是否包含三個二位數字的加減法
    src = row['src']
    # 使用正則表達式找到所有的數字
    numbers = re.findall(r'\b\d{2}\b', src)
    # 過濾包含加減號以外的運算式
    valid_numbers = [int(num) for num in numbers if (int(num) < 26)]
    return len(valid_numbers) >= 3 and ('+' in src or '-' in src) and ('*' not in src and '/' not in src)
```

C. filtered_PlusAndMinus_extreme.csv

取極端值，10 到 17 或 41 到 49 間各個數字的加減運算

```
# 取極端值 (10 - 17, 41 - 49) 的二位數字運算
def check_PlusAndMinus_extreme(row):
    # 檢查是否包含三個二位數字的加減法
    src = row['src']
    # 使用正則表達式找到所有的數字
    numbers = re.findall(r'\b\d{2}\b', src)
    # 過濾包含加減號以外的運算式
    valid_numbers = [int(num) for num in numbers if (int(num) > 40) or (int(num) < 18)]
    return len(valid_numbers) >= 3 and ('+' in src or '-' in src) and ('*' not in src and '/' not in src)
```

D. filtered_PlusAndMinus_out.csv

取 34 到 49 間各個數字的加減運算

```
# 取極端值 (34 - 49) 的二位數字運算
def check_PlusAndMinus_out(row):
    # 檢查是否包含三個二位數字的加減法
    src = row['src']
    # 使用正則表達式找到所有的數字
    numbers = re.findall(r'\b\d{2}\b', src)
    # 過濾包含加減號以外的運算式
    valid_numbers = [int(num) for num in numbers if (int(num) > 33)]
    return len(valid_numbers) >= 3 and ('+' in src or '-' in src) and ('*' not in src and '/' not in src)
```

A 是我用來測試超參數的影響以及我所能訓練出的最高分數為多少，B、C、D 則是有一定規則取出的 dataset，資料皆根據規則抓出後的總資料量隨機提取 30,000 筆。另外，評估的資料特意使用了 B 資料的前 3000 筆，這麼做的原因是想測驗說，如果根據驗算測試的題目著重去測驗的話，答對率是否會比較高？結果如下附表

dataset	B (random)	C (extreme)	D (out)
Accuracy	75.91%	14.77%	0.33%

根據上表的結果我們可以發現，B dataset 有著較高的答對率，而完全沒有學習到相關驗證題目的 D dataset 則是幾乎都無法答對題目，而 C dataset 則是因為有加減學習到一些資料，所以相對提升了一些，但也因為資料量大多不服驗證題目內容，導致打對率相對慘淡。

2991	Expression: 17+(17-21)=	Ans: 13	Pred: 13	Result: Correct
2992	Expression: 16+15-11=	Ans: 20	Pred: 20	Result: Correct
2993	Expression: 14-(16+19)=	Ans: -21	Pred: -2	Result: Wrong
2994	Expression: (24+12)-15=	Ans: 21	Pred: 21	Result: Correct
2995	Expression: 16+10+14=	Ans: 40	Pred: 40	Result: Correct
2996	Expression: 23-25-11=	Ans: -13	Pred: -1	Result: Wrong
2997	Expression: 17-24+13=	Ans: 6	Pred: 6	Result: Correct
2998	Expression: 18+(16-23)=	Ans: 11	Pred: 11	Result: Correct
2999	Expression: 22-10-24=	Ans: -12	Pred: -1	Result: Wrong
3000	Expression: (10-13)+25=	Ans: 22	Pred: 22	Result: Correct
Evaluating(Accuracy): 75.97%				

B dataset 的驗證結果圖，可以發現運算式的數字皆介於 10 到 25 之間。

▶	batch_size = 256
	epochs = 64
	embed_dim = 256
	hidden_dim = 256
	lr = 0.001
	grad_clip = 1
	max_length = 25
	vocab_size = len(tokenizer.cal_to_id)

運算 B、C、D 時，使用之超參數，使用模型皆為 LSTM。

3. Discussion

different learning rate:

▶	batch_size = 512
	epochs = 64
	embed_dim = 256
	hidden_dim = 256

Learning rate	0.01	0.001	0.0001
Accuracy	33.98%	67.49%	66.71%

由上述表格可知，learning rate 越小越好，但是也有一極限值，不需要一味地調小，當到一數值以下時，其對訓練訓能的影响則逐漸變小。

different batch size:

```
epochs = 64
embed_dim = 256
hidden_dim = 256
lr = 0.001
```

Batch size	64	256	512
Accuracy	67.48%	66.06%	66.71%

在未實驗前，因緣際會的查到 IKMLab 的 github，裡面有助教們整理的上課教材，因為自身對 Pytorch 不是很熟悉的原因，就去拜讀了一下，其中，在教材的最下方練習，推薦可以加大 batch size 或許會有更好的學習成果，因此我先入為主的認為只要一直加大 batch size，就能有更好的學習成果，結果實驗數據打了我的臉。在查閱一些資料後才明白，原來 batch size 的大小更多的是影響 training 的速度，但不一定會使得 model 有更好的學習成果，還是要配合著模型的各個超參數。

characteristic of my model:

我使用的模型如上所述，是以 LSTM 為主。選擇此模型的原因是因為，相較於 RNN 來說，他擁有更好的效能。此外，forget gate 的存在也讓這個模型有更好的學習能力與更新資料能力，而且此模型本身就是用來訓練長期記憶的模型，也符合我們希望他記取更多運算式的功能。

● Bonus

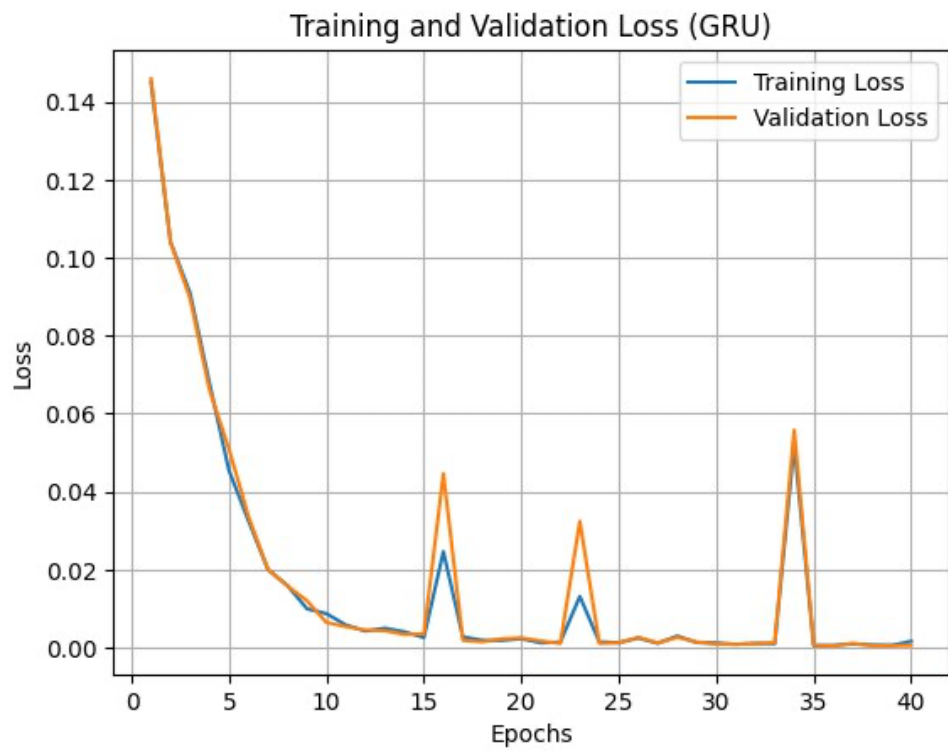
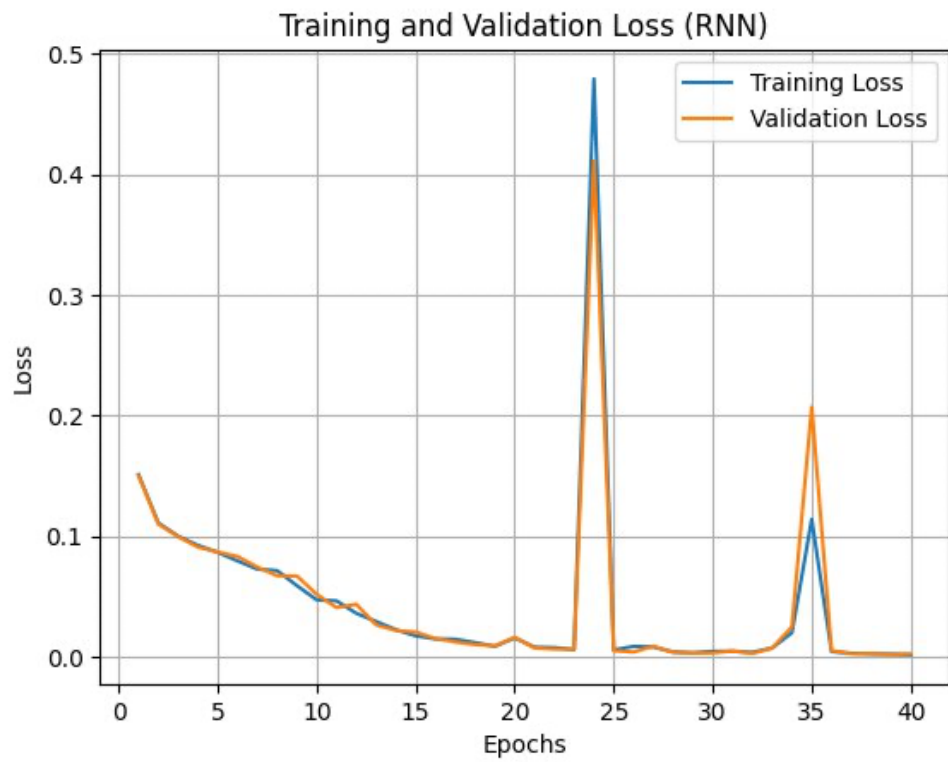
1. Compare the performance of multiple models and provide a brief analysis.

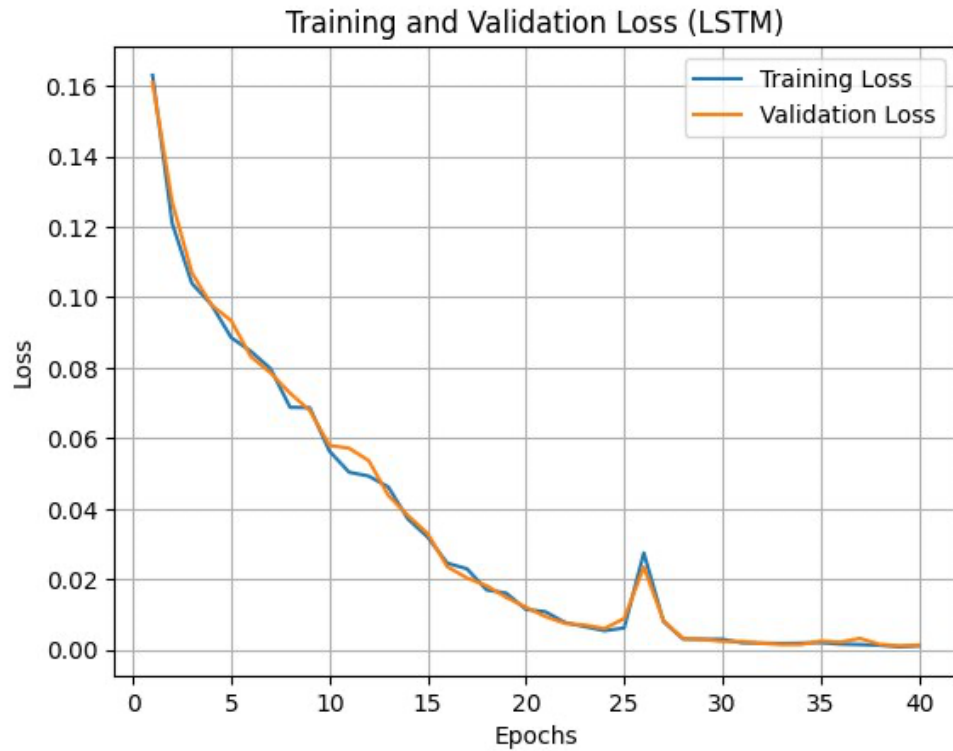
測驗時使用之超參數

```
batch_size = 512
epochs = 40
embed_dim = 256
hidden_dim = 256
lr = 0.0001
grad_clip = 1
max_length = 25
vocab_size = len(tokenizer.cal_to_id)
```

Model	RNN	LSTM	GRU
Accuracy	67.25%	66.39%	65.75%

各模型 Losing rate 示意圖





在我原本的想法中，我覺得 RNN 模型因為其架構較為簡單，所以應該其學習成果應該較差，但測驗結果跟我想的完全相反，於是不信邪的我，決定將 batch size 調小後，再嘗試一遍，因為我覺得說不定是因為一次訓練的數量比較少，所以 RNN 會有優勢。

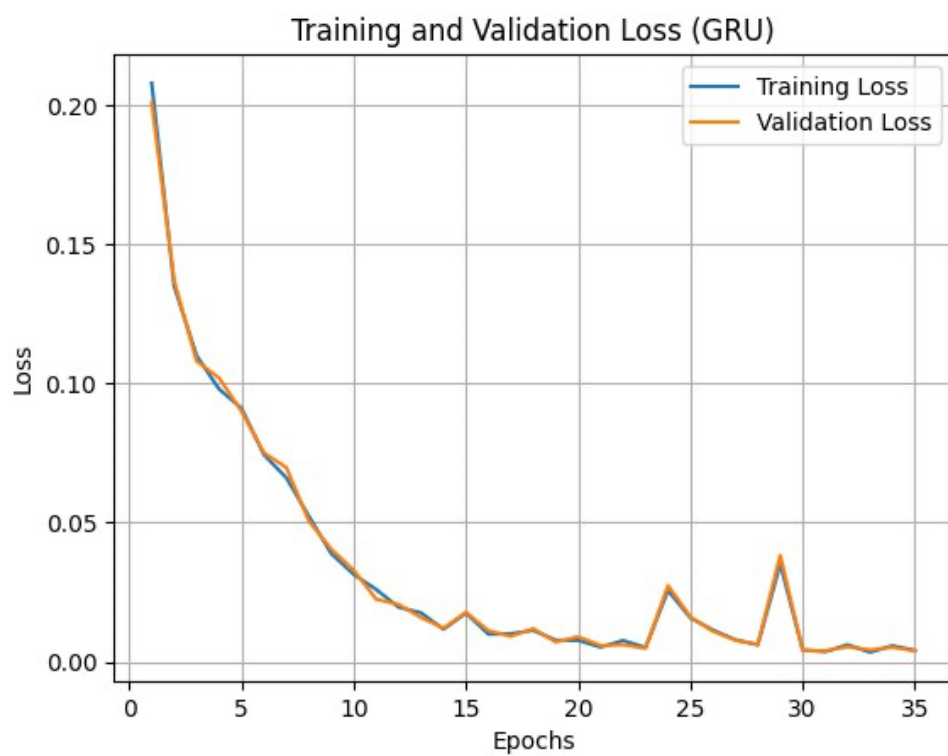
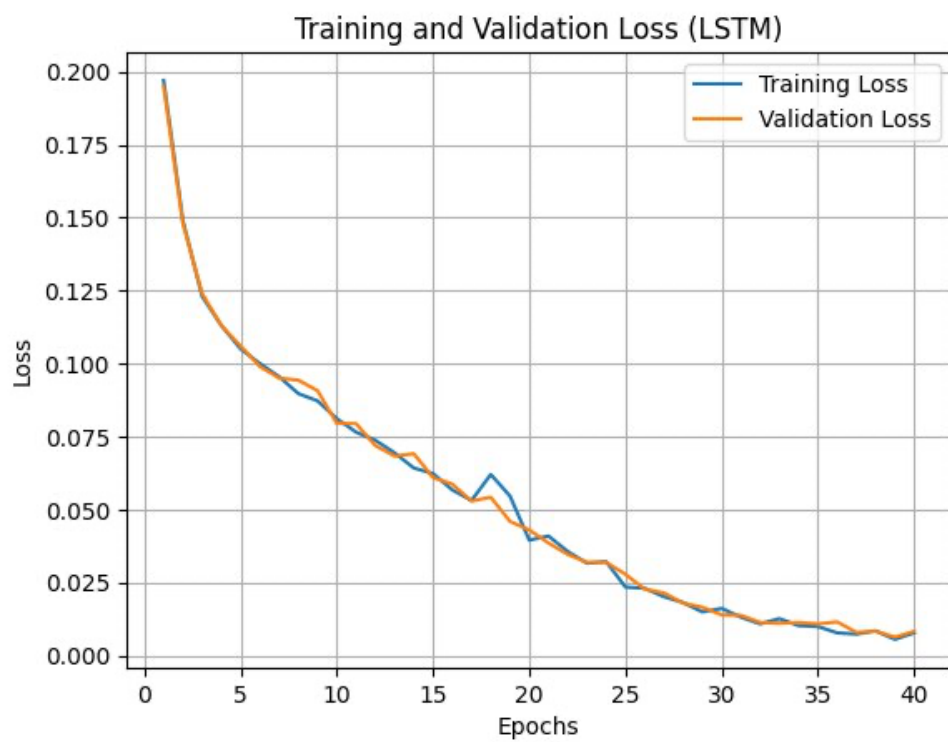
```

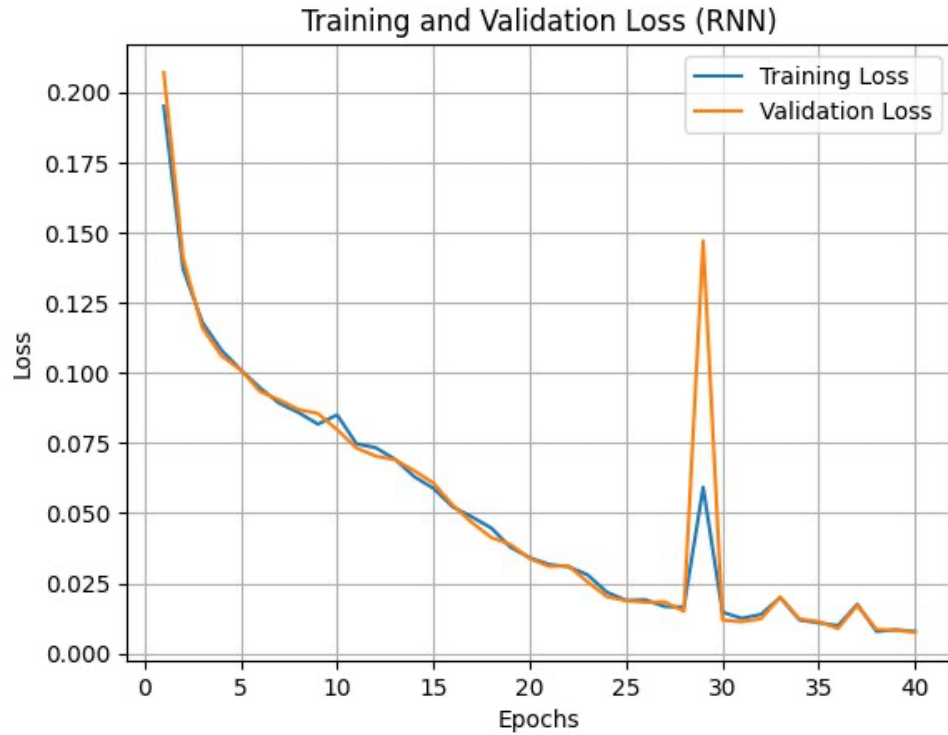
batch_size = 256
epochs = 40
embed_dim = 256
hidden_dim = 256
lr = 0.0001
grad_clip = 1
max_length = 25
vocab_size = len(tokenizer.cal_to_id)

```

Model	RNN	LSTM	GRU
Accuracy	67.47%	67.49%	67.49%

各模型 Losing rate 示意圖





結果竟然實測結果一模一樣，後來去查了資料才知道，其實 RNN 模型最大的缺陷是梯度上的問題，我更改 batch size 說實話應該不太能得到我想要的結果。但其實仔細去看還是有差別的，關鍵就在 losing rate 的圖片裡，可以發現在第 40 個 epoch 時，loss 的大小排序依據是 $RNN > LSTM > GRU$ ，可以側面推測出，其實 GRU 模型的學習效果應該是比較好的沒錯，另外，在 LSTM 的圖片也可以發現其下降穩定，相較於其他兩者比較不會出現忽大忽小的狀況。