DL Lab3: seq2seq model

程式實現

首先建立兩個字典，用於將文本數據轉換成數字形式，讓模型能看得懂並且進行訓練及預測，最後為了將預測輸出轉換成可讀的文本，需要再將數字轉換回相應的字符。

```
char2index = {'sos': 0, 'eos': 1, 'pad': 2, **{char: i + 3 for i, char in enumerate(string.ascii_lowercase)}}
index2char = {i: char for char, i in char2index.items()}
```

- char2index：爲每個字符定義唯一的索引值，其中 sos、eos 和 pad 是三個特殊的符號，分別代表 Start of sequence、End of sequence 和 padding。
- index2char：將索引值轉換回對應的字符，這裡透過對 char2indax 字典的項目進行迭代，並將鍵（字符）和值（索引）交換位置來實現。

接著將資料載入，並處理成要輸入到模型裡的形式。

```
import json

def max_length_in_json(json_file):
    with open(json_file, 'r') as file:
        data = json.load(file)
    max_length = 0
    for item in data:
        for word in item['input']:
            max_length = max(max_length, len(word))
    return max_length

print("Train data:", max_length_in_json('train.json'))
print("Test data:", max_length_in_json('test.json'))
print("New Test data:", max_length_in_json('new_test.json'))
```

✓ 0.0s

```
Train data: 19
Test data: 16
New Test data: 13
```

```
class SpellCorrectionDataset(Dataset):
    def __init__(self, root, split = 'train', padding = 21):
        super(SpellCorrectionDataset, self).__init__()
        self.data, self.targets = self.load_data(root, split)
        self.padding = padding

    def load_data(self, root, split):
        file_path = f"{root}/{split}.json"
        with open(file_path, 'r') as file:
            data = json.load(file)
        input_list, target_list = [], []
        for item in data:
            input_list.extend(item['input'])
            target_list.extend([item['target']] * len(item['input']))
        return input_list, target_list

    def tokenize(self, text):
        return [char2index.get(char, char2index['pad']) for char in text]

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        input_text = self.data[index]
        target_text = self.targets[index]
        input_ids = [char2index['sos']] + self.tokenize(input_text) + [char2index['eos']] + [char2index['pad']] * (self.padding - len(input_text) - 2)
        target_ids = [char2index['sos']] + self.tokenize(target_text) + [char2index['eos']] + [char2index['pad']] * (self.padding - len(target_text) - 2)
        return torch.tensor(input_ids, dtype = torch.long), torch.tensor(target_ids, dtype = torch.long)
```

資料集中最長的單字長度為 19，加上 sos 和 eos 兩個 token，因此將 padding 設為 19+2=21，指定每個序列的填充長度，用於確保所有序列在處理時具有一致的長度。

- 文本標記化（tokenize）：透過查找每個字符在 char2index 字典中索引，將文本進行轉換。如果字符不在字典中，則使用 pad 索引。
- 獲取指定索引的數據（__getitem__）：根據索引獲取特定的 input 和 target，接著透過 tokenize 進行轉換，並在開頭加上 sos，末尾加上 eos，並用 pad 將序列長度補到 21。

最後透過 DataLoader 將 train.json、test.json、new_test.json 三個檔案載入，分別用於訓練、驗證及測試。

```
root_path = '../data/'
trainset = SpellCorrectionDataset(root_path, split = 'train')
trainloader = DataLoader(trainset, batch_size = 16, shuffle = True)
valset = SpellCorrectionDataset(root_path, split = 'test')
valloader = DataLoader(valset, batch_size = 16, shuffle = False)
testset = SpellCorrectionDataset(root_path, split = 'new_test')
testloader = DataLoader(testset, batch_size = 16, shuffle = False)
```

LSTM 程式碼調整及結果比較

1. Teacher Forcing

在 Seq2Seq 的 class 中，有設定一個 Teacher forcing 的機制是用來選擇下一個 time step 的輸入要是什麼：當隨機給定的 boolean 值為 True，就會使用真實的目標輸出，若 boolean 值為 False，則使用模型的預測輸出。這種方法可以加速訓練過程，並幫助模型更有效地學習產生準確的輸出序列。

這裡有使用兩種方法：

(1) 對**每個 time step** 重新評估是否使用 Teacher forcing，提供了更多的隨機性和靈活性。因為它允許模型在每個 time step 經歷不同的訓練條件，這有助於模型學習如何在不依賴於真實目標輸出的情況下進行預測。

```
input = trg[0,:]
for t in range(1, trg_len):
    output, hidden = self.decoder(input, hidden)
    outputs[t] = output
    teacher_force = random.random() < teacher_forcing_ratio
    top1 = output.argmax(1)
    input = trg[t] if teacher_force else top1
outputs_T = torch.transpose(outputs,0,1)
return outputs_T
```

(2) 對**每個 batch** 統一決定是否使用 Teacher forcing，提供了更穩定的訓練環境。因為這樣做使得模型在整個 batch 的所有 time step 都經歷相同的都訓練條件，這有助於模型學習如何在一致的環境下進行預測。

```
input = trg[0,:]
use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False
for t in range(1, trg_len):
    output, hidden = self.decoder(input, hidden)
    outputs[t] = output
    input = trg[t] if use_teacher_forcing else output.argmax(1)
outputs_T = torch.transpose(outputs, 0, 1)
return outputs_T
```

2. Compare performance changes due to different parameters and model structures

基本模型架構及參數設定：

- ✓ Input/Output dimension: 29
- ✓ Embedding dimension: 256
- ✓ Hidden dimension: 512
- ✓ Batch size: 16
- ✓ Optimizer: SGD
- ✓ Learning rate: 0.05

在下方的表格中，有嘗試了其他參數及架構的調整，主要列出了六種進行比較。針對此項拼字修正的任務，從實驗中可以發現Encoder和Decoder的Layer不適合太深，且兩種Teacher forcing的機制看起來沒有太大的差異。

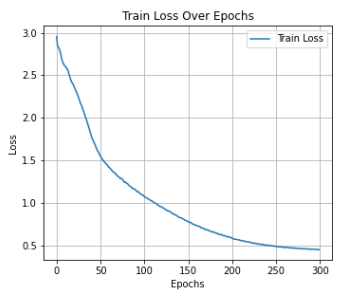
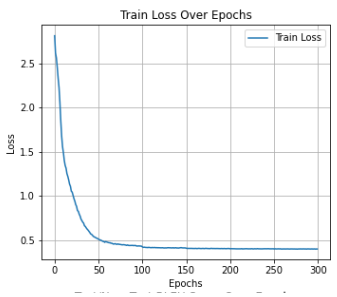
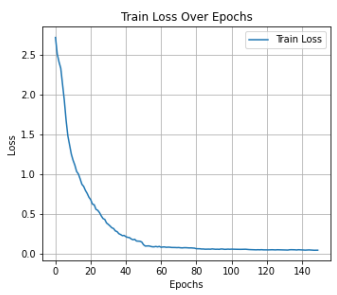
然而在加入momentum和學習率調整策略後，模型可以在較少的訓練epoch數量中，實現更好的準確率和Bleu-4 score。

另外除了最終的準確率及Bleu-4 score，表格中還有特別列出整個訓練過程中，test.json和new_test.json兩個測試集平均效果最好的準確率及其Bleu-4 score。

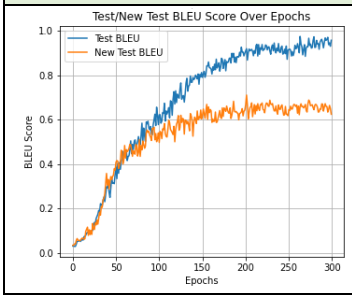
LSTM	1	2	3	4	5	6
Layer	3	2	2	2	2	2
Teacher forcing	Single letter			Whole word		
Momentum	None			0.9		
Scheduler (gamma/step)	None	0.9 100	0.95 50	0.5 50	0.5 50	0.5 30
Epoch	400	300	300	300	150	150
Final accuracy (test/new_test)	60/30	74/44	90/42	62/40	82/48	88/52
Bleu-4 score (test/new_test)	0.865 0.640	0.884 0.662	0.958 0.625	0.842 0.649	0.932 0.699	0.959 0.682
Best accuracy (test/new_test)	84/48	86/46	94/48	92/60	94/52	92/58
Bleu-4 score (test/new_test)	0.923 0.713	0.896 0.654	0.967 0.689	0.944 0.751	0.960 0.695	0.971 0.717

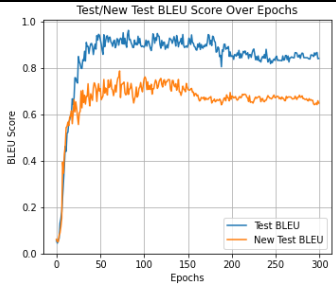
3. Compare different methods through graphs

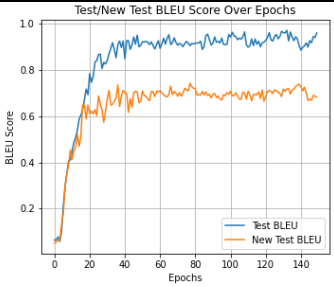
從下方的圖表中，可以看到模型在加了 momentum 之後，收斂的速度提升了許多。而模型 4 和模型 6 在 150 個 epoch 前的 Cross Entropy loss 和 Bleu-4 score 都有良好的表現，模型 4 在大約 150 個 epoch 之後，Bleu-4 score 就停滯了，甚至有所下降，從這個趨勢可以看到模型 4 在測試集的泛化能力有限，因此減少模型 4 的 epoch 數量，並且嘗試在更少的迭代次數中進行學習率衰減，最後呈現模型 6 的效果。

LSTM 3	LSTM 4	LSTM 6
CrossEntropy training loss		
		

BLEU-4 testing score







Best accuracy

test.json

```
==================
input:  oportunity
target: opportunity
pred:   opportunity
==================
input:  parenthasis
target: parenthesis
pred:   parenthesis
==================
input:  recetion
target: recession
pred:   recession
==================
input:  scadual
target: schedule
pred:   schedule
Bleu-4 score:  0.9668, Accuracy:  0.9400
```

```
==================
input:  oportunity
target: opportunity
pred:   opportunity
==================
input:  parenthasis
target: parenthesis
pred:   parenthesis
==================
input:  recetion
target: recession
pred:   recession
==================
input:  scadual
target: schedule
pred:   schedule
Bleu-4 score:  0.9437, Accuracy:  0.9200
```

```
==================
input:  oportunity
target: opportunity
pred:   opportunity
==================
input:  parenthasis
target: parenthesis
pred:   parenthesis
==================
input:  recetion
target: recession
pred:   recession
==================
input:  scadual
target: schedule
pred:   schedule
Bleu-4 score:  0.9705, Accuracy:  0.9200
```

new_test.json

```
==================
input:  unforgatealbe
target: unforgettable
pred:   unforgubateble
==================
input:  unforgattable
target: unforgettable
pred:   unforgubatel
==================
input:  vesiable
target: visible
pred:   visible
==================
input:  visable
target: visible
pred:   visible
Bleu-4 score:  0.6885, Accuracy:  0.4800
```

```
==================
input:  unforgatealbe
target: unforgettable
pred:   unforgetable
==================
input:  unforgattable
target: unforgettable
pred:   unforgrateble
==================
input:  vesiable
target: visible
pred:   visible
==================
input:  visable
target: visible
pred:   visible
Bleu-4 score:  0.7509, Accuracy:  0.6000
```

```
==================
input:  unforgatealbe
target: unforgettable
pred:   unfortunately
==================
input:  unforgattable
target: unforgettable
pred:   unfortunable
==================
input:  vesiable
target: visible
pred:   visible
==================
input:  visable
target: visible
pred:   visible
Bleu-4 score:  0.7166, Accuracy:  0.5800
```

Transformer 結果比較

1. Compare performance changes due to different parameters

基本模型架構及參數設定：

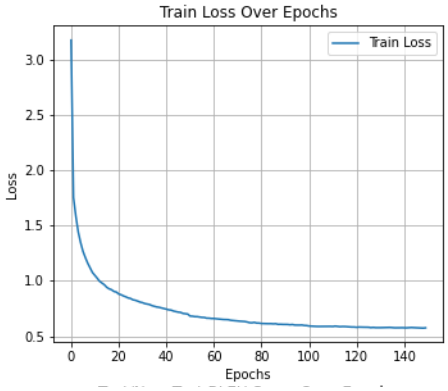
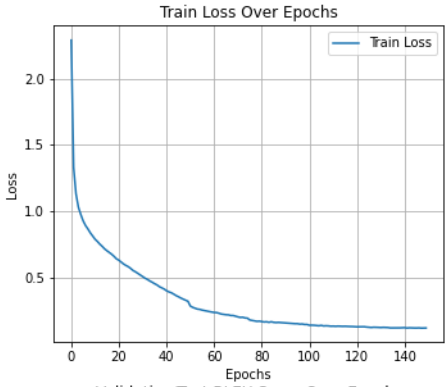
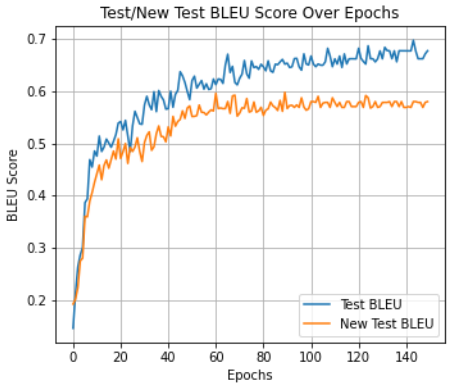
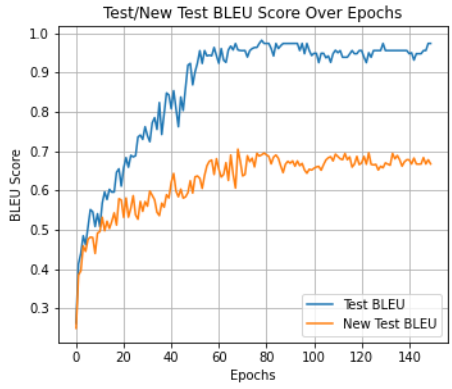
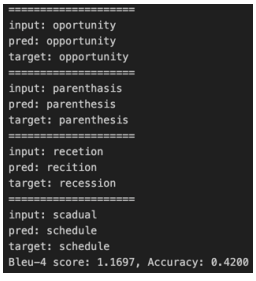
- ✓ Token embedding layer: 29
- ✓ Hidden dimension: 512
- ✓ Number of Layers: 8
- ✓ Multi-head: 8
- ✓ Feedforward dimension: 1024
- ✓ Optimizer: SGD
- ✓ Epoch: 150

在下方的表格中，主要嘗試了batch size和scheduler的調整，這邊列出了三種進行比較。因為使用了8層的網路，需要花比較多的訓練時間，所以一開始決定將batch size設大一點，這裡嘗試64，但訓練的最終結果及最佳結果在test.json及new_test.json測試集中分別只有40%和20%左右的準確率。後來嘗試調整layer、learning rate、scheduler等參數都沒有太大的改善，最後有明顯的效果提升是將batch size調小到16，最佳的準確率有到98%。

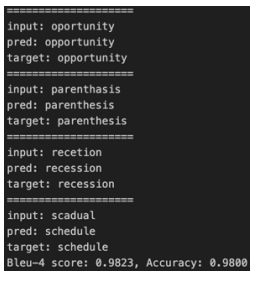
Transformer	1	2	3
Batch size	64	16	16
Learning rate	0.001	0.001	0.001
Epoch	150		
Scheduler (gamma/step)	0.5 25	0.5 25	0.5 30
Final accuracy (test/new_test)	40/20	96/48	90/50
Bleu-4 score (test/new_test)	0.677/0.580	0.974/0.667	0.935/0.690
Best accuracy (test/new_test)	42/20	98/48	92/50
Bleu-4 score (test/new_test)	0.693/0.585	0.982/0.691	0.943/0.705

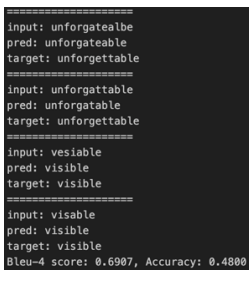
2. Compare different methods through graphs

觀察下方的圖表，其中模型 1 的 batch size 為 64，它的 Cross Entropy loss 和 Bleu-4 score 曲線在 epoch 70 之後就趨於平緩，藉由結果可以看出它沒有找到全局最佳解，但我嘗試把學習率調小後的模型表現更差。最後是將 batch size 調整為 16 後，才呈現出模型 2 的結果，雖然收斂速度相對較慢，但從結果來看有了大大的提升，表示模型 2 其較小的 batch size 在此項任務中具有更好的泛化能力，

<table>
<tr><th>Transformer 1</th><th>Transformer 2</th></tr>
<tr><td colspan="2">CrossEntropy training loss</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">BLEU-4 testing score</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Best accuracy</td></tr>
<tr><td>test.json</td><td>new_test.json</td><td>test.json</td><td>new_test.json</td></tr>
<tr><td>


</td><td>


</td><td>


</td><td>


</td></tr>
</table>

LSTM 和 Transformer 比較

針對兩種網路的程式碼進行了幾點比較如下（詳細內容須至 ipynb 檔案查看）：

	LSTM	Transformer
Encoder 和 Decoder 架構	nn.LSTM 是一個循環神經網路層，在 time step 之間遞歸地處理資訊。	TransformerEncoder 和 TransformerDecoder 內部利用了自注意力機制（Self-Attention）和前饋網絡（Feedforward Network）。
位置編碼	由於其遞歸的結構，能夠自然處理序列的時間資訊，所以不需要位置編碼。	PositionalEncoding 類提供了序列中單詞位置資訊的編碼，這是 Transformer 架構的一個核心特點，因為 Transformer 本身不具備處理序列位置資訊的能力。
嵌入層	均採用了 nn.Embedding 來將輸入的詞彙索引轉換為密集向量。	
Mask 和序列生成	LSTM 的 Seq2Seq 結構不需要 Mask，因為 Decoder 在生成每一輸出時只依賴於其前一個 time step 的隱藏狀態和最新生成的輸出。	在 Decoder 中，需要傳入 tgt_mask，以防止在預測下一詞時看到未來的詞。這是通過 att_mask 實現的，避免 Decoder 在生成輸出時作弊。
參數數量與複雜性	LSTM 的參數通常少於 Transformer，但由於其遞歸的特性，訓練可能需要更多的時間。	Transformer 通常擁有更多的參數，因為它使用了多頭注意力機制，而每個 head 都擁有自己的權重。此外，它還採用了獨立的 Encoder 和 Decoder 網絡，每個網絡都包含多個層次。

*繳交的 LSTM 程式檔為模型 6 的，Transformer 程式檔為模型 2 的。