

$$\begin{cases} 2x_1 + x_2 = 7 \\ x_1 + x_2 - 3x_3 = -10 \\ 6x_2 - 2x_3 + x_4 = 7 \\ 2x_3 - 3x_4 = 13 \end{cases}$$

Project2 a. Report

| 資訊系大四 F74092269 陳冠廷

任務介紹

作業 a 為加減乘除的深度學習計算器，透過生成資料和訓練RNN等序列模型，來了解深度學習在計算簡易算式的能力

資料生成

- 設定最大數值、哪些運算子、幾個數值運算、是否含有括號、多少筆資料來產生資料集
- 本次實作只考慮了以下設定
 - 3數運算
 - 最大數為40最小為0
 - 40000筆資料
 - 只有加減法
- 根據上述條件，我考慮以下兩種分布
 - 資料不含括號運算
 - 資料涵蓋括號運算(且只有一個括號)

```

# 創建單一筆資料
def generate_datum(
    number_ranges: list[str],
    operators: list[str],
    count: int,
    add_parentheses: bool
) -> dict[str, str]:

    nums = random.choices(number_ranges, k=count)
    ops = random.choices(operators, k=count-1)

    if add_parentheses:
        start, end = 0, count-1

        # 不接受將整個式子括住的括號
        while (start == 0 and end == count-1):

            pos = list(range(0, count))
            insert_pos = random.sample(pos, 2)
            start, end = insert_pos[0], insert_pos[1]
            if start > end:
                temp = start
                start = end
                end = temp

            # 上括號只會出現在數字前與下括號只會在數字後
            nums[start] = "(" + nums[start]
            nums[end] = nums[end] + ")"

    src = [ nums[0] ]

    for num, op in zip(nums[1:], ops):
        src.extend([op, num])

    # 轉為字串算式
    formula = "".join(src)
    # 計算正確答案
    answer = str(eval(formula))

```

```

# 尾部添加等號
formula += "="

return {
    "formula": formula, "answer": answer
}

# 創建多筆資料
def generate_dataset(
    max_digits: int = 2,
    operators: list[str] = ['+', '-'],
    max_count: int = 3,
    size: int = 1000,
    add_parentheses: bool = False
) -> pd.DataFrame:

    # 設定數字上下限
    number_ranges = [str(num) for num in range(0, max_digi
    data = [
        generate_datum(number_ranges, operators, max_count
        for _ in range(size)
    ]

    return pd.DataFrame(data)

```

- 產生字典

```

# 一個dict把中文字符轉化成id
char_to_id = {}
# 把id轉回中文字符
id_to_char = {}

# 有一些必須要用的special token先添加進來(一般用來做padding的tok
char_to_id['<pad>'] = 0
char_to_id['<eos>'] = 1
id_to_char[0] = '<pad>'

```

```

id_to_char[1] = '<eos>'

# 運算需要的文字
characters = [str(number) for number in range(0, 10)] + ops
if add_parentheses:
    characters.extend( ["(", ")"] )

for idx, ch in enumerate(characters):
    char_to_id[ch] = idx+2           # 開始於 2
    id_to_char[int(idx+2)] = ch      # index 為整數

vocab_size = len(char_to_id)
print('字典大小: {}'.format(vocab_size))
print(char_to_id)

```

- 資料切分與讀取至Torch，不同於使用rnn_padding的方法，我直接將所有向量用0補到最長算式的長度，就不用考慮複雜的額外參數。並用8:1:1的方式切分出訓練、驗證和測試資料。

```

# 把資料集的所有資料都變成id
df['formula_id_list'] = df['formula'].apply(lambda text: [char_to_id[c] for c in text])
df['answer_id_list'] = df['answer'].apply(lambda text: [char_to_id[c] for c in text])

# 而對於加減法的任務：
# input: 1 + 2 + 3 = 6
# output: / / / / 6 <eos>
# /的部分都不用算loss，主要是預測=的後面，這裏的答案是6，所以output
class Dataset(torch.utils.data.Dataset):
    def __init__(self, df):
        self.formula = df["formula_id_list"]
        self.answer = df["answer_id_list"]
        self.max_len = 0

        # 計算最長序列長度
        for f, a in zip(self.formula, self.answer):
            length = len(f) + len(a)
            if length > self.max_len:

```

```

        self.max_len = length

    def __getitem__(self, index):

        x = self.formula.iloc[index] + self.answer.iloc[index]
        # 等號以前的y設為padding可以讓損失函數失效(設定損失函數的index)
        y = [char_to_id['<pad>']] for _ in range(len(self))

        if len(x) < self.max_len:
            # 未滿最長序列需要補滿
            x = [char_to_id['<pad>']] for _ in range(self.max_len - len(x))
            y = [char_to_id['<pad>']] for _ in range(self.max_len - len(y))

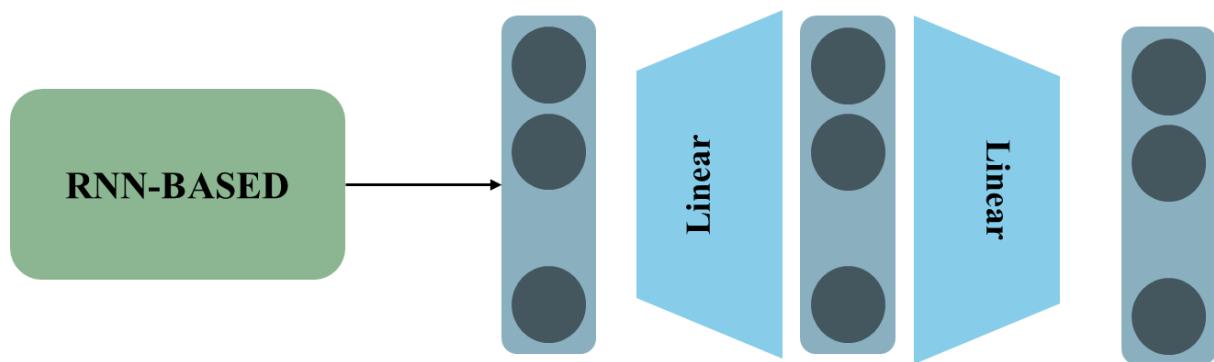
        return torch.tensor(x), torch.tensor(y)

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

```

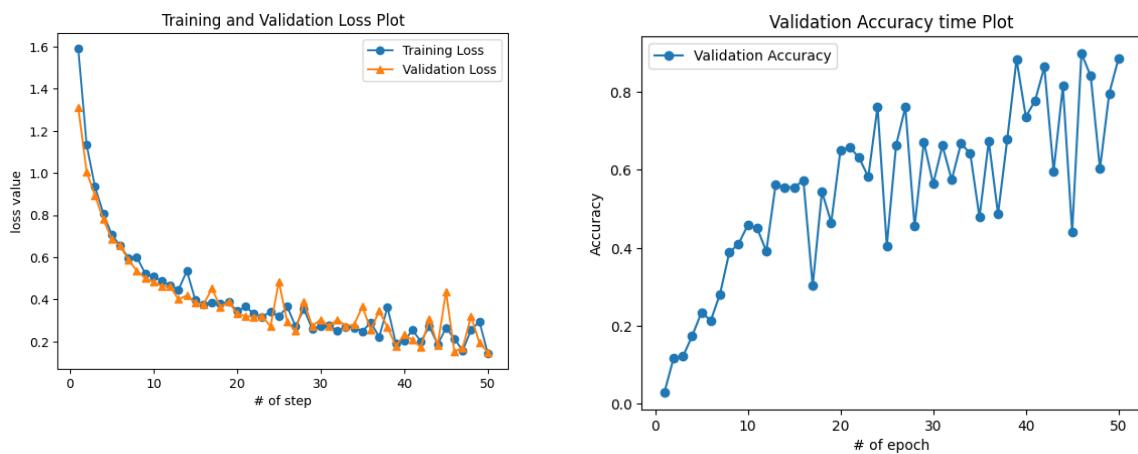
模型 (勘誤 → # of step 是 # of epoch → 圖表沒注意到)

僅調整RNN-BASED區塊，可以是RNN/LSTM/GRU，且僅堆疊一層時間序列模型，並將其輸出傳給兩個線性層以生成下一個字的機率分布。

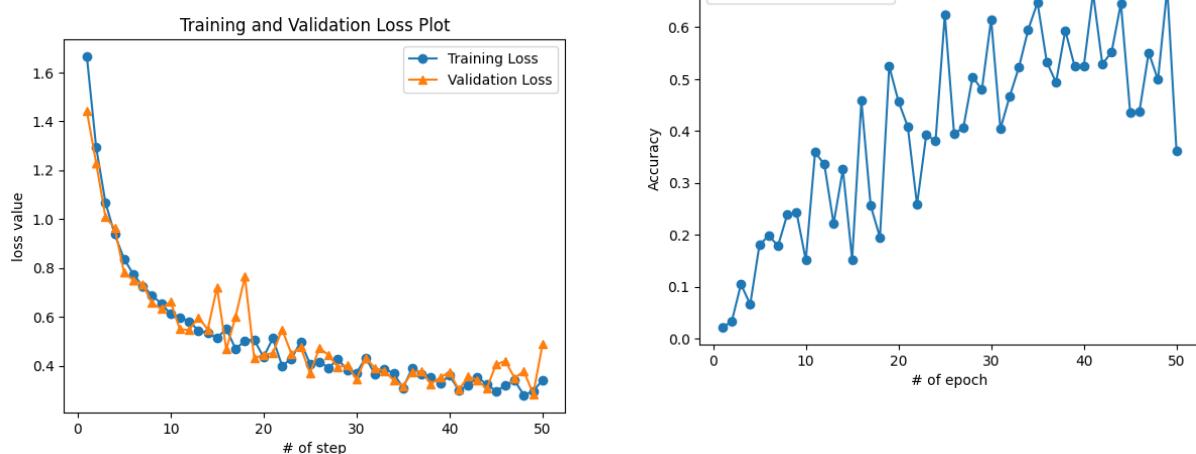


RNN

- 沒括號的資料

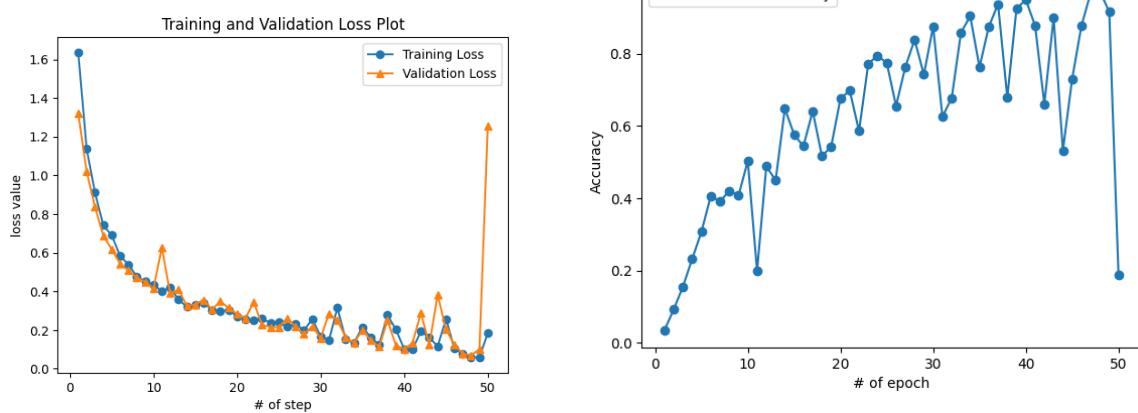


- 有括號的資料

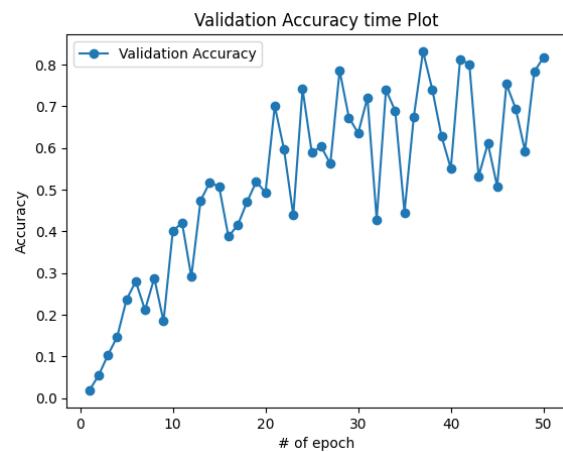
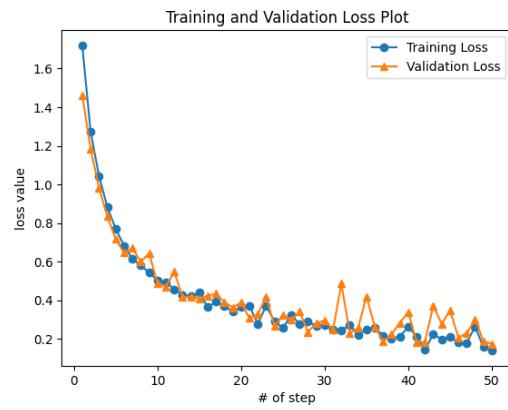


LSTM

- 沒括號的資料

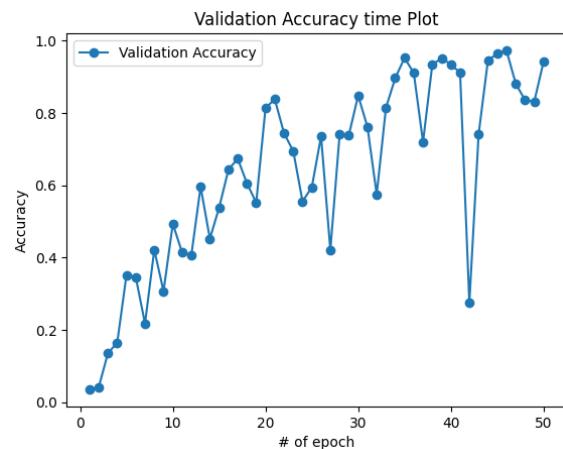
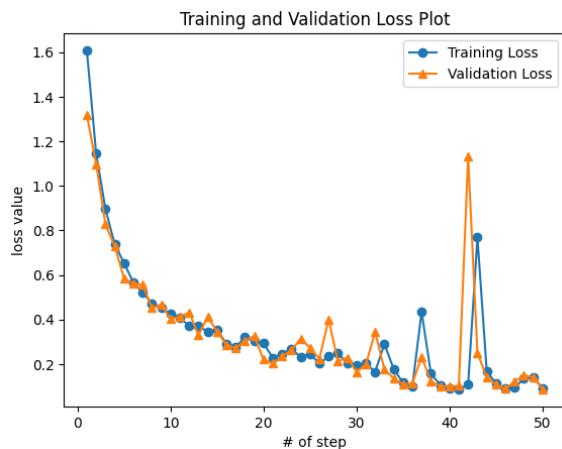


- 有括號的資料

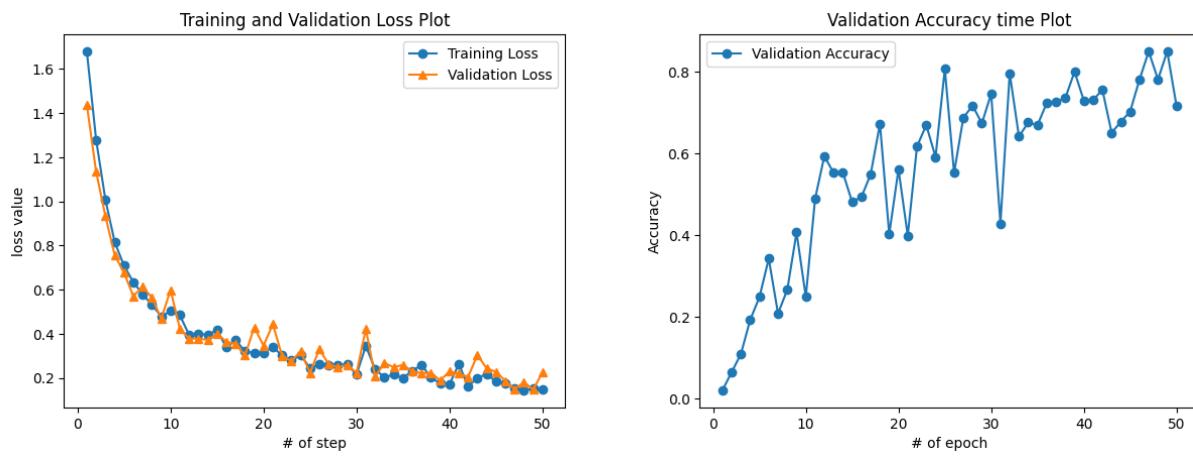


GRU

- 沒括號的資料



- 有括號的資料



模型綜合比較

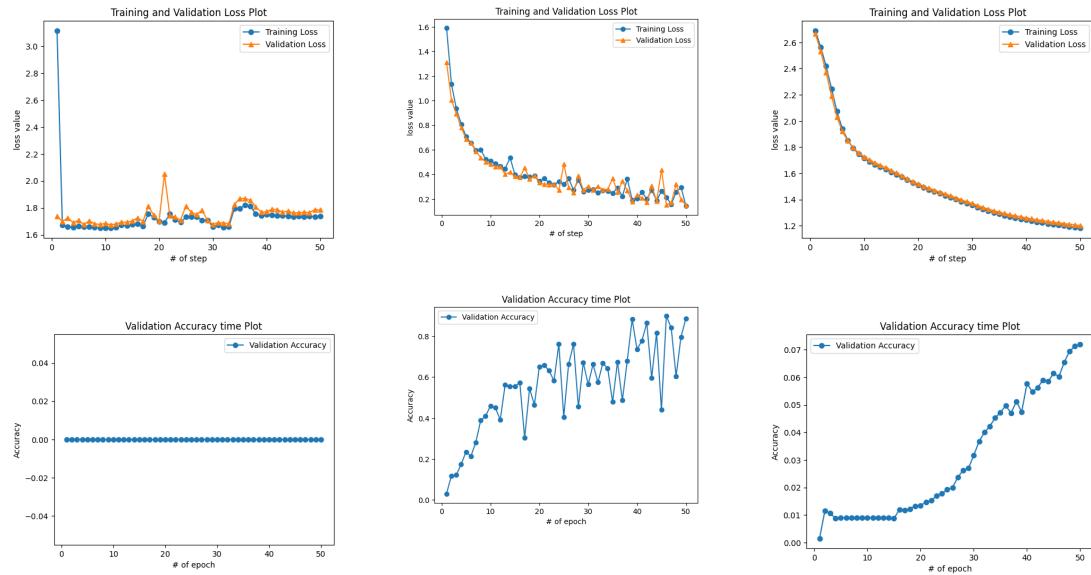
比較在測試資料上的準確度表現

準確度	RNN	LSTM	GRU
沒括號	87.25%	97.42%	93.67%
有括號	68.35%	81.23%	84.23%

探討

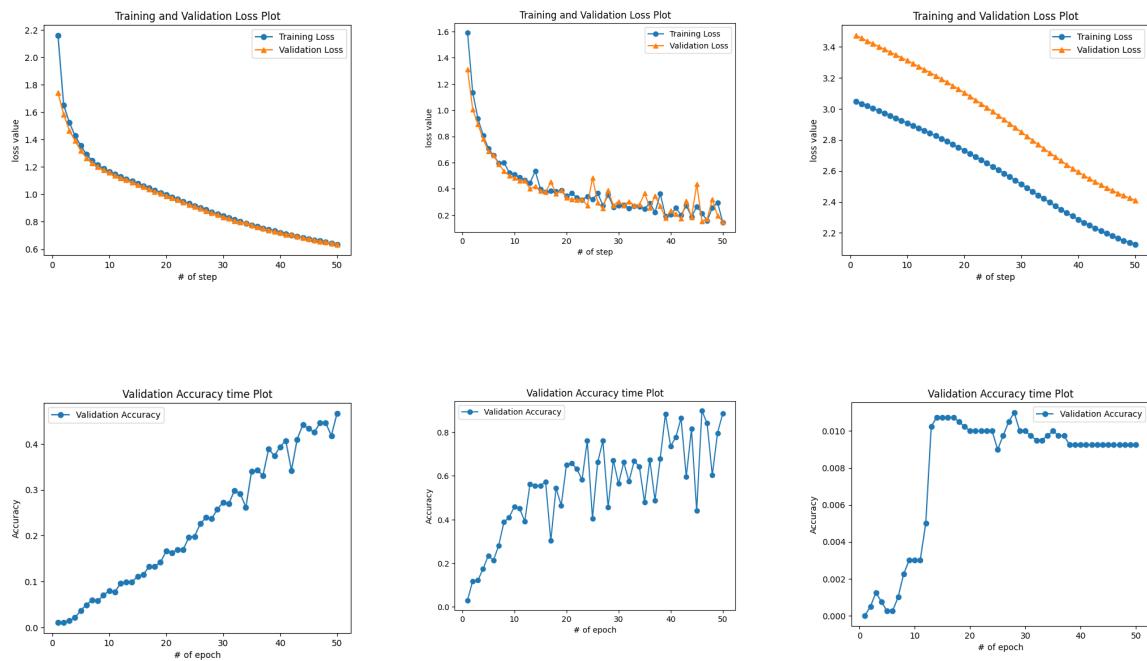
- 參數的影響力
 - Learning Rate

針對RNN我調整三種不一樣的學習率 (0.1, 0.001, 0.00001) 訓練於無括號之資料集上來看其差異。最左邊為學習率為0.1，最右邊為0.00001。學習率過大導致模型每一步長過大而在一個較佳解上徘徊，沒辦法朝較佳解收斂，因此損失函數幾乎不降而且準確度為0。較標準的學習率可以看到損失函數非常完美的下降，預測的準確度也隨之上升。學習率過低容易讓模型有非常好的收斂曲線，但是通常需要非常長的時間來收斂，而且也需要面對卡在局域最佳解的窘境，從圖可看出，曲線似乎還有再繼續收斂的傾向，也就是加大epoch數有機會讓模型收斂到更好的解但也相對耗時。



◦ Batch Size

針對RNN我調整三種不一樣的 batch_size (50, 512, 5120) 訓練於無括號之資料集上來看其差異。最左邊為batch_size=50，最右邊為batch_size=5120。batch_size過低。較小的batch_size要訓練很久，理論上較難收斂，此處的例子較看不出。較標準的batch_size可以看到損失函數非常完美的下降，預測的準確度也隨之上升。batch_size過大計算時間通常較快，但是可能會收斂到很差的解，泛化能力差。



- 模型影響力

- 從模型大小和訓練時長來看 $RNN < GRU < LSTM$
- 模型表現來看 $RNN < GRU \sim LSTM$
- 在我看來表現變好最主要的原因是因為參數量增加，但並不是漫無目的增長，
GRU和LSTM都使用了一些**有效的參數增加**來使模型變強。最主要就是設計
Gate允許模型有一定機率遺忘和記憶，一方面減緩了梯度下降的問題，一方
面在此次資料也能看出，較長的資料集(即有括號者)RNN表現會相對很差，**因
為梯度消失導致算式後者與前者的關係會在計算梯度時被影響而表現差。**反觀
GRU和LSTM就相對穩健，因為其模型的gate能讓他們更能有能力去推理近期
與遠期的依賴關係，而且較不容易出現梯度消失問題，至於誰好誰壞似乎互有
利弊，其中也涵蓋一些隨機性。

- 資料集差異

我僅僅更改資料集是否含有括號就可以發現模型表現皆掉了一個層級，我覺得主
要的原因為長度增加導致梯度問題外，模型也需要能"回顧"上一個括號在哪裡。從
搜尋的角度來看，模型的整個搜索空間、輸入端的組合數、輸出端的組合數都變
大了，所以模型的performance也應當下降。