# Assignment1

李鹏宇

October 16, 2024

## Contents

# 1 Task 1: Python 基础

本任务包括两个部分：flatten list 和 char count，每一个问题都实现了两种不同的算法，对它们的效率进行比较。

## 1.1 Flatten list

### 1.1.1 Algorithms

我分别基于 for-loop、列表推导式和 extend 方法实现了三种算法：

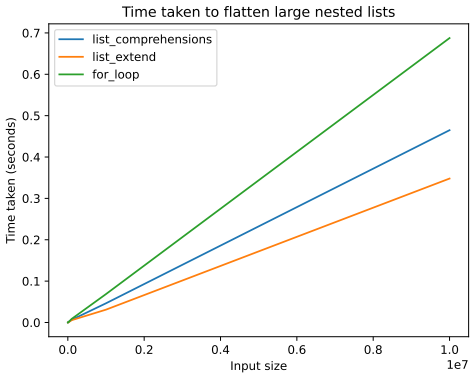
```
#Using for-loop
def flatten_list_for(input_list):
    result = []
    for sublist in input_list:
        for item in sublist:
            result.append(item)
    return result

# Using python list comprehension
def flatten_list_list(nested_list: list):
    return [item for sublist in nested_list for item in sublist]

# Using extend method
def flatten_list_extend(nested_list: list):
    result = []
    for sublist in nested_list:
        result.extend(sublist)
    return result
```

### 1.1.2 Experiments

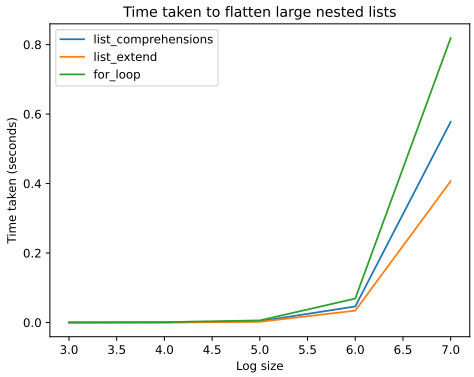
通过对总元素数量为 $10^3, 10^4, 10^5, ..., 10^7$ 的列表进行展平，记录算法耗时，得到如下结果 (table1, figure1)。这表明三种算法都是 O(N) 复杂度，但是相比较使用 list 的 extend 方法效率最高，其后是 list comprehension 和 for-loop。

| Input Size | list_comprehensions (s) | list_extend (s) | for_loop (s) |
|---|---|---|---|
| $10^3$ | 0.000000 | 0.000000 | 0.000000 |
| $10^4$ | 0.000000 | 0.000000 | 0.000000 |
| $10^5$ | 0.003000 | 0.002000 | 0.005002 |
| $10^6$ | 0.044864 | 0.033000 | 0.064002 |
| $10^7$ | 0.520547 | 0.428107 | 0.742279 |

Table 1: Comparison of Timing for Different Methods on flatten list at Various Input Sizes




(a) Plot over input size.




(b) Plot over log input size.

Figure 1: Compare three methods on flatten list task.

## 1.2 Character count

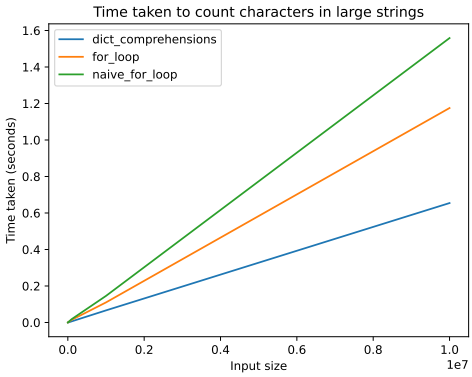
### 1.2.1 Algorithms

我分别基于字典推导式、使用内置方法的 for-loop 和 naive for-loop 方法实现了三种算法：

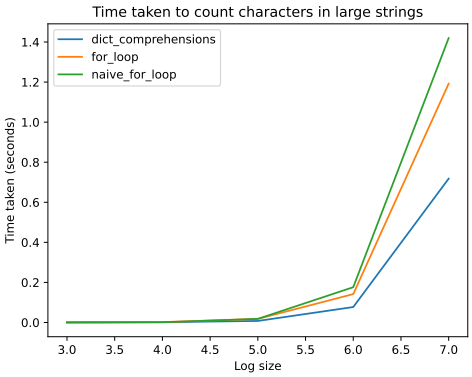
```python
# Using dictionary comprehension
def char_count_dict(input_str: str):
    return {char: input_str.count(char) for char in set(input_str)}

# Using for-loop with builtin methods
def char_count_for(input_str: str):
    char_count = {}
    for char in input_str:
        char_count[char] = char_count.get(char, 0) + 1
    return char_count

# Using naive for-loop
def char_count_naive_for(input_str: str):
    char_count = {}
    for char in input_str:
        if char in char_count:
            char_count[char] += 1
        else:
            char_count[char] = 1
    return char_count
```

### 1.2.2 Experiments

我下载了 wiki 的示例数据，将文本切分成 $10^3, 10^4, 10^5, ..., 10^7$ 的 slice，作为 character count 的测试，得到如下结果 (table2, figure2)。这表明三种算法都是 O(N) 复杂度，但是其中使用字典推导式的算法最快，而使用的 get 方法的 for-loop 要快于 naive for-loop 实现。




(a) Plot over input size.




(b) Plot over log input size.

Figure 2: Compare three methods on char count task.

| Input Size | dict_comprehensions (s) | for_loop (s) | naive_for_loop (s) |
|---|---|---|---|
| $10^3$ | 0.000000 | 0.000000 | 0.000000 |
| $10^4$ | 0.000000 | 0.002002 | 0.001367 |
| $10^5$ | 0.006053 | 0.014441 | 0.018172 |
| $10^6$ | 0.066294 | 0.109994 | 0.145695 |
| $10^7$ | 0.654463 | 1.175184 | 1.558554 |

Table 2: Comparison of Timing for Different Methods on char count at Various Input Sizes

# 2 Task 2: 基于 CNN 的文本分类

在本任务中，我实现了一个卷积神经网络（CNN）用于中文句子的分类。首先，根据文本进行分词，建立词表，并转换为 one-hot 编码。CNN 网络包含嵌入层、卷积层、最大池化层和全连接层。在训练时，我采用 early stopping, 设置 patience 为 5，当验证集上的损失连续五次不减后停止训练。

## 2.1 模型配置

我的 CNN 模型配置如下：

- Embedding Layer：词向量维度为 100
- Convolution Layer：使用三种不同卷积核大小（3, 4, 5），每种 10 个卷积核
- Activation Function：ReLU
- Max Pooling
- Dropout：0.5
- Fully Connected Layer：输出 4 种类别
- Patience：5

### 2.2 实验结果

在训练集分类准确率 93%, 验证集上的分类准确率为 77%。在验证集上使用 early stopping 达到了最优模型，避免了过拟合。最终在测试集上的准确率为 78%。

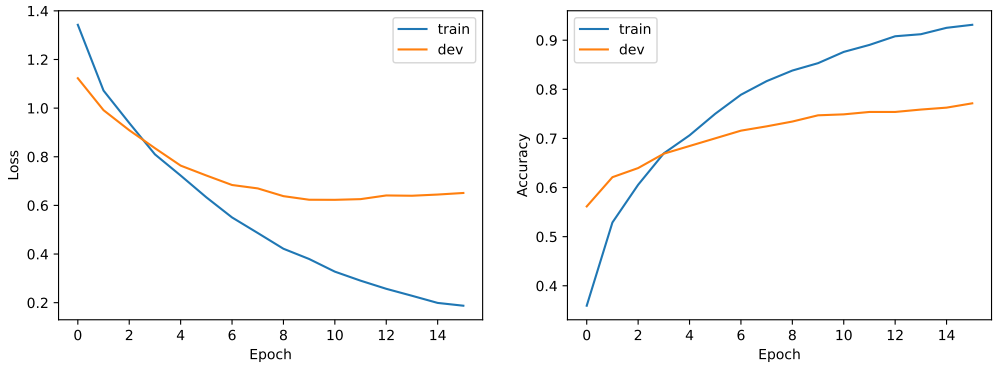



Figure 3: Caption

| Epoch | Train Loss | Train Acc | Dev Loss | Dev Acc |
|---|---|---|---|---|
| 0 | 1.3426 | 0.3594 | 1.1230 | 0.5615 |
| 1 | 1.0722 | 0.5289 | 0.9916 | 0.6211 |
| 2 | 0.9391 | 0.6056 | 0.9092 | 0.6396 |
| 3 | 0.8099 | 0.6697 | 0.8353 | 0.6689 |
| 4 | 0.7229 | 0.7063 | 0.7637 | 0.6846 |
| 5 | 0.6333 | 0.7502 | 0.7230 | 0.7002 |
| 6 | 0.5505 | 0.7890 | 0.6839 | 0.7158 |
| 7 | 0.4866 | 0.8167 | 0.6699 | 0.7246 |
| 8 | 0.4218 | 0.8382 | 0.6380 | 0.7344 |
| 9 | 0.3796 | 0.8532 | 0.6233 | 0.7471 |
| 10 | 0.3276 | 0.8762 | 0.6229 | 0.7490 |
| 11 | 0.2905 | 0.8904 | 0.6258 | 0.7539 |
| 12 | 0.2570 | 0.9080 | 0.6409 | 0.7539 |
| 13 | 0.2284 | 0.9122 | 0.6397 | 0.7588 |
| 14 | 0.1990 | 0.9252 | 0.6446 | 0.7627 |
| 15 | 0.1875 | 0.9314 | 0.6511 | 0.7715 |

Table 3: Training and Validation Loss and Accuracy

# 3 Task 3: 基于 RNN 的机器翻译

本任务中，我实现了一个基于 LSTM 和注意力机制的 RNN 模型，用于日文到英文的机器翻译。

## 3.1 Word2Vector

首先，我用空格对英语文本进行分词，用 sudachipy 对日语进行分词，然后基于文本构建词汇表。

```
class EngLang:
    def tokenize(self, sentence):
            return sentence.split(' ')

class JpnLang:
    def tokenize(self, sentence):
            tokens = self.tokenizer_obj.tokenize(sentence, self.mode)
            return [token.surface() for token in tokens]
```

其次，我分别采用 Cbow 和 Skip-gram 算法训练 word embedding。最后，我通过将词向量映射到二维平面、计算相似度、利用现有相似度数据集测试等方法，评估训练得到的词嵌入。从表格4可以看出，相似的词汇之间具有很高的相似度。但是去除 OOV 之后，在 wordsim 数据集上测试相似度和相关性的表现都不好 (table5)，这可能是因为用于训练的数据太少，不足以捕捉词汇间更精细的关系。

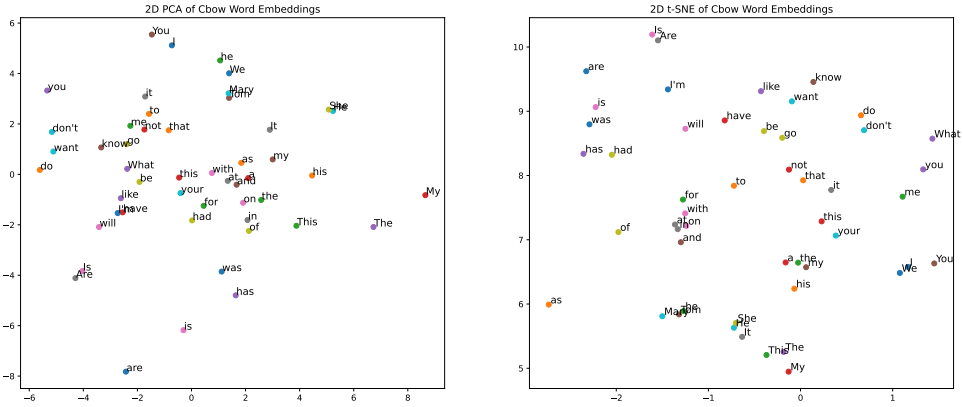



Figure 4: Embedding trained by Cbow

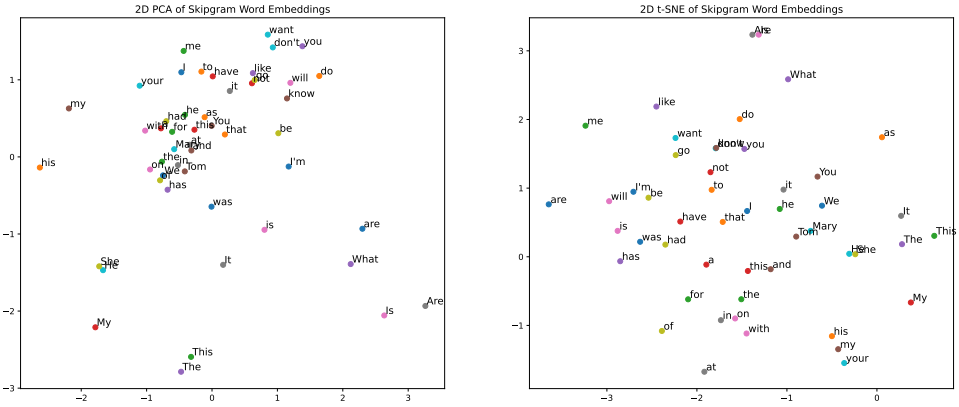



Figure 5: Embedding trained by Skip-gram

## 3.2 模型配置

RNN 模型配置如下：

| Word Pair | Similarity |
|---|---|
| (Tom, Mary) | 0.7510916 |
| (He, She) | 0.9654538 |
| (in, on) | 0.8808856 |

Table 4: Similarity Scores Between Word Pairs

| Model | Similarity | Relatedness |
|---|---|---|
| skipgram | 0.1071 | 0.0167 |
| cbow | 0.1304 | 0.0546 |

Table 5: Spearman Correlation for Different Models

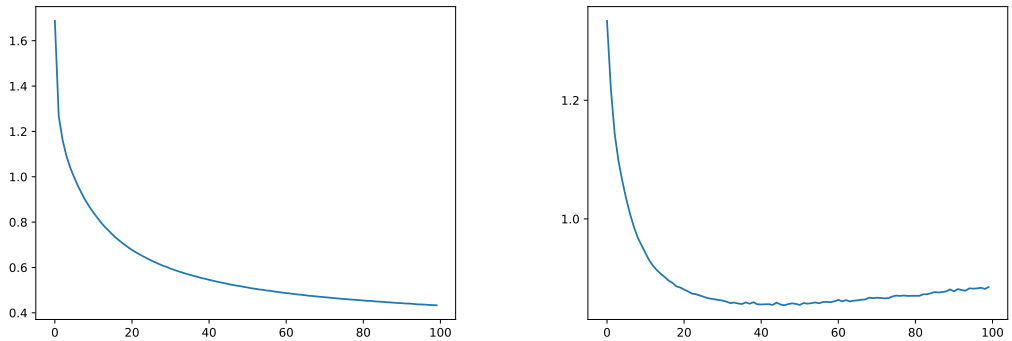
- Embedding Layer：使用 Cbow 算法训练的词向量，维度为 100
- LSTM：单层 LSTM，隐藏层维度为 100
- Attention：Bahdanau attention
- Fully Connected Layer：(hidden_size, output_size) 即 (100, 100)
- Dropout: 0.1

## 3.3 训练与测试

当不采用 early stopping 的时候，训练集和验证集上的损失函数曲线见图6，可以看出 validation loss 在后续 epoch 上出现上扬趋势，说明训练出现过拟合现象。于是，我添加了 early stopping 机制，发现在第 35 个 epoch 模型训练就停止了 (figure7)。观察模型在验证集上的 bleu score 和 perplexity 变化 (figure8)，可以发现在这些评价指标上的表现逐步提高。最后在测试集上，得到 Loss = 0.79, BLEU = 0.049, PPL = 2.22。

## 3.4 案例分析

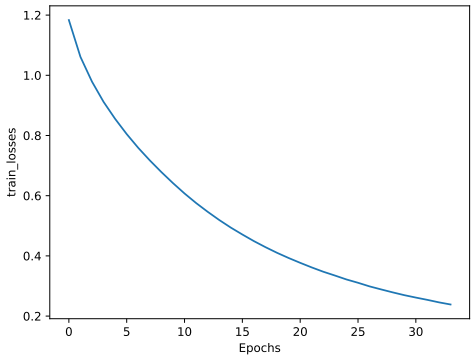
以下是几个翻译案例的输出：



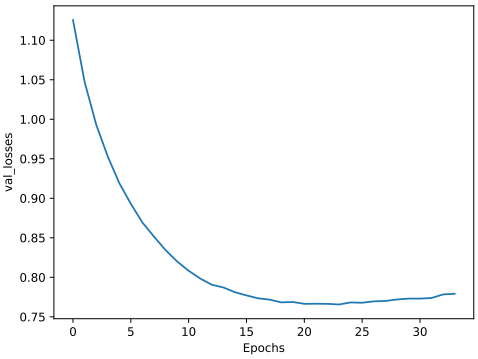

(a) Training loss.

(b) Validation loss.

Figure 6: Train without early stopping.

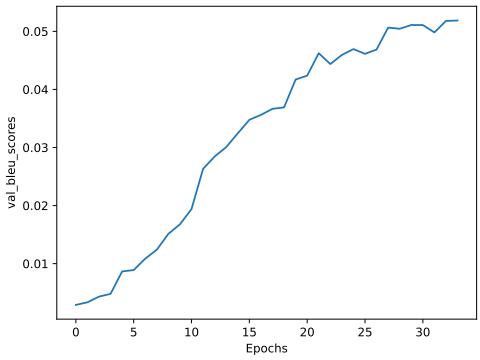



(a) Training loss.

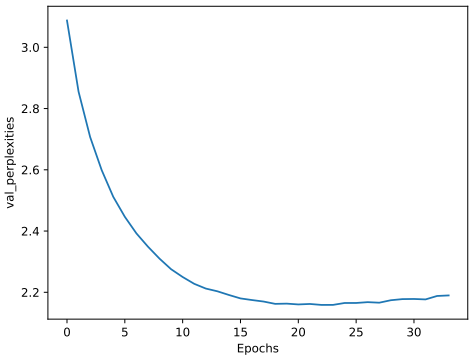



(b) Validation loss.

Figure 7: Train with early stopping.




(a) BLEU score.




(b) Perplexity.

Figure 8: BLEU and Perplexity on validation set

- Case 1: 私の名前は愛です -> My name is orange. <EOS>
- Case 2: 昨日はお肉を食べません -> I don't eat meat tonight. <EOS>
- Case 3: いただきますよう -> I try. <EOS>
- Case 4: 秋は好きです -> Fall is delicious. <EOS>
- Case 5: おはようございます -> How's your date you? <EOS>

从中可以看出，我们的模型正确翻译了部分单词，一些单词尽管错误，但是仍比较相关，句子整体比较可读。模型能力较差的原因，一则可能是我们用于训练的数据太少，导致 word embedding 效果不好，一则可能是模型架构简单，可以考虑改变 embedding dim、更多层 lstm 与其他 attention 机制。