DL-NLP Assignment1 Report

本实验报告为 2024 年秋季学期《基于深度学习的自然语言处理》课程第一次作业实验报告，作者为 2200017416 康子熙，本报告所有代码与第三问的所有模型均可以在我的 github 仓库中找到。

一、 Task 1

1.1 函数解答

对于 flatten_list 函数，考虑链表的嵌套层数对于不同的 list 是不相同的，所以应该使用递归的思路，在编写代码过程中，考虑到 python 有自带的递归层数限制，所以我使用一个 stack 存储了所有待处理的元素，具体代码如下：

```python
def flatten_list(nested_list: list):
    stack = list(reversed(nested_list))
    flattened_list = []

    while stack:
        item = stack.pop()
        if isinstance(item, list):
            stack.extend(reversed(item))
        else:
            flattened_list.append(item)

    return flattened_list
```

对于 char_count 函数，可以将整个字符串建模成为一个字典，字典的 key 表示字母，value 代表字幕出现的次数，最后通过返回字典即可，具体代码如下：

```python
def char_count(s: str):
    set_s = dict()
    for char in s:
        if char in set_s:
            set_s[char] += 1
        else:
            set_s[char] = 1
    return set_s
```

1.2 时间复杂度分析

对于 flatten_list 函数，虽然使用了递归，但是整个栈对于每个元素其实只考虑了一次，对于长度为 m，展开后长度为 n 的 list 来说，时间复杂度为 $O(n)$。对于 char_count 函数，时间瓶颈在于对 string 进行的循环，对于长度为 n 的字符串来说，时间复杂度为 $O(n)$。

为了验证数学推导，我针对两个函数处理不同长度的数据分别进行了测试，测试结果显示，随输入的长度增大，函数运行的时间成本基本呈线性的增长，这和数学推导出的时间复杂度一致。

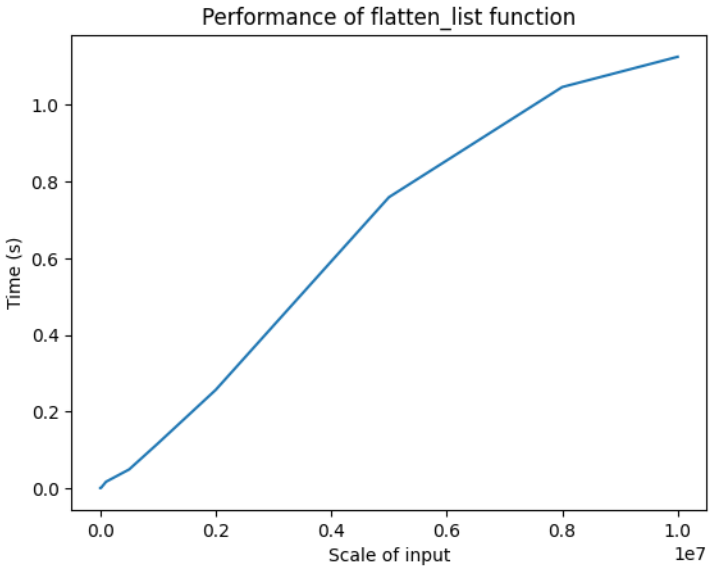

图 1　对 flatten_list 函数的时间复杂度实验

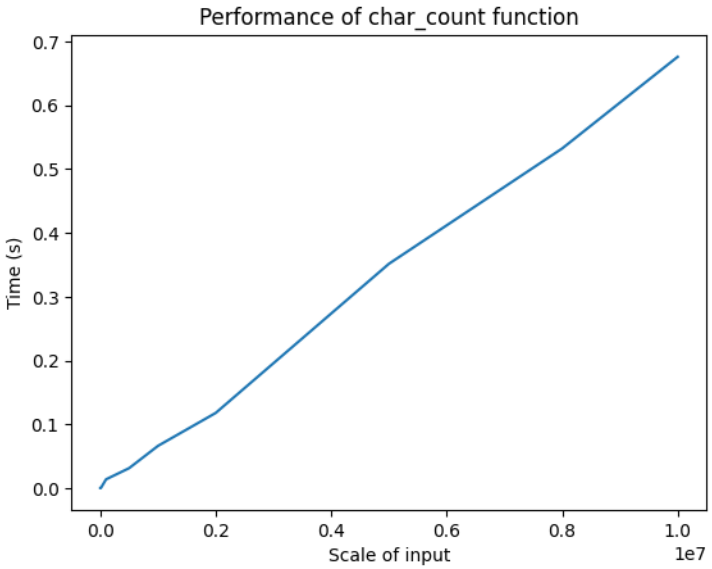

图 2　对 char_count 函数的时间复杂度实验

二、 Task 2

2.1 训练过程

在这一任务中，我尽可能的复现了这篇论文中的模型结构。具体模型包括一个 embedding 层，一个 120 个大小不同的 filter 组成的卷积层，一个沿时间维度的池化层以及一个全连接层。同时还在验证集上运用了 early stopping 的逻辑，发现模型在第三个 epoch 中已经出现了过拟合的现象，停止后在测试集上的准确率达到了 78.17%。

由于我在第一次训练时达到了助教设定的 75%，所以没有进行超参数的调整，模型部分参数如下：

- 词嵌入输出维度为 256
- 1D 卷积核大小为 1、2、3、4、5、6，每种卷积核 20 个，共 120 个
- 学习率为 1e-3，dropout 设置为 0.1
- 优化器为 Adam 优化器，损失函数为交叉熵损失函数

2.2 效果分析

模型在训练 3 个 epoch 之后由于早停停止了训练，Test Loss 为 0.954，Test Acc 为 78.17%，达到了指标要求。

由于训练的 epoch 和超参数调整均较少，无法进行详细的实验分析，训练过程中的损失和准确率如图所示：

```
Epoch: 01
        Train Loss: 1.102 | Train Acc: 63.44%
        Val Loss: 1.026 | Val Acc: 70.63%
Epoch: 02
        Train Loss: 0.882 | Train Acc: 86.24%
        Val Loss: 0.953 | Val Acc: 79.07%
Epoch: 03
        Train Loss: 0.816 | Train Acc: 92.75%
        Val Loss: 0.958 | Val Acc: 77.68%
Early stopping at epoch 2
Test Loss: 0.954 | Test Acc: 78.17%
```

图 3　训练 log

三、 Task 3

这个任务需要我们处理一个从日语到英语的翻译任务，为简化分词，对于日语，我使用了 WeCab 进行分词，对于英语则使用 nltk 库进行分词。在预训练和训练中分词的方法保持了一致。

3.1 预训练过程

由于任务提供的数据集较少，CBOW 适用于处理数据集较小的任务，所以在于训练过程中我使用了 CBOW 的方法捕捉语义。

对于 CBOW 模型，我将其设置为一个 embedding 层和一个全连接层，全连接层的输出结果应当和所要预测的中间词 one-hot 编码一致，故使用交叉熵损失函数。在数据处理方面，考虑到无论是英语还是日语，句子长度均较短，所以将 window_size 设置为了 2，即向前向后各考虑两个子词，如果有边界情况则使用 padding 填充。在模型具体参数方面，我将 word2vec 输出维度设置为了 256，batch size 设置为 64，学习率为 1e-3，使用 Adam 优化器，共训练 10 个 epoch。

我使用测试集对预训练的 CBOW 模型进行了测试，具体结果如下表格所示：

表 1　Model Performance on Training and Evaluating Data

Model	Training Accuracy (%)	Evaluating Accuracy (%)
CBOW English Model	58.39	43.82
CBOW Japanese Model	61.56	48.39

上述结果证明了预训练达到了很好的捕捉语义的效果，由于测试集中有一些子词其实是没有出现过的，只使用训练集进行预训练达到这样的效果已经很好了，我没有对预训练模型进行更多的参数调整改进效果。

3.2 训练过程

我编写了一个具有 attention module 的，encoder-decoder 架构的 LSTM 模型，用于处理日语-英语翻译任务。其中 encoder 和 decoder 的词嵌入均各自使用了预训练的日语 CBOW 和英语 CBOW。

encoder 的结构包括了一个日语的 embedding 层，一个双向 LSTM，将 hidden 和 cell 两个状态作为 decoder 的 hidden 和 cell 的初值，并且保存了 encoder 的 output，由于做加权注意力处理。

attention module 包括了一个 MLP，MLP 包括两个 FC 层和 tanh、softmax 两个非线性变换，输出为长度为序列长度的 attention score，表示了对 encoder output 的注意力。

decoder 的结构包括了一个英语 embedding 层，一个单向 LSTM，一个 attention module 和一个 FC 层。decoder 的 LSTM 处理的输入是经过 embedding 的英语 token 和 encoder output 结合在一起形成的 feature。attention module 对 encoder output 进行了加权处理，使 decoder 能够更好的捕获来自于 encoder output 的信息。

具体结构如下图所示：

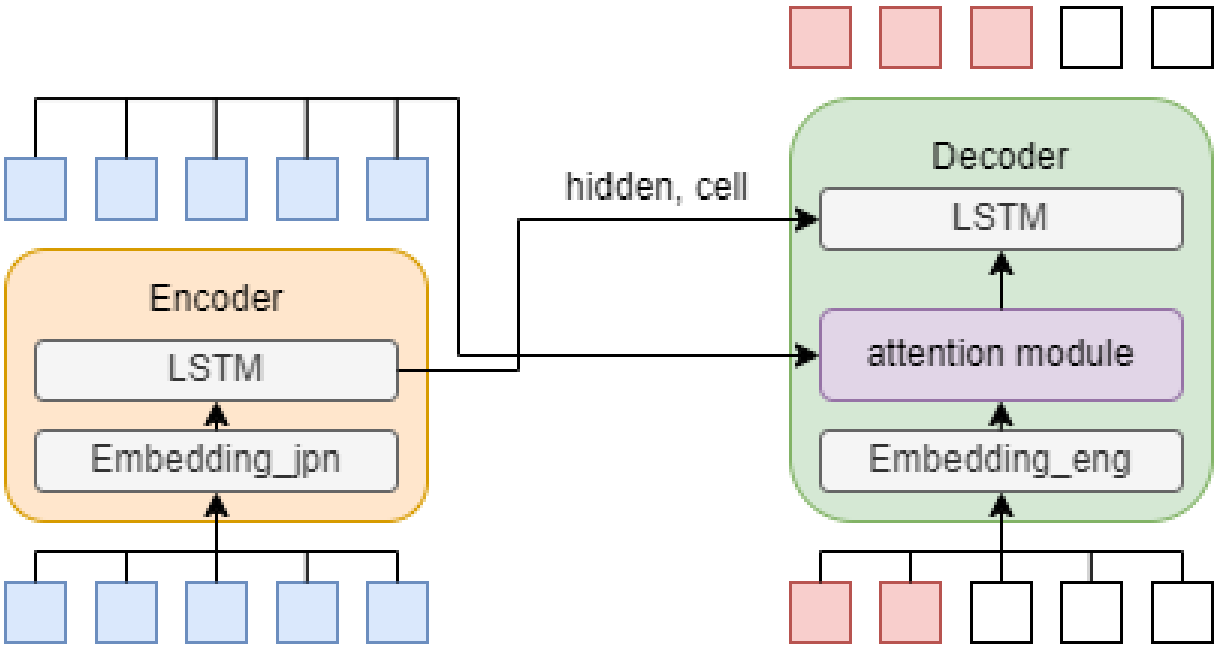

图 4　模型结构

在参数方面，embedding 输出维度为 256，LSTM 隐藏层维度为 512，encoder 和 decoder 的 LSTM 均只有一个 layer，使用单向而非双向，损失函数为交叉熵损失函数，优化器为 Adam 优化器，初始学习率为 1e-3。在此基础上我训练了 10 个 epoch。

3.3 效果分析

在早期训练时，我注意到虽然模型在早期训练阶段的表现相对较好，随着训练的继续，模型的收敛速度显著下降，且在验证数据集上的 BLEU 分数和困惑度指标没有达到预期的提升。为此，我决定将 LSTM 的隐藏层维度从 256 增加到 512，以增强模型的学习能力和表达能力。

此外，我还调整了学习率的调度策略。最初，我采用了固定的学习率 1e-3，但为了更好地适应训练过程中的动态变化，我引入了学习率衰减机制。

针对模型在长序列处理上的不足，我还尝试在注意力机制中添加了 tanh 的非线性层。

同时，我还将 encoder 设置为双向的 LSTM，使 encoder 可以更好地捕获日语句子 token 之间的关系，增强模型的效果。

模型在训练集、验证集、测试集上的效果如下表：

表 2　Model Performance on Different Datasets

Dataset	BLEU Score	Perplexity
Training Data	0.7669	2.4233
Validating Data	0.6503	5.9132
Evaluating Data	0.6567	5.9333

对于助教给出的样例，我的模型的输出是：

```
Predicted:  ['My', 'name', 'is', 'love', '.', '<pad>', '<pad>', '<pad>', '<pad>', '<pad>']
Predicted:  ['I', 'do', "n't", 'have', 'much', 'meat', 'yesterday', '.', '<pad>', '<pad>']
Predicted:  ['I', 'would', 'you', '.', '<pad>', '<pad>', '<pad>', '<pad>', '<pad>', '<pad>']
Predicted:  ['I', 'like', 'the', '.', '<pad>', '<pad>', '<pad>', '<pad>', '<pad>', '<pad>']
Predicted:  ['Good', 'morning', '.', '<pad>', '<pad>', '<pad>', '<pad>', '<pad>', '<pad>', '<pad>']
```

图 5　样例输出

由此可见，模型的输出与翻译软件给出的答案比较相近，翻译软件给出的答案是：

- My name is love.
- I didn’t eat meat yesterday.
- Thank you.
- I like autumn.
- Good morning.

但是模型仍然有一些不足，这可能是因为参数以及训练方法导致的，可以通过更多的手段改进模型的效果。