自然语言处理技术报告

袁昭新 2020K8009926029

目录

一、简介

本报告旨在通过使用基于 FNN、RNN 和 LSTM 三种模型，获取汉语和英语词向量，并对它们进行比较。本报告选取北京大学标注的《人民日报》1998 年 1 月份的分词语料和网络爬虫收集的小说和新闻英文文本语料，来获取汉语和英语词向量，并利用它们进行相似性计算，并对比三种模型获得的词向量的差异。 最后，从每份语料中抽取 20 个单词，计算与其词向量最相似的前 10 个单词（见附件）。

二、数据集

汉语：北京大学标注的《人民日报》1998 年 1 月份的分词语料。

英语：通过网络爬虫获取的小说、新闻英文文本语料。

	中文	英文	英文
类别	新闻	小说	新闻
来源	《人民日报》1998年1月	Full English Books	ABC News, CNN
规模	8.42MB	147.7MB	36.1MB
清洗后规模	3.95MB	87.6MB	34.0MB

表1

三、数据预处理

（1）中文

中文语料是已经进行分词和标注的语料，但在该任务中不需要用到其中的标注，因此将所有的词性标注删除。

```
# 删除任何非汉字符号
# 定义正则表达式，匹配除了中文以外的任何字符
pattern = re.compile(r'[^\u4e00-\u9fa5]')

# 逐行读取输入文件内容，并将每行繁体字转换为简体字，然后写入输出文件
for line in input_file:
    # 将 line 中的任何非汉字符号替换为空格
    cleaned_line = re.sub(pattern, ' ', line.strip())
    # 将连续的空格替换为一个空格
    output_line = re.sub(r'\s+', ' ', cleaned_line)
    if output_line.strip() != '':
        output_file.write(output_line.strip() + '\n') # 将换行符写入输出文件
```

(2) 英文

对于英文语料，处理后的文件每行是一个句子，句子中的单词以空格分割。由于语料是使用网络爬虫获取的，难免会有不规范的句子，包括前导空格以及罗马尼亚语字符等非英文字符。因此，将长度小于 3 的、含有非英文和非标点字符的句子删除。

```
# Loop through each line in the input file
for line in f_in:
    # Remove leading spaces, brackets and their contents
    line = re.sub(r'^\s*\([^)]*\)\s*', '', line)
    # Split the line into sentences using regex
    sentences = re.split(r'[.?!""“”]+', line.strip())
    for sentence in sentences:
        # Split the sentence into words
        words = sentence.split()
        # Check if the number of words is greater than or equal to 3
        if len(words) >= 3:
            if re.search(r'[^a-zA-Z.,!?;:\'"()\[\]{}<>\n\s]', sentence):
                continue
            # delete leading spaces
            sentence = sentence.strip()
```

```
            # Convert the sentence to lowercase
            sentence = sentence.lower()
            # delete non-eng chars
            sentence = re.sub(r'[^a-z\'\s]', '', sentence)
            # Write the cleaned sentence to the output file
            f_out.write(sentence + '\n')
```

四、方法

利用以下三种基于神经网络的模型，FNN、RNN 和 LSTM，来获取词向量。

（一）FNN

FNN（Feedforward Neural Network）是一种具有前向传播的神经网络模型。本次作业采使用 `pytorch` 构建 `FNN` 模型以获取词向量。

1. 模型构建

构建一个文本预测的 `FNN` 模型。模型包含三个主要组件：嵌入层 (embedding layer）、隐藏层 (fc1) 和全连接层 (fc2）。其中嵌入层用于将输入数据转换为向量形式以利于神经网络对其进行处理，随后的两个全连接层负责对向量进行权重调整和输出预测结果。

嵌入层将每个输入转化为一个向量，最终我们仅需要嵌入层的权重，该权重就是每个词的词向量。

```
class FNN(nn.Module):
    def __init__(self, vocab_size, embedding_size, hidden_size):
        super(FNN, self).__init__()
        self.vocab_size = vocab_size
        self.embedding_size = embedding_size
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(vocab_size, embedding_size)
        self.fc1 = nn.Linear(embedding_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, vocab_size)
    def forward(self, inputs):
        embedding = self.embedding(inputs)
        out = F.relu(self.fc1(embedding))
```

```
        out = self.fc2(out)
        return F.log_softmax(out, dim=-1)
```

2. 模型训练

(1) 训练数据准备

对于神经网络，需要将单词映射到一个数值以便输入和训练。为此，建立一个词汇表，将单词作为 key, 索引得到的值作为训练使用的数值。另外，设置词汇表大小的上限 `VOCAB_SIZE`, 超过词汇表上限的词用 `<UNK>` 替代，以减轻硬件压力。相关代码如下：

```
# read corpus from file
corpus = []
with open('v3-en-norvel-fulleng-splited.txt', 'r', encoding='utf-8') as f:
    for line in f:
        corpus.append(line.strip().split(' '))
# build word_to_ix
word_to_ix = {}
for sentence in corpus:
    for word in sentence:
        if word not in word_to_ix:
            word_to_ix[word] = len(word_to_ix)
word_to_ix = {k: v if v < VOCAB_SIZE else VOCAB_SIZE for k, v in
word_to_ix.items()}
word_to_ix['<UNK>'] = VOCAB_SIZE
```

本次采用 skip-gram 方法进行训练，即在语料中选取一个词作为训练输入，将这个词长度为 `WINDOW_SIZE` 的上下文作为训练的目标。

例如，假设 `WINDOW_SIZE` 为 2。如果用 `[input, target]` 的方式表示一对训练输入和训练目标，则对于句子“迈向 充满 希望 的 新 世纪”，以“迈向”作为输入时，可以构建 `[迈向，充满]` `[迈向，希望]` 两组输入输出；以“希望”作为输入时，可以构建 `[希望，迈向]` `[希望，充满]` `[希望，的]` `[希望，新]` 四组输入输出。

构建训练数据的代码如下：

```
inputs = []
targets = []
for sentence in corpus:
    for i in range(len(sentence)):
        for j in range(-WINDOW_SIZE, WINDOW_SIZE+1):
            if j != 0 and i+j >= 0 and i+j < len(corpus):
                inputs.append(word_to_ix[corpus[i]])
                targets.append(word_to_ix[corpus[i+j]])

# convert to tensor
inputs = torch.tensor(inputs, dtype=torch.long).to(device)
targets = torch.tensor(targets, dtype=torch.long).to(device)
```

(2) 训练

在训练时分别使用了 NLLLoss 和 CrossEntroypyLoss 两种损失函数，在其他参数相同的情况下，对比训练后词向量的效果，即计算与 *years* 词向量最相似的前十个单词：

NLLLoss	Similarity	CrossEntropyLoss	Similarity
months	0.93846625	minutes	0.848302
days	0.93782306	percent	0.8303785
decades	0.9371136	ways	0.81471
times	0.9254089	decades	0.81293297
weeks	0.9113575	times	0.81205803
hours	0.87781966	yards	0.7987586
minutes	0.86304075	months	0.7943185
percent	0.8360166	thoughts;	0.79224014
seconds	0.82983637	incapable	0.78430146
paragraphs	0.8276598	babies	0.77159023

表2

根据生活常识，在使用 NLLLoss 时，获得的词向量最相近的 10 个词中有 8 个都是与 years 很类似的日期、时间，而使用 CroosEntropy 时仅有 4 个，可以认为在该条件下使用 NLLLoss 的效果更好。

另外，对比了 Adam 和 SGD 两种优化算法，发现使用 Adam 优化算法的效果更好。

```
model = FNN(VOCAB_SIZE+1, EMBEDDING_SIZE, 128)
model.to(device)
loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
for epoch in range(EPOCH):
    total_loss = 0
    for i in tqdm.tqdm(range(0, len(corpus), BATCH_SIZE)):
        input = inputs[i:i+BATCH_SIZE]
        target = targets[i:i+BATCH_SIZE]
        model.zero_grad()
        log_probs = model(input)
        loss = loss_function(log_probs, target)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    print('Epoch:', epoch, 'Loss:', total_loss)
word_vectors = np.array(model.embedding.weight.data.cpu())
```

（二）RNN

RNN（Recurrent Neural Network）是一种具有反馈机制的神经网络模型。本次作业采使用`pytorch`构建`RNN`模型以获取词向量。

1. 模型构建

构建一个文本预测的`RNN`模型，将输入的文本序列转化为预测序列，其输入为一个序列的文本数据，输出为相应的预测序列。

该神经网络包含以下几个层：embeading 层、RNN 层和全连接层。其中，embedding 层负责将序列中的每个单词转化为相应的向量表示，RNN层通过在每个时间步处处理当前隐藏状态和当前输入，实现了对序列中上下文信息的建模，全连接层将最后一个时间步处的隐藏状态映射为输出。

```
class RNN(nn.Module):
    def __init__(self, vocab_size, embedding_size, hidden_size,
num_layers):
        super(RNN, self).__init__()
        self.vocab_size = vocab_size
        self.embedding_size = embedding_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.embedding = nn.Embedding(vocab_size, embedding_size)
        self.rnn = nn.RNN(embedding_size, hidden_size, num_layers,
batch_first=True)
        self.fc = nn.Linear(hidden_size, vocab_size)
    def forward(self, inputs):
        embedding = self.embedding(inputs)
        out, hidden = self.rnn(embedding)
        out = self.fc(out)
        return F.log_softmax(out, dim=-1)
```

2. 模型训练

(1) 训练数据准备

将单词映射到数值的部分与上述 `FNN` 模型中的相同，不再赘述。`RNN` 模型的输入是文本序列，输出是预测序列。具体来说，输入到该模型的数据是一个文本序列，每个元素代表该位置处的单词，而输出是经过模型预测得到的下一个单词的序列。

将每个句子最后一个词删除，作为输入序列；将第一个词删除，作为目标序列。

```
inputs = []
targets = []
for sentence in corpus:
    input_sentence = [word_to_ix.get(word, VOCAB_SIZE) for word in
sentence[:-1]]
    target_sentence = [word_to_ix.get(word, VOCAB_SIZE) for word in
sentence[1:]]
    inputs.append(torch.tensor(input_sentence, dtype=torch.long))
    targets.append(torch.tensor(target_sentence, dtype=torch.long))
```

(2) 训练

使用 `pad_sequence` 将一个batch中的序列填充到相同的长度（该 batch 中最长长序列的长度），以便于将它们传入模型进行训练。使用 `CUDA` 加速训练。

```
# train model
model = RNN(VOCAB_SIZE+1, EMBEDDING_SIZE, 128, 1)
model.to(device)
loss_function = nn.NLLLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
for epoch in range(EPOCH):
    total_loss = 0
    for i in tqdm.tqdm(range(0, len(corpus), BATCH_SIZE)):
        input = inputs[i:i+BATCH_SIZE]
        target = targets[i:i+BATCH_SIZE]
        model_input = pad_sequence(input, batch_first=True)
        model_target = pad_sequence(target, batch_first=True)
        model_input = model_input.to(device)
        model_target = model_target.to(device)
        model.zero_grad()
        log_probs = model(model_input)
        loss = loss_function(log_probs.view(-1, VOCAB_SIZE+1),
model_target.view(-1))
        loss.backward()
```

```
        optimizer.step()
        total_loss += loss.item()
    print('Epoch:', epoch, 'Loss:', total_loss)
word_vectors = np.array(model.embedding.weight.data.cpu())
```

（三）LSTM

LSTM（Long Short-Term Memory）是一种带有“记忆单元”的神经网络模型。本次作业采使用 `pytorch` 构建 `LSTM` 模型以获取词向量。

1. 模型构建

构建一个文本预测的 `LSTM` 模型，将输入的文本序列转化为预测序列，其输入为一个序列的文本数据，输出为相应的预测序列。该模型的结构包括一个嵌入层，一个LSTM层和一个全连接层。

```
class LSTM(nn.Module):
    def __init__(self, vocab_size, embedding_size, hidden_size,
num_layers):
        super(LSTM, self).__init__()
        self.vocab_size = vocab_size
        self.embedding_size = embedding_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.embedding = nn.Embedding(vocab_size, embedding_size)
        self.lstm = nn.LSTM(embedding_size, hidden_size, batch_first=True,
num_layers=num_layers)
        self.fc = nn.Linear(hidden_size, vocab_size)

    def forward(self, inputs):
        embedding = self.embedding(inputs)
        out, (hidden, cell) = self.lstm(embedding)
        out = self.fc(out)
        return F.log_softmax(out, dim=-1)
```

2. 模型训练

(1) 训练数据准备

同 `RNN` 模型。

(2) 训练

同 `RNN` 模型。

五、结果及分析

作业要求随机选择 20 个单词，计算与其词向量最相似的前 10 个单词，由于太占篇幅，因此将该结果放在附件中。

本次作业选取了多个语料和多种模型来获取词向量，因此将从多方面比较不同模型和不同语料获得的词向量之间的差异。

为了直观展示不同词之间词向量的差异，将获得的词向量进行降维处理，从而在二维图像上表示不同词的相似度和差异性。相关代码如下：

```
# tsne
tsne = TSNE(n_components=2, random_state=0)
word_vectors_tsne = tsne.fit_transform(word_vectors[:DISNUM])

# plot
plt.rcParams['font.sans-serif'] = ['SimHei'] # display Chinese
plt.rcParams['axes.unicode_minus'] = False # display '-'
fig, ax = plt.subplots(figsize=(10, 10), dpi=500)

cmap = plt.get_cmap('viridis')
for i, word in enumerate(word_to_ix):
    # only display DISNUM words
    if i == DISNUM:
        break
    color = cmap(i/DISNUM)
    ax.scatter(word_vectors_tsne[i, 0], word_vectors_tsne[i, 1],
color=color)
```

```
    ax.annotate(word, xy=(word_vectors_tsne[i, 0], word_vectors_tsne[i,
1]), color=color)
plt.savefig(name + '.png')
```

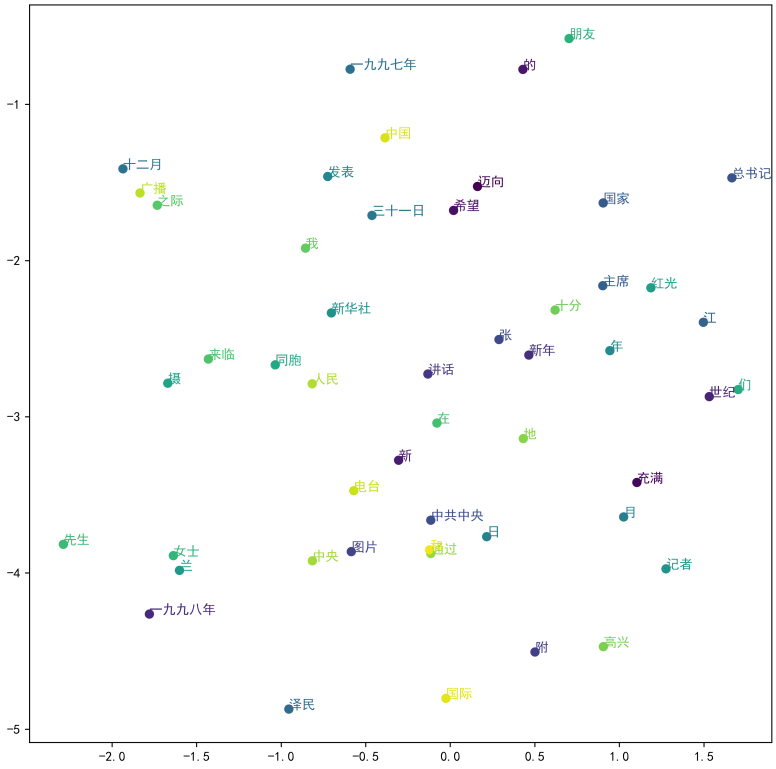
（一）中文语料

对于中文语料，三种模型对于词向量的获取效果都较差，对于同一词汇，不同模型获取的词向量具有较大的差异。同时，按照人的生活经验，多数具有相近词向量的词在语义上并没有很强的关联。例如，对于“中国”，三个模型获取的词向量最相近的词如下：

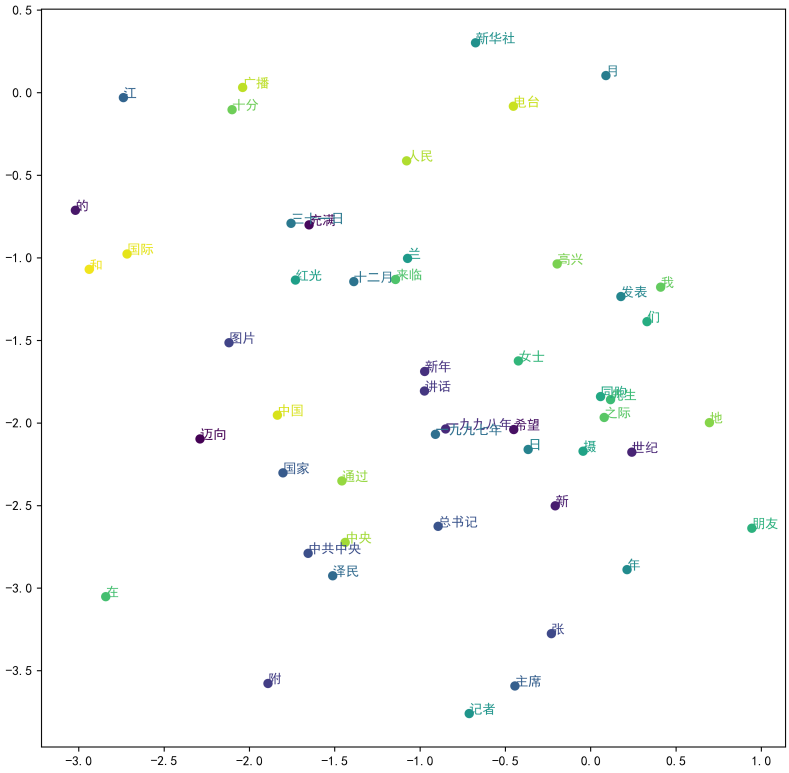
FNN	RNN	LSTM
一道	书记	届
指挥	月	组织
全党	宿堂	全国
地区	展望	指挥
坚决	握手	进行
成就	中央	选曲
各界	联合国	杜绝
激烈	我国	刘
差距	贾	出席
内容	调度	海外

表3

FNN 的词向量二维展示如下：



LSTM 的词向量二维展示如下：



“一九九七年”和“一九九八年”两个词，以及“中共中央”和“中央”的词向量使用 LSTM 获取的结果相较于 FNN 都更接近。LSTM 模型相对于 FNN 和 RNN 模型表现更好，获得的词向量具有更高的相似性计算准确度，聚类效果也更好。

（二）英文语料

对于英文语料，`RNN` 和 `LSTM` 模型的词向量获取效果显著优于 `FNN` 。`LSTM` 略优于 `RNN` 。对于同一英文语料(这里展示来自 ABC News, CNN 的新闻语料的结果)，对于 *years*，三个模型获取的词向量最相近的词如下：

FNN	RNN	LSTM
analyst	months	months
voting	weeks	weeks
justice	days	days
democracy	hours	hours
anyone	minutes	minutes
wanting	lie	success
minutes	occurred	shipped
disappear	oath	shortly
committing	crowd	significance
how	elsewhere	year

表4

`FNN` 获取的结果除了 *minutes* 外，其它词与 *years* 似乎关系不大。而`RNN` 和 `LSTM` 中前五个词都是与 *years* 非常接近的表示时间的词，它们获取的词向量具有更好的准确性。

另外，具有相同训练参数的 `FNN` 模型在结果上差异较大，

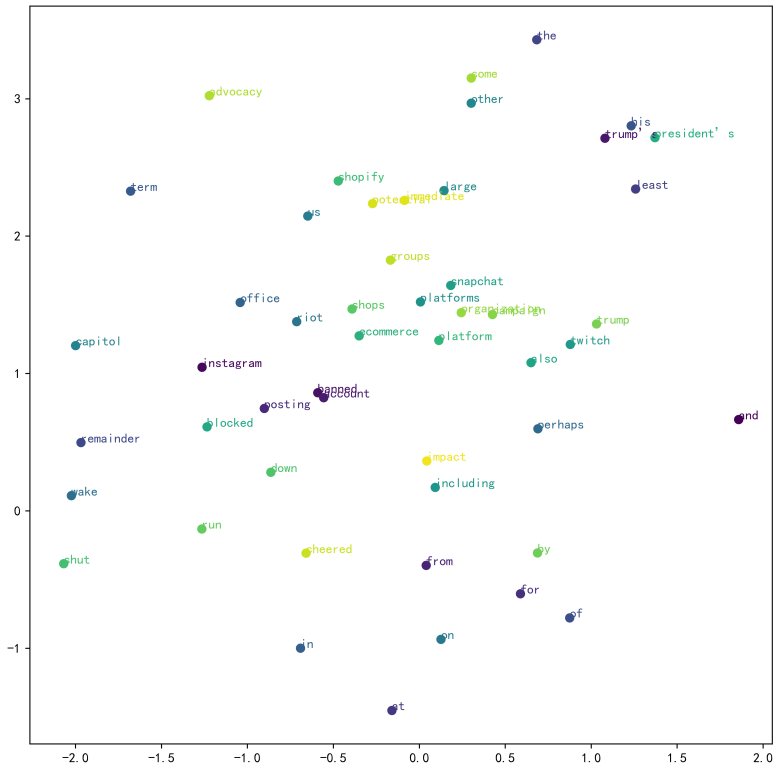
为了对比 `LSTM` 和`RNN` 的效果，选取了以下具有代表性的结果，获取与 *his* 词向量最相近的十个词：

RNN	LSTM
biden's	biden's
trump's	trump's
their	my
president's	mccarthy's
mcconnell's	raskin's
facebook's	pence's
its	whose
initial	president's
controversial	your
campaign's	twitter's

表5

RNN 中出现了 2 个关联不强的词，但 LSTM 中前 10 个词都是与 *his* 相似的形容词性的词语，语义上都表示所有格。

LSTM 的词向量二维展示如下(RNN 与之类似，可在附件中查看)：



可以看到图中下方集中了许多介词，如 *in*, *on*, *at*, *from*, *by*, *for*, *of*，这表明其聚类效果较好，词向量获取效果优异。

（三）其它

1. 英文语料之间的差异

对比发现，使用两种不同类型的英文语料获得的词向量效果差别不大，尽管小说语料的的规模比新闻语料的规模大很多，但在效果上的提高并不显著。两种语料获取的词向量有不同的倾向，例如，与 *man* 词向量最相近的十个词（使用 `LSTM` 训练）：

news	norvel
person	kid
rioter	woman
officer	boy
pair	girl
walters	guy
berman	gentleman
reception	therapist
resident	grandmother
insurrectionists	mother
pelosi	maria

表6

在新闻语料中，与之词向量相近的词中有 *rioter* 和 *officer* 这类带有一定刻板印象的词，以及常见于男性的名字 *walter* 和 *berman*；而在小说中，与 *man* 词向量接近的词都是具有普遍意义的描述性别或者带有其它特征的名词。

2. 语料规模的影响

但对于中文和英文的新闻语料，它们的规模相差约一个数量级。在中文新闻语料上使用三种模型训练得到的词向量效果都很差，相似性计算准确度以及聚类效果都不尽人意。在英文新闻语料上得到的词向量效果明显具有更高的准确性。

综合比较，可以得出结论，不同的语料类型和不同的神经网络模型对于词向量的获得具有一定的影响。在一定大小内，语料的规模对词向量的训练和获取有较为显著的影响；当规模达到一定大小后，语料的类型（内容）对词向量的影响更为显著。在不同的应用场景中，选择合适的语料类型和神经网络模型可以提高词向量计算的准确度和可靠性。

六、结论

本作业的结果表明，对于汉语语料，基于 `FNN`、`RNN` 和 `LSTM` 获取的词向量效果都很差，可能的原因是语料的大小不足。对于英语语料，基于 `RNN` 和 `LSTM` 获取的词向量显著地优于使用 `FNN` 获取的词向量，`LSTM` 略优于 `RNN` 。

本次作业仍然有一些可改进的不足之处。

第一，关于语料规模对训练效果的影响，应该使用一个规模更大的中文语料来进行对比，排除语言对结果的影响。

第二，在使用 `FNN` 结合 `Skip-gram` 方式训练时，在训练参数相同的情况下得到的结果变化较大。可能的原因有模型欠拟合或者过拟合，或者模型的结构设计不当等。还需再深入探究。

第三，模型效果的评估完全根据本人的生活经验判断。可以利用公认的效果较好的词向量模型 `word2vec` 在相同的数据集上训练，对比自己构建的模型获取的词向量与其的差异，从而更具体地衡量模型训练的效果。

通过本次作业，我学习了如何使用 `PyTorch` 来构建神经网络模型，如何调整神经网络的参数和优化训练效果，并且学习了如何使用文本分类、预测模型来获取副产物——词向量，同时也能更熟练地使用正则表达式、`CUDA` 等工具来提高效率。