任务二：基于深度学习的文本分类

1. 实验任务

使用 Pytorch 框架分别实现基于 CNN 和 RNN 的文本分类，比较不同的词嵌入方法对模型性能的影响，了解 Dropout 方法。

2. 数据集

使用和任务一相同的数据集，这里将 Kaggle 给定的有标签的训练集进一步划分为训练集和验证集，训练集用于训练模型参数，验证集用于调整模型超参数。这里训练集和验证集的划分比例为 3:1。

3. 模型实现

3.1 整体流程

完成该任务主要分为以下几步：

1) 数据预处理：读入数据集，而后使用 torchtext 文本处理工具，生成训练集、验证集和测试集迭代器
2) 词嵌入：使用 torchtext 读取预训练文件
3) 文本分类：分别完成基于 CNN 和 RNN 的文本情感分类

3.2 词嵌入

任务一中使用 N-gram 提取文本特征，这种方式是以句子为单位，每个句子被表示为词表长度的向量。本任务在提取文本特征时从词的角度考虑，以词语为单位，将每个词语表示为相应的特征向量，即词嵌入(Word Embedding)。词嵌入主要有 one-hot representation 和 distributed representation 这两种表示方法。

本实验使用 torchtext 完成词嵌入及数据预处理。Torchtext 是一个优秀的 NLP 数据预处理工具，主要包括 Field，Dataset 和 Iterator 这三个组件。Field 主要包含数据预处理的配置信息，如指定分词方法、起始字符、结束字符、补全字符，大小写转化等。Dataset 用于家在数据并划分为训练集、验证集和测试集。Iterator 主要是数据输出的模型迭代器。

使用 torchtext 构建测试集的迭代器时，由于测试集本身是没有标签的，在打包生成 examples 时不能将 LABEL 一栏写为 None，而是需要给定某个初始化的标签值，否则 Iterator 无法迭代。此外，为保证测试集数据顺序不被打乱，还需要在生成测试集迭代器时如下设置：test_iter = Iterator(test,batch_size=batchsize, sort=False, shuffle=False, sort_within_batch=False, repeat=False,train=False)

3.2.1 Random embedding

Pytorch 中随机初始化词向量，使用 torch.nn.Embedding 命令完成，之后词向量会随模型训练一起更新。若要使词向量在训练中保持固定，则需要在模型中设置如下参数：”self.embed.weight.require_grad=False”。

3.2.2 Pre-trained embedding —— Glove

词语的分布式表示则能够将语义融入到词向量中。它的基本思想是词的语义由其上下文决定，因此可以通过模型刻画某个词与其上下文之间的关系，由此生成它的词向量。基于这

种思想有两种常见的构建模型，分别是基于神经网络的 word2vec 模型和基于矩阵的 Glove 模型。

Glove 是一个基于全局词频统计的词表征工具，它结合了全局矩阵分解和局部上下文窗口这两种方法。它的基本思想是首先基于语料库构建词的共现矩阵，然后基于共现矩阵和 GloVe 模型学习词向量。本实验使用预训练的 50 维 Glove 词向量。

3.3 模型原理

3.3.1 text-CNN

Yoon Kim2014 年的一篇文章开辟了使用 CNN 完成文本分类任务的先河，该文章中的模型也就是 text-CNN 模型，模型的基本结构如下图：

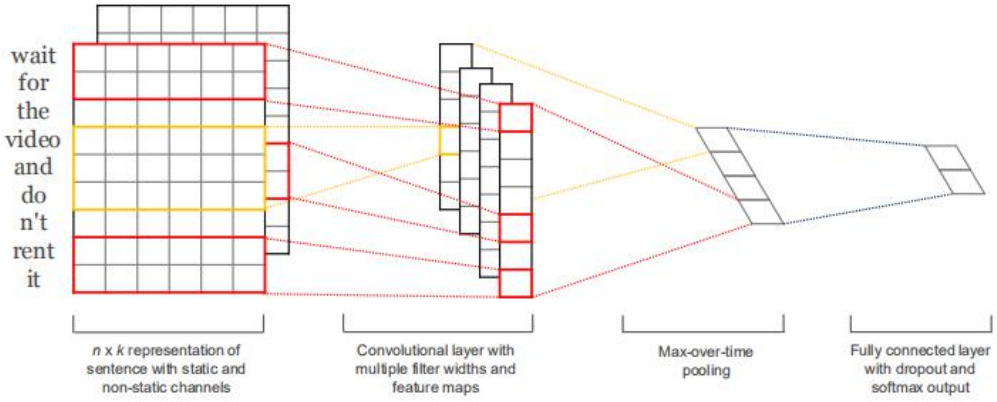

图 1 text-CNN 模型结构

第一层将单词嵌入到低位矢量中，下一层使用多个过滤器对嵌入的单词向量执行卷积，如一次滑动 3,4,或 5 个单词。接下来将卷积层的结果最大池化为一个长特征向量，添加 dropout 正则。

该模型与传统的卷积神经网络相比，其卷积核的宽度即为词向量的维度，步长常取 1，步长增大会使其更接近 RNN 模型。使用 text-CNN 每一条文本的长度必须是确定的，因此首先需要观察数据集短语的长度分布，然后挑选合适的 fixed-length。已有的训练集数据中短语长度的描述性统计结果如下：

表 1 训练集短语长度统计

Count	156060.000000
Mean	**7.203377**
Std	7.024593
Min	1.000000
25%	2.000000
50%	5.000000
75%	**10.000000**
max	52.000000

这里将 fix-length 设置为 15，因为训练集的文本数据中，有近 90%的短语长度都小于或等于 15。本实验中使用的模型超参数如下表：

表 2 text-CNN 模型超参数

Hyper-parameter	Value
Word Embedding	Random/Glove
filter	(3,4,5)

Num(Filter) per size	100
Active Function	Relu
Max-pooling	1-max-pooling
Epoch	50
Learning Rate	0.01
Dropout rate	0.5

TextCNN 的网络结构简单，引入预训练的词向量依旧可以有很好的模型效果。

3.3.2 RNN

循环神经网络在自然语言处理领域有着广泛的应用，可以很好地处理时序数据。同样，在进行文本分类时，也会指定固定的序列长度。对于每个输入的文本序列，在 RNN 的每一个时间步上输入文本中一个单词的向量表示，然后计算当前时间步长上的隐藏状态，用于当前时间步的输出并传递给下一个时间步。在之后的每一个时间步上重复这一操作，模型基本结构如下：

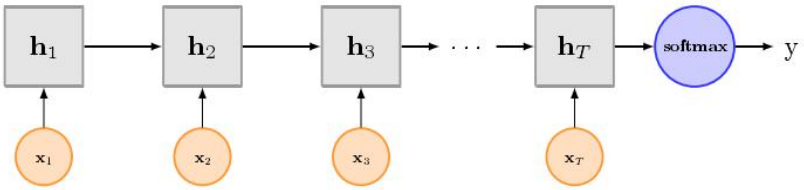

图 2 textRNN 结构

textRNN 与 textCNN 相比，更擅长捕获更长的序列信息。本实验中 Text-RNN 使用的超参数如下：

表 3 textRNN 模型超参数

Hyper-parameter	Value
Layer Num	2
Hidden Size	32
Epoch	50
Learn Rate	0.001
Dropout Rate	0.5

4. 模型评估

4.1 不同词嵌入方法对模型性能的影响

这里主要比较使用随机词向量和预训练的 Glove 词向量的模型效果，两种模型 batchsize 均设置为 128。实验还比较了不同词向量设置下的模型效果，实验结果如下：

表 4 不同词嵌入方法对模型性能的影响

	Acc(Text-CNN)	Acc(Text-RNN)
Random Embedding	0.6119	0.5902
Glove	**0.6125**	**0.6209**

预训练的词向量由于包含更多的语义信息，其结果优于随机初始化的词向量。进一步比较 TextCNN 和 TextRNN 的训练表现，使用本实验的模型参数，CNN 模型与 RNN 模型相比，训练中收敛速度更快，但所需时间更长。跟据以上的实验数据，使用 RNN 模型和预训练的 Glove 词向量完成该文本分类任务，效果最好。

4.2 测试集上的模型效果

选取在上述验证集中表现最好的词向量及训练方法，在 Kaggle 测试集上进行测试，结果如下：

Model	Accuracy
Text-CNN	0.5775
RNN	0.5910

该结果也明显好于实验一。

5. 实验小结

与实验一相比，本任务有两点改进：使用词向量提取文本特征，使用深度学习方法完成文本分类任务。在词嵌入方面，无论是使用随模型训练更新的随机词向量还是预训练的词向量，其本身蕴含的信息多于 Bow 或 N-gram 模型。在模型方面，深度学习模型的性能也优于 softmax-regression 这种简单的机器学习模型。