華中科技大學

Python 大数据与人工智能实践课程

设计报告

姓　　名：　　杨明欣、张森磊

学　　院：　　计算机科学与技术学院

专　　业：　　计算机科学与技术

班　　级：　　CS2106、CS2104

学　　号：　　U202115514、U202115424

指导教师：　　邹逸雄、周正勇

分数	
教师签名	

2023 年 1 月 15 日

目　　录

1 实验目的

1.1 研究背景与意义

1.1.1 研究背景

电影，一种融合了视觉与听觉的当代艺术，实质上是一个围绕戏话与表达艺术的联合场景。由于网络行业的快速发展，大量的电影评论网站及平台应运而生。在这些影评网站和 APP 上，都有很多关于影片的评价，而这些海量的影评数据则蕴含大量潜在的价值: 一方面，用户可以根据影片的好评率来决定是否观影; 另一方面，影院或导演、投资人等可以利用情感分析技术来挖掘影院服务或影片等各个维度的好评情况，多个维度进行情感分析以帮助影院改进经营状况、给到导演观众的的真实评价及建议以及利于投资人掌握投资风向。

情感分析也被称作情绪倾向分析或者观点发掘，它是从使用者的观点中抽取出一些有用的信息，即从具有情感性的文本入手，发掘其情绪取向，并将其归类。情感分析在自然语言处理中占有举足轻重的地位。在国外，人们对情感分析的研究多依附于社会媒介推特，通过对推特上的政治情绪和对候选人的评价进行评估可以预测大选的结果，利用该平台上用户沟通的情绪特征，可以侦测到网络上的暴力及欺凌行为，及时制止，为网络用户创造一个更为友善的网络生态。

1.1.2 研究意义

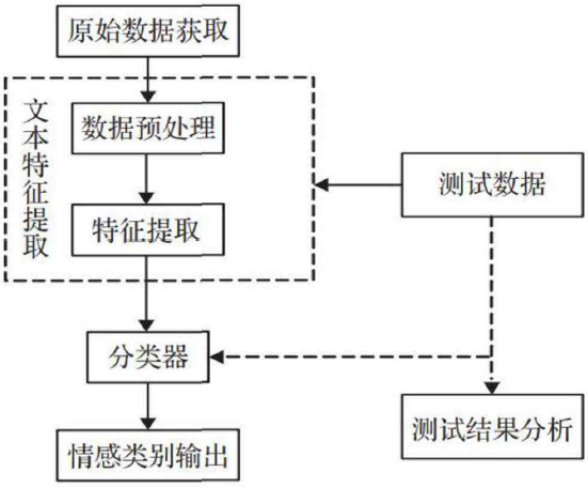

图 1-1 文本情感分析过程

本文使用第三方平台烂番茄电影评论数据集作为研究对象。文本情感分析流程如图所示，主要包括数据获取、预处理，分类器选择、情感类别输出等步骤。电影评论挖掘与其他文字评论资料的区别在于，它既包括个体对电影情节的观点和看法，也包括对导演、编剧、主角等电影相关人物的主观评价。评价要素的多元化、情感信息的丰富程度、语言表达方式的灵活多变，使得对电影评价的情感分类剖析难度较大，但深入研究具有较大的理论与实践价值。

1.2 国内外研究现状

1.2.1 情感分析方法研究现状

根据使用方法的不同，情感方法分为下列三种。

（1）基于情感词典的情感分析

该方法是以各种情感词典中所提供的倩绪词的倩绪极性为基础，对文本情感倾向进行分类。该方法基本步骤是; 先将已处理好的文本 (包括去重、删除无关文字等) 输入，再进行分词，在此基础上，将情感词典中各种类别、属性以及表达程度的词汇纳入到该模型中进行训练，并按照情感判定的规则将其分类标签输出。一般情况下，现存的词典基本上是人为构建的。在构建情感词典的过程中，英文情感词典发展得很快，并且已经相当成熟，目前使用最多的 GeneralInquirer,SentiWordNet, Opinion Lexicon 和 MP QA 等。

情感词典能够精确地表达非结构文本的特点，便于对其进行分析和理解。然而，该方法也有其不足之处: 目前，此方法依赖于建立的情感词典质量，但随着互联网的飞速发展以及资讯的快速更新，出现了大量的新词语。很多新词语，如歇后语、习语或网络专用语，都无法很好地识别。此外，该方法在进行情感分析时往往忽略了语境中的意义联系。所以，关于情感辞典的研究还有待于进一步深入的研究。

（2）基于机器学习的情感分析

为提高情感分类的精确度，有学者对传统的机器学习技术进行了深入的探讨，并获得了较好的结果。机器学习是一种通过对所获得的数据进行训练和预测的学习方式。该方法是一种利用众多有标记和无标记的语料库，利用统计学的机器学习算法进行特征抽取，从而得出情感分类的最后结果。在传统的机器学习算法中，通过优化情感特征提取方法和情感分类器的不同组合方式可以实现对语

篇情感的归类。

然而，随着研究领域深度的不断扩大，在文本情感分析中，传统机器学习算法的应用陷入了一个瓶颈。传统的机器学习方法侧重于情感特征的抽取和分类组合的选取，不同的组合方式对于情感分析的效果则存在着不同的影响。由于传统的文本识别技术难以将上下文、前后词语在语境中的差异进行有效的区分，因而在情感分析中会出现忽视上下文语义的问题。基于此，许多学者将深度学习应用到情感分析中，并在此基础上进行了大量的探索。

（3）基于深度学习的情感分析

国内外的学者对深度学习进行了大量的探索，并且在很多领域都有突出的成果。Hinton 等人 (Hinton et al. 2006) 最先利用深度神经网络来分析数据中的重要信息即深度学习的方法，从而有效地体现出其特征，以改善其性能表现。深度学习方法是利用连续、低维向量来表达文档和词语，从而能够很好地处理数据的稀疏问题; 另外，深度学习以端对端训练方式，能够实现文本属性的自动提取，降低了文本特征创建的难度。

对于基于深度学习的情感分析，目前主流研究的方向在于两个方面，一是对底层词向量的研究。另一个主流的研究方向是对神经网络模型的研究。

深度学习方法可以从众多语料库中自动提取隐藏特征，然后学习更深层次的文本语义知识。因此，与常规的机器学习算法相比，对情感分析的分类效率更高。

1.2.2 电影评论的情感分析研究现状

电影评论这类短文本因其句子包含的情感词较少，而非情绪性词语如程度副词、否定词等对语句的情绪取向有很大的作用，因此，单靠情感词的权重以判定情感分类往往精度不高，必须将其与相关的机器学习、深度学习方法相结合进行研究。近年来，针对电影领域文本的情感分析大多也是以机器学习和深度学习的算法展开研究的，且使用的数据集由公开的数据集逐渐转向本土语言数据集。

作为一个独特的领域，影评评价要素的多元化、情感信息的丰富程度、语言表达方式的灵活多变，对电影评价的情绪取向的剖析带来很大的难度，这就导致分析影评的情感倾向极具挑战性，但同时也具有极大的理论与应用研究价值。本文针对影评数据，一方面希望训练出更适合电影领域内的词向量，另一方面希望基于深度学习算法构造出更高性能的情感分类模型。

2 实验内容

2.1 常见情感分析算法简介

2.1.1 支持向量机（SVM）

支持向量机 (Support Vector Maehin，SVM) 在机器学习算法领域有着举足轻重的地位。该算法是一种应用于类别识别二分类问题的机器学习方法。支持向量机作为机器学习中的经典算法之一几，主要用于处理数据分类问题。在采样的范围内，我们需要找到一个可以很好地将所有的样品空间资料都分离出来的超平面。支持向量机的分类原理示意图如图2-1。

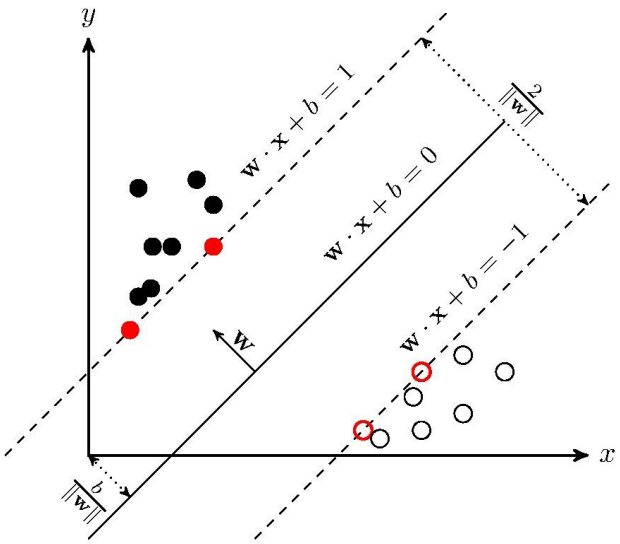

图 2-1 支持向量机原理示意图

从支持向量机的基本原理上可以看出，要寻找一个超平面与每个样品之间的间距最大。从图中可以看出可知，超平面计算公式为:

$$f(x) = w * x + b \tag{2.1}$$

x 表示训练的数据，w，b 表示参数。

设 γ 为距离，可得如下距离公式 3.2,3.3。

$$max\gamma \tag{2.2}$$

$$\text{s.t.} \left(y_i \frac{w}{\|w\|} \cdot x_i + \frac{b}{\|w\|} \right) \geq \gamma, i = 1, 2, \ldots, n \tag{2.3}$$

y_i 表示数据的标签分类，取值为 1 或-1。

对式 3.3 两边同时除以 γ 并化简，计算公式如 3.4，3.5 所示。

$$\min \frac{1}{2}\|w\|^2 \tag{2.4}$$

$$\text{s.t. } y_i\left(w \cdot x_i\right) - 1 \geq 0, i = 1, 2, \ldots, n \tag{2.5}$$

这是一个带有不等式限制的凸二次规划问题，换成对偶问题。计算公式如下。

$$L(w, b, a) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i y_i \left(w \cdot x_i + b\right) + \sum_{i=1}^{n} \alpha_i \tag{2.6}$$

根据上述公式将其归结为一类等价的最优解，对公式 3.2，3.3 进行化简，如下列所示：

$$\min \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left(x_i \cdot x_j\right) - \sum_{i=1}^{n} \alpha_i \tag{2.7}$$

$$\text{s.t. } \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{2.8}$$

$$\alpha_i \geq 0, i = 1, 2, \ldots, n \tag{2.9}$$

2.1.2 卷积神经网络（CNN）

卷积神经网络 (Convolution Neural Network, CNN) 是一种经典的深度学习的方法，它将卷积式操作引入到深层次的网络中，是目前最典型的算法之一。

在 2014, Kim 基于卷积神经网络的一维卷积核，从文字中抽取出部分 N-Gram 的特征，并在此基础上提出了一种基于卷积神经网络的卷积体算法用于文本分类，其模型结构如下图2-2。

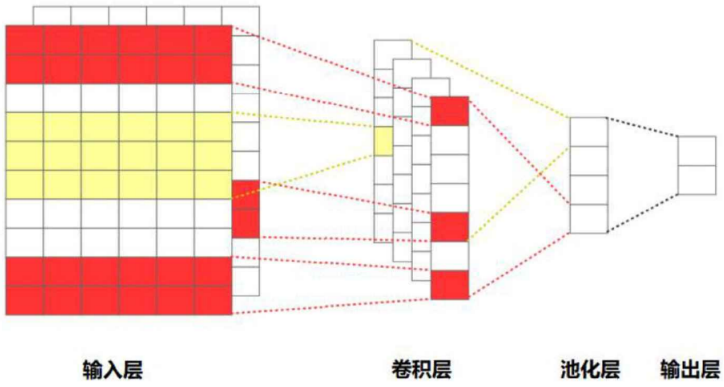

图 2-2　CNN 模型结构图

输入层:CNN 输入层采用 one-hot 编码 (独热编码) 或词向量编码的方式，将文本中的每一个词、符号、表情等语言符号映射成对应的向量形式，每一个文本输入形成一个二维文本矩阵 S。矩阵 S 的每一行代表基本符号的词向量，矩阵 S 的行数代表整个文本的长度。

卷积层: 卷积部分的特点是利用卷积核的方法进行局部特征的抽取。卷积技术的特征是选择符合卷积核大小的数据，再与卷积核中对应的位元进行乘积，然后加入一个偏置、得到一个数据。在此基础上，按照一定的步骤，把所有数据都按一定的顺序滚动。当全部数据都被完整地覆盖，就可以得到该数据的全部特征。同时，其相同层具有一致的权重，这使各层间的参大为减小。

池化层: 在采用多个卷积核时，可能会产生多个特征，而池化层是通过对卷积特征进行池化处理，从而得到更有意义的特征，从而降低了过度拟合。

输出层: 输出层是将已完成池化的矢量集完全连通拼接后，再将其输入分类模型中得出分类结果。

2.1.3　循环神经网络 (RNN)

与传统的神经网络相比，RNN 采用了“记忆”这一概念，把上一个瞬间的输出和现在的输入结合起来，这就非常适用于文本、语音、视频等连续的数据的加工。这些数据的取样之间有连续的关系 (通常是时序关系)，每一个样本与其先前的样本有联系。RNN 内部结构如下图2-3所示。

RNN 的网络结构图主要分为三层：输入层，隐含层和输出层。

（1）输入层：输入层是将序列 x_t 在神经网络的 t 时刻进行输入，输入层的输入集合为 $\{x_0, x_1, \cdots, x_{n-1}, x_n\}$, 在文本序列中，$x_t$ 可以表示为文本中第 t 个单词的

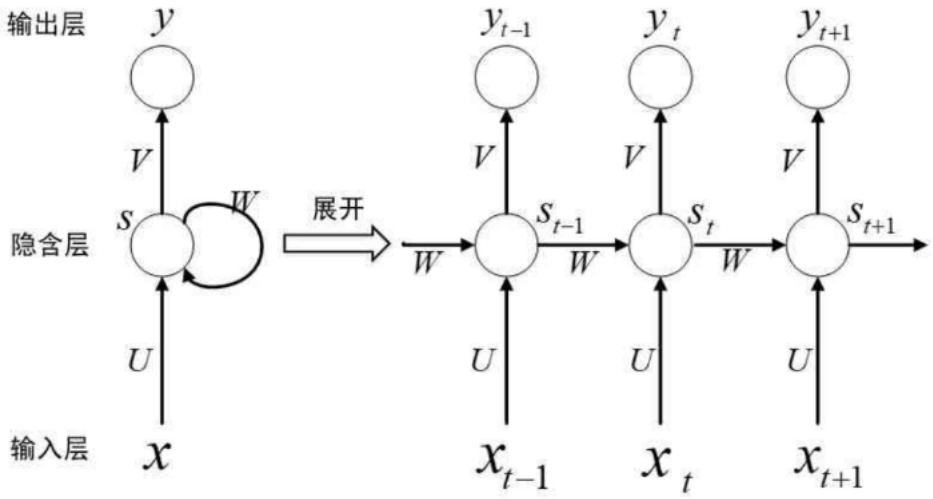

图 2-3　RNN 模型结构图

词向量。

(2) 隐含层: 隐含层的计算不仅依赖于当前输入层 x_t , 还依赖于上一个计算节点的状态值 s_{t-1} , 隐藏层节点实际上对上个节点的信息进行了保存, 因此隐藏层节点的计算主要分为当前节点和上一节点, 计算公式为:

$$s_t = f(Ux_t + Ws_{t-1}) \tag{2.10}$$

其中, U 为输入层到隐藏层的连接权重, W 是前一时刻到当前时刻隐藏层的连接权重, s_{t-1} 表示上一时刻的节点状态值。

(3) 输出层：在时间步骤 t 时刻的输出结果为 y_t, 输出层与隐含层连接的权重为 V，则输出层的计算公式为:

$$y_i = softmax(Vs_t) \tag{2.11}$$

其中 softmax 函数将变量进行归一化处理。

2.1.4　长短时记忆神经网络（LSTM）

长短时记忆神经网络 (Long Short Term Memory, LSTM) 是基于 RNN 并对其优化的一种模型，采用特殊“门控单元”克服了传统 RNN 不能实现长时间存储的缺陷。LSTM 在机器翻译、语法分析、语义表达等方面有着广阔的应用前景。下图**??**是 LSTM 模型结构图，概括了一个细胞单元 (Cell) 在 LSTM 网络中的具体操作。

第一步是将哪些信息从细胞单元中删除，这个过程由一个“遗忘门”来实现。

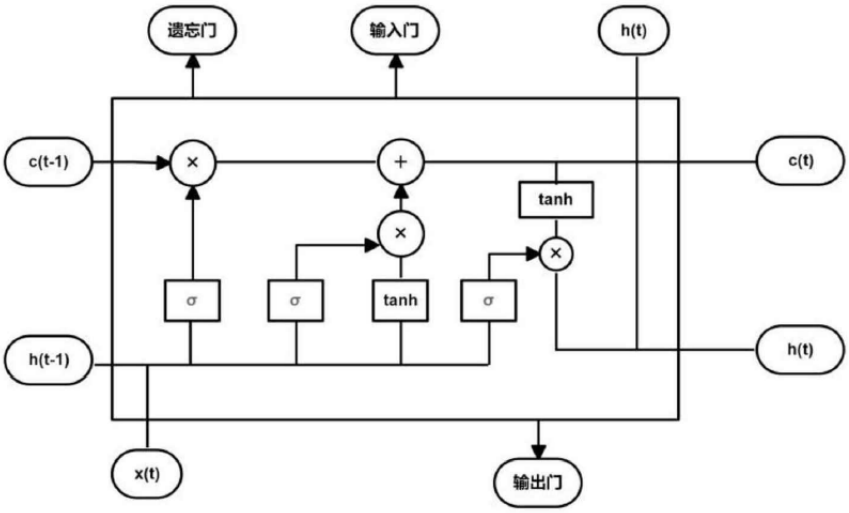

图 2-4　LSTM 模型结构图

遗忘门: 在 sigmoid 网络层次上存在一个“遗忘门”，通过其来进行信息的加工。前后两道遗忘门的输入与输出柑互关联。该方法可以使各种状态的流量与到达下一时间的信息量成正比，而遗忘门则能实现对信息流的流量的控制。其运算方法是:

$$f_t = \sigma\left(W_f \cdot \text{concat}\left(h_{t-1}, x_t\right) + b_f\right) \tag{2.12}$$

其中，W_f 为权重矩阵，b_f 为偏置项，σ 为 sigmoid 函数。

输入门:LSTM 中使用一个“输入门”来确定最近的数据序列流向当前 Cell 来。首先夕在 sigmoid 网络层次上，利用网络的梯度值来进行参数的更新，并根据权值的改变来进行权值的调整，最后利用 tanh 层来进行计算并替换。当更新发生时，将二者相乘:

$$i_t = \sigma\left(W_i \cdot \text{ concat }\left(h_{t-1}, x_t\right) + b_i\right) \tag{2.13}$$

$$C_t^{-} = \tanh\left(W_c \cdot \text{concat}\left(h_{t-1}, x_t\right) + b_c\right) \tag{2.14}$$

输出门: 这部分分两个阶段来完成。首先，在预先确定的 sigmoid 层上，通过对输出的数据进行计算，得到各分量的权重，接着在 tanh 级中，通过对这些参数的更新，将二者的结果相乘，并以下列方式进行运算:

$$o_t = \sigma\left(W_o \cdot \text{concat}\left(h_{t-1}, x_t\right) + b_o\right) \tag{2.15}$$

$$h_t = \tanh\left(C_t\right) * o_t \tag{2.16}$$

综上，任何一个语句矢量都可以由上面的运算获得一个矢量，即 LSTM 所获得的本征矢量。

2.1.5 双向长短时记忆神经网络 (BI-LSTM)

传统 RNN 或 LSTM 仅能捕捉到一段历史数据，因为其本身的语法特点，在进行顺序标记时，往往要使用将来的数据，也就是在与现有单词相匹配的情况下，使用目前单词 (右边) 的单词。因此，双向循环神经网络 (Bi-RNN) 被提出以解决此类问题 (Paliwal and Schuster 1997)，其基本思想是在一个隐藏层中使用两个 RNN 对自左向右 (前向) 和自右向左 (后向) 项中的序列进行建模，然后对结果进行连接。其中，自左向右 RNN 计算公式如下:

$$h_t^l = H\left(W^l \cdot \text{concat}\left(h_{t-1}^l, x_t\right) + b^l\right) \tag{2.17}$$

自右向左的 RNN 计算如式:

$$h_t^r = H\left(W^r \cdot \text{concat}\left(h_{t-1}^r, x_t\right) + b^r\right) \tag{2.18}$$

t 时刻的 $h_t = concat(h_t^l, h_t^r))$(即将 h_t^l 和 h_t^r 两个向量首尾拼接起来) 双向的 LSTM <Bi-directional Long Short-Term Memory, Bi-LSTM) 是将 Bi-RNN 中的重复单元替换为 LSTM 结构，这样可以将 Bi-RNN 的优点和 LSTM 结合起来。

2.1.6 自注意力机制 (SelfAttention)

自注意力机制（Self-Attention）, 其基本思路就是将输入 source 和 query 当做同一数据源, 被广泛的用在机器翻译模型中。自注意力机制能够捕获一个句子中的单词之间的语义特征或者是句法特征。自注意力机制关注的是文本序列中每一部分对其他部分的对应关系, 远距离依赖特征之间的距离被极大的缩短, 能够充分利用这些特征。自注意力机制模型结构图如图2-5

自注意力的计算公式为:

$$f\left(x_i, q\right) = W^T \sigma\left(W^{(1)} x_i + W^{(2)} q\right) \tag{2.19}$$

自注意力机制关注的是文本序列中内部的相互关系, 这样使得模型能够突出句子中的主要内容, 比如形容词和副词等。这种对于长距离依赖关系对于自然语言处理任务来说具有重要的作用, 不仅能够提高模型的准确率, 而且能够获取到

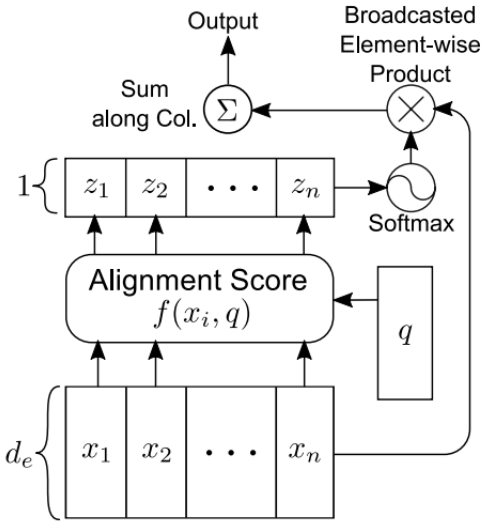

图 2-5　自注意力机制结构图

文本中更加重要的信息。

2.2　实验介绍

本项目使用深度学习方法对烂番茄电影评论情感分析，通过多种方法模型进行实验，通过多种技巧提高模型的效果，实现对于数据处理和深度学习的相关内容进行深入学习和提高。

3 实验设计

本章构建了使用深度学习方法的烂番茄电影评论情感分析（Sentiment Analysis on Movie Reviews）总体流程，并详细描述了本文的数据获取和预处理过程，同时还介绍了本文的实验环境以及评价标准设定。

3.1 整体流程

如图所示为使用深度学习方法对于烂番茄电影评论情感分析实验进行的整体设计。我们所处理的数据集为 kaggle 比赛中的数据集，需要对于数据集文本进行向量化，然后将 train.tsv 进行数据集划分为训练集和验证集，在使用训练集训练神经网络模型，使用验证集查看效果并及时纠正模型参数，最终对于测试集进行预测，通过 kaggle 网站进行结果的评估。

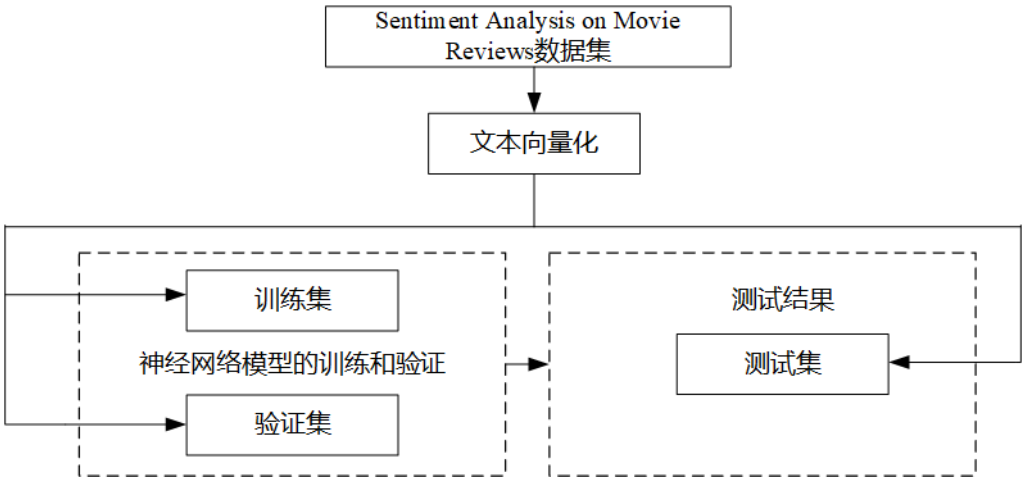

图 3-1 整体流程

3.2 文本数据集

本项目采用 Kaggle 提供的 Sentiment Analysis on Movie Reviews 数据集，同时对于数据集进行了一定程度的分析，以便于更好地了解和利用数据的特征，构建效果更好的模型。

数据集的基本统计特征包括数据量、平均值、标准差、方差、数据分布等一系列特征。其中测试集共 156060 条评论文本（按照最细粒度计算），电影评论

来源于烂番茄网站。根据情感分类结果，电影评论中好评（positive 和 somewhat positive）、中评（neutral）、差评（negative 和 somewhat negative）分别各有 42133 条、79582 条、34345 条评论文本，语料中 90% 以上评论长度在 50 个英文单词以内，平均长度为 8 个英文单词。训练集的具体分析结果如图所示。

	PhraseId	SentenceId	Sentiment
count	156060.000000	156060.000000	156060.000000
mean	78030.500000	4079.732744	2.063578
std	45050.785842	2502.764394	0.893832
min	1.000000	1.000000	0.000000
25%	39015.750000	1861.750000	2.000000
50%	78030.500000	4017.000000	2.000000
75%	117045.250000	6244.000000	3.000000
max	156060.000000	8544.000000	4.000000

图 3-2　训练集特征

同时，由于情感分析任务的特殊性，对于每一个句子进行的情感分析时，句子中的某些词语对于情感分类的结果可能会有更大的影响，因此分析文本中不同情感对应的词频对于模型结构的优化和解释会有很重要的意义，电影评论中积极情感对应词云图如图所示，中立情感对应词云图如图所示，消极情感对应词云图如图所示。

图 3-3　积极情感对应词云图

从积极情感词云图中可以看到，除了一些电影评论中的常规词以及语句结构词（film,movie,performance）等，偏向积极情感的词语的确占多数（brilliant,funny,love）等。

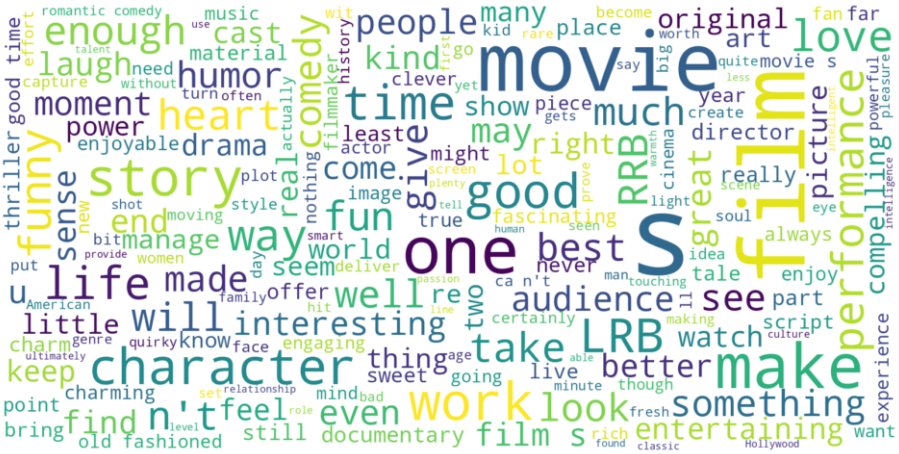

图 3-4　中立情感对应词云图

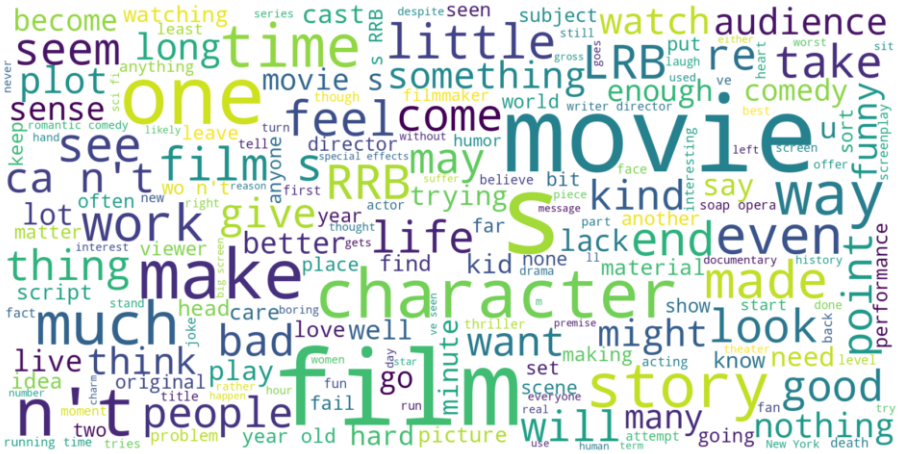

图 3-5　消极情感对应词云图

同时，消极情感对应的词语特征也比较明显，为模型的构建提供了很多的依据。

最终我们将 train.tsv 按照 9：1 进行划分为训练集和验证集，具体划分后的数据集特征如下。可以看到划分后的数据集中各个情感倾向的比例接近 9：1，说明数据集划分较为合理。

表 3-1 数据集划分

划分的数据集	总数	情感倾向	数目
训练集	140454	积极情感	8303
		中立情感	29709
		消极情感	102442
验证集	15606	积极情感	903
		中立情感	3218
		消极情感	11485
测试集	59663	积极情感	-
		中立情感	-
		消极情感	-

3.3 文本预处理

由于 kaggle 比赛中提供的 Sentiment Analysis on Movie Reviews 数据集较为干净，无需进行进一步的清洗和结构化。

3.4 文本向量化

文本向量化（又称“词向量模型”、“向量空间模型”）即将文本表示成计算机可识别的实数向量，根据粒度大小不同可将文本特征表示分为字、词、句子或篇章几个层次。文本向量化的方法主要分为离散表示和分布式表示。

文本向量化的**离散表示方法**包括 One-Hot Representation 和词袋模型（包括 TF-IDF、CountVectorizer 等），优点是方法简单，当语料充足时，处理简单的问题如文本分类，其效果比较好。词袋模型的缺点是数据稀疏、维度大，且不能很好地展示词与词之间的相似关系。对于本项目的深度学习框架可能并不适合。

文本向量化的**分布式表示**包括（1）基于矩阵的分布表示（2）基于聚类的分布表示（3）基于神经网络的分布表示等。模型包括 Word2vec 和 GloVec 等，本文主要采用 glove 词向量进行文本向量化，通过加载斯坦福大学网站 GloVe: Global Vectors for Word Representation 提供的 pre-trained word vectors 进行文本的向量化。

3.5 开发环境

为了实验效果和实验速度，本项目主要在 autodl 云服务器上进行运行，使用 Ubuntu20.04 操作系统，CUDA 为 11.3，具体硬件参数如图所示。

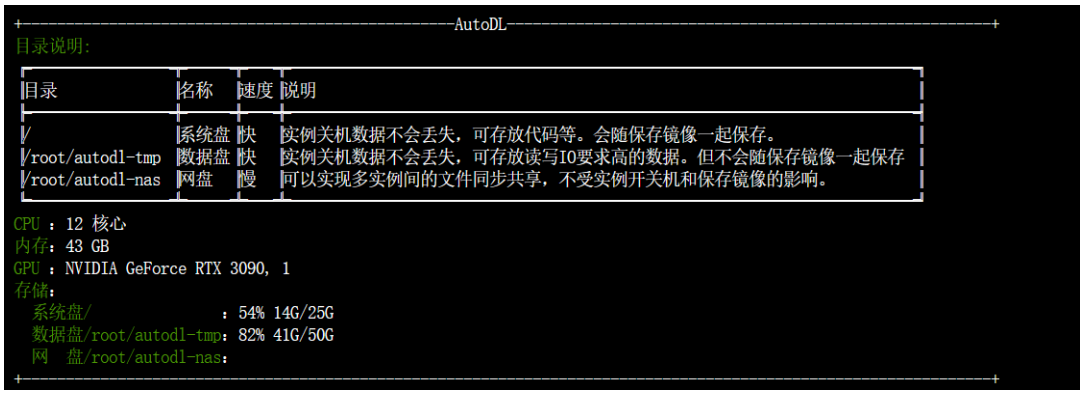

图 3-6 项目开发环境

本项目主要基于 Pytorch 框架进行模型的构建和训练。Pytorch 是 torch 的 python 版本，是由 Facebook 开源的神经网络框架，专门针对 GPU 加速的深度神经网络（DNN）编程。Torch 是一个经典的对多维矩阵数据进行操作的张量（tensor）库，在机器学习和其他数学密集型应用有广泛应用。与 Tensorflow 的静态计算图不同，pytorch 的计算图是动态的，可以根据计算需要实时改变计算图。

本项目还使用了 numpy，pandas，sys，transformers 等一系列的库，具体安装需求请见附录。

3.6 评价标准

对于评论文本情感分析任务，使用机器学习中分类任务的常用评价标准 Acc（准确率）来评估模型效果。Acc 的定义如式所示。

$$\text{Acc} = \frac{|\{x : x \in X \wedge f(x) = y(x)\}|}{|X|} \tag{3.1}$$

式中 $f(x)$ 为模型预测类别，$y(x)$ 为样本 x 实际类别，x 为数据集中数据量，该指标的含义是模型分类正确的样本数占样本总数的比例，比例越大，说明模型分类效果越好，是对模型整体性能的评价。

4 模型框架设计

本章主要构建常用的模型框架，包括 CNN（卷积神经网络）、RNN（循环神经网络）等，是用于文本情感分析的最常用深度学习模型。CNN 通过设置不同的卷积核能够提取到文本的局部 N-Gram 特征，RNN 的结构决定它适合处理文本这种序列数据，实际中常用的是 RNN 的两种变体 LSTM（长短时记忆网络）和 GRU（门控循环网络）。

4.1 基于 CNN 的文本分类模型 TextCNN

TextCNN 将卷积神经网络 CNN 应用到文本分类任务，利用多个不同 size 的 kernel 来提取句子中的关键信息，从而能够更好地捕捉局部相关性。与传统图像的 CNN 网络相比，textCNN 在网络结构上没有任何变化，包含一层卷积层，一层 max-pooling 层, 最后将输出外接 softmax 来进行情感分类。

4.1.1 模型基础

卷积神经网络（Convolutional Neural Network, CNN）是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，对于大型图像处理有出色表现。与普通神经网络非常相似，具有可学习的权重和偏置常量 (biases) 的神经元组成。每个神经元都接收一些输入，并做一些点积计算，输出是每个分类的分数，普通神经网络里的一些计算技巧到这里依旧适用。

随后，CNN 也被用在自然语言处理领域的文本分类中，也取得了非常好的效果。

4.1.2 网络结构

本项目使用 TextCNN 的网络结构参考 Yoon Kim 于 2014 年在 “Convolutional Neural Networks for Sentence Classification” 论文提出的算法框架。

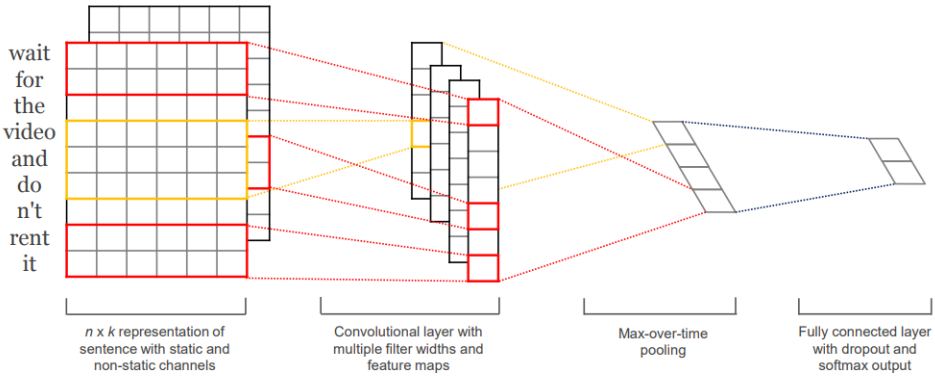

图 4-1　TextCNN 网络结构

4.1.3　模型解释

本项目主要由输入层，卷积层，池化层和输出层组成。

第一层是输入层，输入的词向量矩阵，在本项目中设置的词向量矩阵为 50 * 300，词向量维度为 300。

第二层是卷积层，在本文中我们设置 4 个卷积核，分别为 3*300，5*300，7*300，9*300，输入层分别与 4 个卷积核进行卷积操作，再使用激活函数激活，使得每个卷积核都得到了对应的 feature maps。

第三层是池化层，使用 1-max pooling 提取出每个 feature map 的最大值，然后进行级联，得到 6 维的特征表示。

第四层是输出层，本文与原论文的结构有所不同，本文采用的是线性层，对于特征进行进一步的映射，便于更好地进一步处理特征。

4.1.4　模型优点

TextCNN 最大优势网络结构简单，在模型网络结构如此简单的情况下，通过引入已经训练好的词向量依旧有很不错的效果，在多项数据数据集上超越 benchmark。网络结构简单导致参数数目少，计算量少，训练速度快，收敛速度快。同时也是工业界使用较为广泛的文本分类模型。

4.2 基于 RNN 的 BiLSTM 模型

本项目设计使用 BiLSTM 模型，其中 BiLSTM，是 Bi-directional Long Short-Term Memory 的缩写，是由前向 LSTM 与后向 LSTM 组合而成。在自然语言处理任务中常被用来建模上下文信息。在本项目的情况中，对于强程度的褒义、弱程度的褒义、中性、弱程度的贬义、强程度的贬义的五分类任务，情感粒度较细，需要注意情感词、程度词、否定词之间的交互。因此，通过 BiLSTM 可以更好的捕捉双向的语义依赖。

4.2.1 模型基础

RNN 用于处理序列数据。在传统的神经网络模型中，是从输入层到隐含层再到输出层，层与层之间是全连接的，每层之间的节点是无连接的。但是这种普通的神经网络对于很多问题却无能无力。例如，你要预测句子的下一个单词是什么，一般需要用到前面的单词，因为一个句子中前后单词并不是独立的。RNN 之所以称为循环神经网路，即一个序列当前的输出与前面的输出也有关。具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。理论上，RNN 能够对任何长度的序列数据进行处理。但是在实践中，为了降低复杂性往往假设当前的状态只与前面的几个状态相关。

LSTM，全称 Long Short Term Memory (长短期记忆) 是一种特殊的递归神经网络。这种网络与一般的前馈神经网络不同，LSTM 可以利用时间序列对输入进行分析；简而言之，当使用前馈神经网络时，神经网络会认为我们当前时刻输入的内容与其他时刻输入的内容完全无关，对于许多情况，例如图片分类识别，这是毫无问题的，可是对于一些情景，例如自然语言处理 (NLP, Natural Language Processing) 或者我们需要分析类似于连拍照片这样的数据时，合理运用之前的输入来处理当前时刻显然可以更加合理的运用输入的信息。LSTM 从被设计之初就被用于解决一般递归神经网络中普遍存在的长期依赖问题，使用 LSTM 可以有效的传递和表达长时间序列中的信息并且不会导致长时间前的有用信息被忽略（遗忘）。与此同时，LSTM 还可以解决 RNN 中的梯度消失/爆炸问题。

4.2.2 网络结构

本项目使用 BiLSTM 的框架进行学习，具体的框架如图所示。

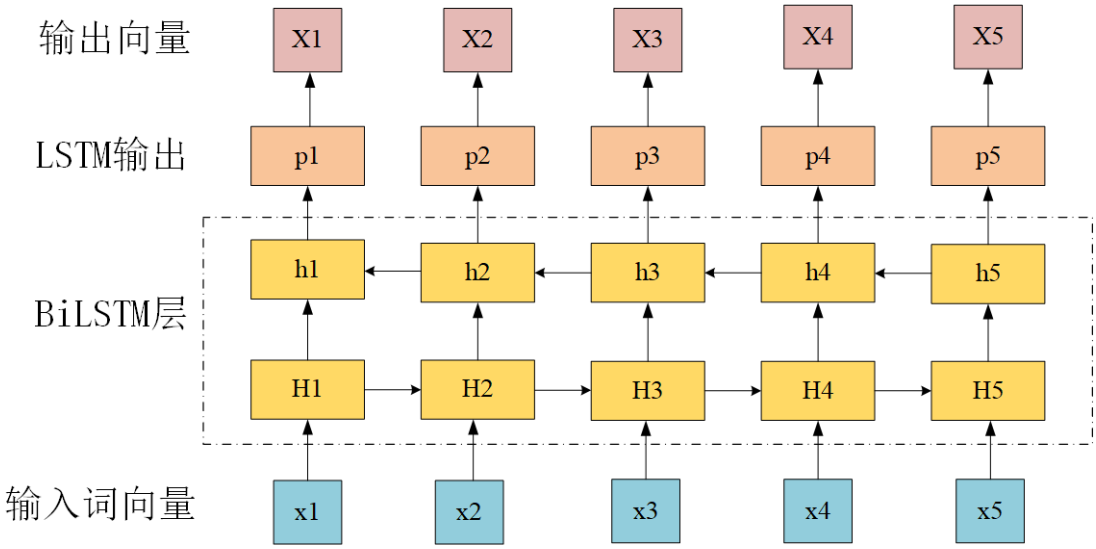

图 4-2 基于 RNN 的 BiLSTM 模型网络结构

4.2.3 模型解释

长短时记忆网络（Long Short Term Memory，简称 LSTM）模型，本质上是一种特定形式的循环神经网络（Recurrent Neural Network，简称 RNN)。LSTM 模型在 RNN 模型的基础上通过增加门限（Gates）来解决 RNN 短期记忆的问题，使得循环神经网络能够真正有效地利用长距离的时序信息。LSTM 在 RNN 的基础结构上增加了输入门限（Input Gate)、输出门限（Output Gate)、遗忘门限（Forget Gate） 3 个逻辑控制单元，且各自连接到了一个乘法元件上（见图 1），通过设定神经网络的记忆单元与其他部分连接的边缘处的权值控制信息流的输入、输出以及细胞单元（Memory cell）的状态。

与 LSTM 计算流程类似，BiLSTM 在其基础上添加了反向的运算。可以理解为把输入的序列反转，重新按照 LSTM 的方式计算一遍输出，最终的结果为正向 LSTM 的结果与反向 LSTM 的结果的简单堆叠。最终，模型实现对于上下文信息的考虑。

4.2.4 模型优点

使用了 BiLSTM 可以更好地处理输入前后的特征。

4.3 BiGRU 和注意力机制结合的深度模型

本项目自主设计将 BiGRU 和注意力机制相结合的深度模型进行训练。

4.3.1 模型基础

GRU（Gate Recurrent Unit）是循环神经网络（Recurrent Neural Network, RNN）的一种。和 LSTM（Long-Short Term Memory）一样，也是为了解决长期记忆和反向传播中的梯度等问题而提出来的。

BiGRU 是由单向的、方向相反的、输出由这两个 GRU 的状态共同决定的 GRU 组成的神经网络模型。在每一时刻，输入会同时提供两个方向相反的 GRU，而输出则由这两个单向 GRU 共同决定。

注意力机制（Attention Mechanism）是在计算能力有限的情况下，将计算资源分配给更重要的任务，同时解决信息超载问题的一种资源分配方案。在神经网络学习中，一般而言模型的参数越多则模型的表达能力越强，模型所存储的信息量也越大，但这会带来信息过载的问题。那么通过引入注意力机制，在众多的输入信息中聚焦于对当前任务更为关键的信息，降低对其他信息的关注度，甚至过滤掉无关信息，就可以解决信息过载问题，并提高任务处理的效率和准确性。

4.3.2 网络结构

本项目设计的具体框架如图所示。

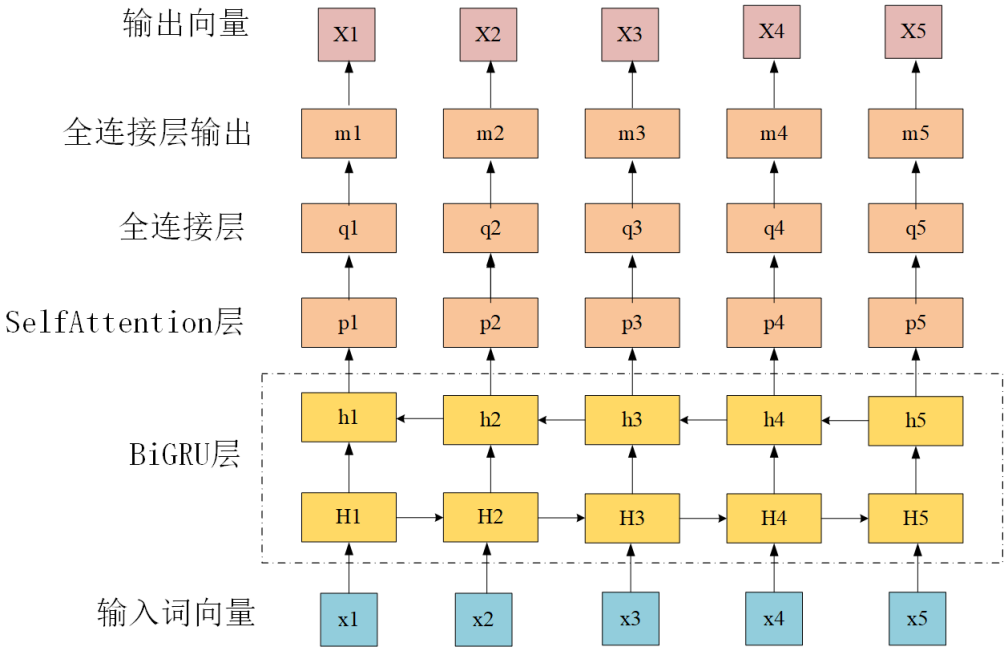

图 4-3　BiGRU 和注意力机制结合的深度模型网络结构

4.3.3　模型解释

首先，通过 BiGRU 层将信息进行有效的叠加，获取其中的有用信息。

其次，通过注意力机制的使用，将有效信息更加集中，特征抽取更加明显，模型效果更好。

4.3.4　模型优点

GRU 中引入自注意力（self-attention）机制，进一步强化了 GRU 的长时记忆能力，使其对长距离信息作用具有更好的感知力。

4.3.5　模型缺点

模型参数量大，训练难度大，收敛速度慢。

4.4　基于 Bert 的大规模预训练模型 Sibert

本项目采用的是 RoBERTa-large 的微调模型。它可以对各种类型的评论进行可靠的情感分析。对于每个实例，它预测积极（1）或消极（0）情绪。该模型对来自不同文本来源的 15 个数据集进行了微调和评估，以增强不同类型文本（评

论、推文等）的概括性。因此，当用于如下所示的新数据时，它优于仅针对一种类型的文本（例如，来自流行的 SST-2 基准测试的电影评论）训练的模型。

在本项目中，我们将该模型从 Hugging Face 官网上进行下载使用并进行一定程度上的微调，将预测目标设定为五类情绪分类。

4.4.1 模型解释

BERT 的全称为 Bidirectional Encoder Representation from Transformers，是一个预训练的语言表征模型。它强调了不再像以往一样采用传统的单向语言模型或者把两个单向语言模型进行浅层拼接的方法进行预训练，而是采用新的 masked language model（MLM），以致能生成深度的双向语言表征。BERT 论文发表时提及在 11 个 NLP（Natural Language Processing，自然语言处理）任务中获得了新的 state-of-the-art 的结果，令人目瞪口呆。

RoBERTa 是在论文《RoBERTa: A Robustly Optimized BERT Pretraining Approach》中被提出的。此方法属于 BERT 的强化版本，也是 BERT 模型更为精细的调优版本。RoBERTa 模型是 BERT 的改进版 (从其名字来看，A Robustly Optimized BERT，即简单粗暴称为强力优化的 BERT 方法)。

4.4.2 模型优点

roberta 是 bert 的一个完善版，相对于模型架构之类的都没有改变，参数量增加很多，相对应效果确实更好。

5 实验调试

本章将进行模型训练的配置，通过依次对比模型的参数，优化器，损失函数等进行模型训练的配置。

5.1 模型参数

本项目中模型参数设置主要包括 epoch、batchsize、optimizer 等，具体说明如下：

epoch 表示模型训练的次数，模型仅仅训练一次并不会得到最好的效果，因此需要多次训练，但是，epoch 并非越大越好，训练次数过多，模型会过拟合并且训练花费时间较长。选择合适的 epoch 能保证模型在设置的 epoch 内能达到较好效果，同时又避免了过大的 epoch 所导致的训练时间开销。

batchsize 表示在模型训练时，每批训练数据样本的数量。使用深度学习方法训练数据量通常较为庞大，对于有限的机器资源，无法一次训练所有数据，需要设置分批进行训练。同时，batchsize 的大小也会影响模型的优化和训练速度。

optimizer 是模型训练优化器。它能够在模型训练过程中计算和优化神经网络模型的网络参数，进而使模型达到最优状态。目前常用的优化器有 SGD55、Adagradlsl、RMSPropls7l、Adaml5s]。SGD 是基本的随机梯度下降法，其每次运算都会对每个样本进行梯度更新。Adagrad 是一种能够在训练过程中自动改变学习率的优化算法，它能针对不同的数据频率做不同幅度的更新，能提高 SGD 的鲁棒性。RMSProp 在每次更新学习步长时，都会结合之前每次的梯度变化情况，因此模型参数更新更加平缓，收敛速度更快。Adam 是一种自适应的学习算法，它融合了 Adagrad 和 RMSProp 优化器的优点，是一种更好的模型训练优化算法。在针对具体数据进行训练时，还是需要通过对比实验来选取最佳优化器。

在本项目中，epoch 为 20 轮，batchsize 为 32，optimizer 采用 Adm 算法，计算损失函数采用交叉熵损失函数。

5.2 训练策略

在本章提出了项目中所使用的训练策略，能够有效提升模型的泛化能力，有利于更好地解决问题。

5.2.1 Dropout 策略

dropout 的提出是为了防止在训练过程中的过拟合现象，那就有人想了，能不能对每一个输入样本训练一个模型，然后在 test 阶段将每个模型取均值，这样通过所有模型共同作用，可以将样本最有用的信息提取出来，而把一些噪声过滤掉。

在每一轮训练过程中，对隐含层的每个神经元以一定的概率 p 舍弃掉，这样相当于每一个样本都训练出一个模型。假设有 H 个神经元，那么就有 2H 种可能性，对应 2H 模型，训练起来时间复杂度太高。通过权重共享的方法来简化训练过程，每个样本所对应模型是部分权重共享的，只有被舍弃掉那部分权重不同。使用 dropout 可以使用使一个隐含结点不能与其它隐含结点完全协同合作，因此其它的隐含结点可能被舍弃，这样就不能通过所有的隐含结点共同作用训练出复杂的模型（只针对某一个训练样本），不能确定其它隐含结点此时是否被激活，这样就有效的防止了过拟合现象。

5.2.2 Pretrain 策略

Pretrain 使得模型的参数权重不再是随机初始化的，而是经过训练后进行的初始化，能够更好的学习文本之间的关系，有效提升模型的效果。

在本项目中我们进行预训练的方式是通过 MLM 任务。MLM，全称 “Masked Language Model”，可以翻译为 “掩码语言模型”，实际上就是一个完形填空任务，随机 Mask 掉文本中的某些字词，然后要模型去预测被 Mask 的字词。MLM 任务的示意图如图所示。

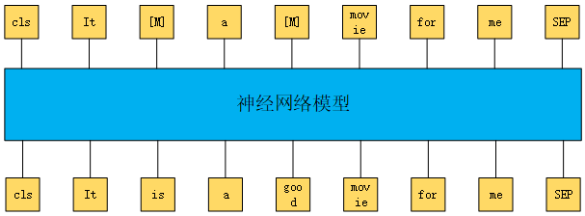

图 5-1 MLM 任务示意图

5.2.3 对抗训练策略

对抗训练（Adversarial Training）最初由 Ian Goodfellow 等人提出，作为一种防御对抗攻击的方法，其思路非常简单直接，将生成的对抗样本加入到训练集中

去，做一个数据增强，让模型在训练的时候就先学习一遍对抗样本。

在对抗训练的过程中，样本会被混合一些微小的扰动（改变很小，但是很可能造成误分类），然后使神经网络适应这种改变，从而对对抗样本具有鲁棒性。在语言模型领域尤其显著，对抗训练既提高了鲁棒性也提高了泛化性。

在本项目中使用 PGD 的方法进行模型的对抗训练，PGD（Project Gradient Descent) 攻击是一种迭代攻击，可以看作是 FGSM 的翻版——K-FGSM (K 表示迭代的次数)，大概的思路就是，FGSM 是仅仅做一次迭代，走一大步，而 PGD 是做多次迭代，每次走一小步，每次迭代都会将扰动 clip 到规定范围内。其迭代公式如下：

$$x_{t+1} = \prod_{x+S} \left(x_t + \alpha \cdot \operatorname{sign}\left(\nabla_x J\left(x_t, y\right)\right)\right) \tag{5.1}$$

PGD 攻击是最强的一阶攻击，如果防御方法对这个攻击能够有很好的防御效果，那么其他攻击也不在话下了。因此，PGD 攻击能够对模型效果产生显著的提升。

5.2.4 SWA 策略

SWA (Stochastic Weight Averaging) 是一种用简单的方法，利用随机梯度下降法就能够提升深度学习模型的泛化能力的方法，而且不需要多余的代价，通过可调整的学习率的 SGD 算法得到的多个权重进行平均的方法。SWA 最终的结果就是在一个广阔的平坦的 loss 区域的中心，而 SGD 趋向于收敛到低 loss 区域的边界，这使得结果容易受到训练和测试误差平面之间转换的影响，即在低 loss 区域的边界可能在测试集合误差平面较大的位置，也就是模型不够鲁棒。

SWA 能够显著的提高语言任务的泛化能力，能够提高训练的稳定性，也提高了策略梯度法的最终平均奖励。

5.3 模型训练

（1）epoch 参数的选择对于前三个模型分别进行训练，对于基于 CNN 的文本分类模型 TextCNN 进行训练时，模型的训练集和验证集的损失相对于 epoch 如图所示。

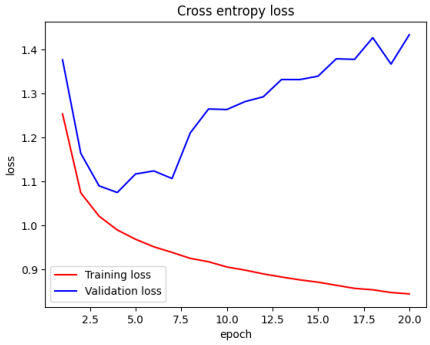

图 5-2　TextCNN 训练集和验证集的损失

对于基于 RNN 的 BiLSTM 模型进行训练时，模型的训练集和验证集的损失相对于 epoch 如图所示。

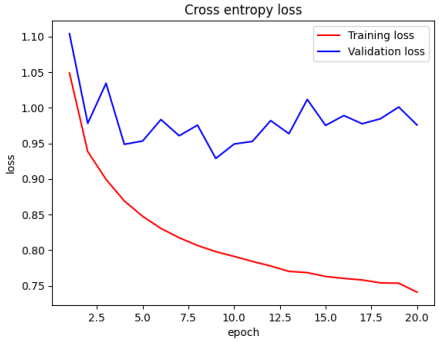

图 5-3　BiLSTM 训练集和验证集的损失

对于 BiGRU 和注意力机制结合的深度模型进行训练时，模型的训练集和验证集的损失相对于 epoch 如图所示。

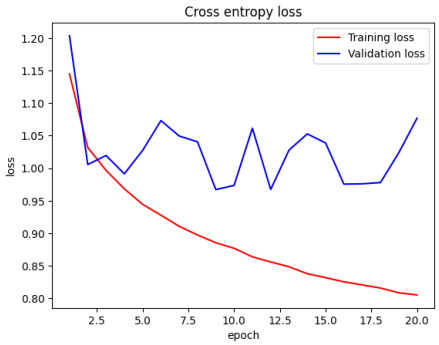

图 5-4　BiGRU 和注意力机制结合的深度模型训练集和验证集的损失

通过图像可知，设置 epoch 为 20 轮较为合理，能够很好的训练模型，同时不会过于拟合。

（2）batchsize 参数的选择对于三个模型分别进行训练和实验，最终确定使用 32 作为 batchsize 可以获得最好的效果。

（3）optimizer 参数的选择在本项目中，我们选择 Adam 算法作为 optimizer 进行试验。Adam 算法是一种基于“momentum”思想的随机梯度下降优化方法，通过迭代更新之前每次计算梯度的一阶 moment 和二阶 moment，并计算滑动平均值，后用来更新当前的参数。这种思想结合了 Adagrad 算法的处理稀疏型数据，又结合了 RMSProp 算法的可以处理非稳态的数据。最后在传统的凸优化问题和最近火热的深度学习优化问题中，取得了非常好的测试性能。达到了当时的 state-of-the-art 的评价。

具体的训练过程示意图如图所示。

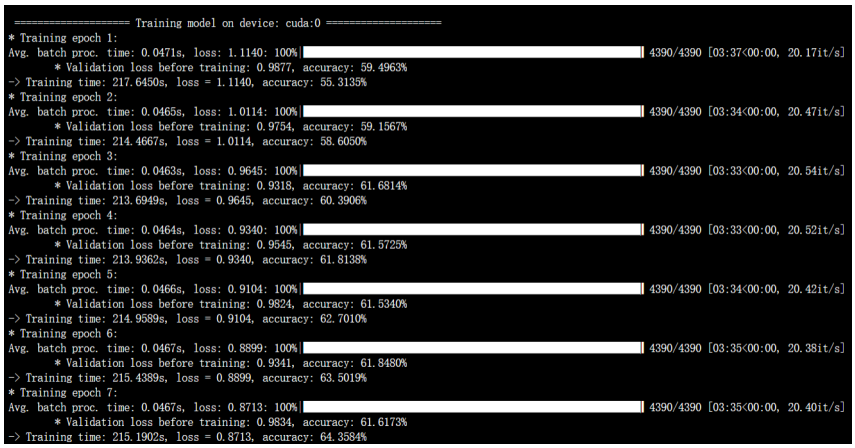

图 5-5　训练过程

5.4　模型效果

通过训练好的模型后，对于测试集进行预测，通过 kaggle 进行评测，获取最终结果如表所示。

表 5-1 模型测试效果

所使用的模型	最终得分
全连接层	0.6168
RNN	0.62485
LSTM	0.61717
TextCNN	0.63487
TextCNN+pgd	0.65012
BiLSTM	0.64481
BiLSTM+pgd	0.6551
BiGRU+SelfAttention	0.65589
BiGRU+SelfAttention+pgd	0.66148
SiBert	0.70886
SiBert+pgd	0.71607

通过最终实验结果可知，模型获得的最优实验结果为 0.71607，单模型获得的最优实验结果为 0.66148。

5.5 模型改进

（1）模型的进一步 pretrain 没有取得更好的效果，究其原因可能是模型对于上下文的信息已经获取的很好了，pretrain 已经无法获得更好的效果，同时还会造成收敛速度慢，很难获得更好的效果。

（2）进一步考虑模型集成可能会考虑更好的模型。

（3）由于训练资源有限，模型无法在深度上获取优势，因此设计的模型没有在深度上进行进一步的实验，只能在模型结构上进行提高，进一步可以考虑加深模型深度，使得模型发挥更大的优势。

6 实验总结和感想

本项目基于 Kaggle 提供的 Sentiment Analysis on Movie Reviews 数据集，采用深度学习方法的对烂番茄电影评论进行情感分析，并将其划分为消极，有点消极，中性，有点积极，积极。

CNN（卷积神经网络）、RNN（循环神经网络）等，是用于文本情感分析的最常用深度学习模型。我们从这些模型入手，构建了 RNN，LSTM，BiLSTM，BiGRU,TextCNN，SiBert 等一系列模型，分析对比它们的效果，为进一步优化做准备。同时，采用 BiGRU 和注意力机制相结合的深度模型进行训练，PGD 的方法进行模型的对抗训练作进一步优化，均取得了不错的效果。最后，通过 kaggle 进行评测，获取最终结果，模型获得的最优实验结果为 0.71607，单模型获得的最优实验结果为 0.66148。

通过本次实验，我们对课堂上老师所讲解的 CNN、RNN，LSTM 等深度学习算法进行了实践。同时，通过搜集资料学习，掌握了自然语言处理情感分析常见方法。之后，我们通过多种技巧提高模型的效果，实现对于数据处理和深度学习的相关内容进行深入学习和提高。

通过对实验结果的不断优化，我们体会到了不断进步的快乐。这也要求我们要在以后的学习生活中精益求精，不断深入学习，丰富自己，提高自己。

参考文献

[1] 梁斌, 刘全, 徐进, 周倩, 章鹏. 基于多注意力卷积神经网络的特定目标情感分析 [J]. 计算机研究与发展,2017,54(08):1724-1735.

[2] Greff Klaus,Srivastava Rupesh K,Koutnik Jan,Steunebrink Bas R,Schmidhuber Jurgen. LSTM: A Search Space Odyssey.[J]. IEEE transactions on neural networks and learning systems,2017,28(10).

[3] Vaswani A , Shazeer N , Parmar N , et al. Attention Is All You Need[J]. arXiv, 2017.

[4] Jeffrey L. Elman. Finding structure in time[J]. Cognitive Science,1990,14(2).

[5] Ma Zerui,Zhao Linna,Li Jianqiang,Xu Xi,Li Jing. SiBERT: A Siamese-based BERT Network for Chinese Medical Entities Alignment.[J]. Methods (San Diego, Calif.),2022,205.

[6] Eliyahu Kiperwasser,Yoav Goldberg. Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations[J]. Transactions of the Association for Computational Linguistics,2016,4.

A 附录主要代码

模型整体框架

```
import torch
import torch.nn as nn
import warnings

warnings.filterwarnings('ignore')

from layers import RNNDropout, PositionalEncoding, SelfAttention,
    MaskLM, MLMHead, Bert, SiBert, TextCNN
from config import Config
import torch.nn.functional as F

class Mymodel(nn.Module):
    """
    Implementation of the model.
    """

    def __init__(self,
                 vocab_size,
                 embedding_dim,
                 hidden_size,
                 embeddings=None,
                 padding_idx=0,
                 dropout=0.5,
                 num_classes=5,
                 device="cpu",
                 mode = "train"):
        """
        Args:
            vocab_size: The size of the vocabulary of embeddings
                in the model.
            embedding_dim: The dimension of the word embeddings.
            hidden_size: The size of all the hidden layers in the
```

```
            network.
        embeddings: A tensor of size (vocab_size,
            embedding_dim) containing
            pretrained word embeddings. If None, word
                embeddings are
            initialised randomly. Defaults to None.
        padding_idx: The index of the padding token in the
            sentences
            passed as input to the model. Defaults to 0.
        dropout: The dropout rate to use between the layers
            of the network.
            A dropout rate of 0 corresponds to using no
                dropout at all.
            Defaults to 0.5.
        num_classes: The number of classes in the output of
            the network.
            Defaults to 3.
        device: The name of the device on which the model is
            being
            executed. Defaults to 'cpu'.
    """
    super(Mymodel, self).__init__()

    self.vocab_size = vocab_size
    self.embedding_dim = embedding_dim
    self.hidden_size = hidden_size
    self.num_classes = num_classes
    self.dropout = dropout
    self.device = device
    self.config = Config()
    self.mode = mode

    self._masklm = MaskLM(self.config, mlm_probability=0.25)

    self._mlmhead = MLMHead(self.config, self.hidden_size,
```

```
            self.vocab_size)
58
59          # self._bert = Bert(self.config, mode=mode)
60
61          self._word_embedding = nn.Embedding(self.vocab_size,
62                                              self.embedding_dim,
63                                              padding_idx=
                                                    padding_idx,
64                                              _weight=embeddings)
65
66          self._textcnn = TextCNN(self.config)
67
68          self._dense = nn.Sequential(
69              nn.Dropout(self.dropout),
70              nn.Linear(self.hidden_size, self.hidden_size * 2),
71              nn.GELU(),
72              nn.Dropout(self.dropout),
73              nn.Linear(self.hidden_size * 2, self.hidden_size)
74          )
75
76          self._positional_encoding = PositionalEncoding(
               embedding_dim,dropout=0.2,max_len=100)
77
78          if self.dropout:
79              self._rnn_dropout = RNNDropout(p=self.dropout)
80              # self._rnn_dropout = nn.Dropout(p=self.dropout)
81
82          self._lstm = nn.LSTM(self.embedding_dim,
83                               self.hidden_size,
84                               num_layers = 1,
85                               bias = True,
86                               batch_first=True,
87                               dropout=dropout,
88                               bidirectional=True)
89
```

```
90          self._GRU = nn.GRU(self.embedding_dim,
91                             self.hidden_size,
92                             num_layers = 1,
93                             bias = True,
94                             batch_first=True,
95                             dropout=dropout,
96                             bidirectional=True)
97
98          self._selfattention = SelfAttention(self.config, self.
                hidden_size)
99
100         self._turn = nn.Linear(self.hidden_size * 2, self.
                hidden_size)
101
102         self._classification = nn.Sequential(nn.Dropout(p=self.
                dropout),nn.Tanh(),nn.Dropout(p=self.dropout),nn.
                Linear(self.hidden_size * 50,self.num_classes))
103
104         self._classification1 = nn.Sequential(nn.Dropout(p=self.
                dropout),nn.Tanh(),nn.Dropout(p=self.dropout), nn.
                Linear(self.hidden_size * 2, self.num_classes))
105
106         self._classification2 = nn.Sequential(nn.Dropout(p=self.
                dropout),nn.Tanh(),nn.Dropout(p=self.dropout), nn.
                Linear(self.hidden_size, self.num_classes))
107
108         # self._sibert = SiBert(self.config.checkpoint)
109         if mode == 'pretrain':
110             self.lm = MaskLM(self.config)
111             self.mlm_head = MLMHead(self.config, self.hidden_size
                    , self.vocab_size)
112
113         # Initialize all weights and biases in the model.
114         self.apply(_init_esim_weights)
115
```

```
116     def forward(self,
117                 sentences,
118                 sentences_lengths,
119                 model = None):
120         """
121         Args:
122             sentences: A batch of varaible length sequences of
                    word indices
123                 representing premises. The batch is assumed to be
                        of size
124                 (batch, sentences_lengths).
125             sentences_lengths: A ID tensor containing the lengths
                    of the
126                 sentences in 'sentences'.
127             model: A string incading the state of the training
                    model.Default to
128                 None.
129 
130         Returns:
131             logits: A tensor of size (batch, num_classes)
                    containing the
132                 logits for each output class of the model.
133             probabilities: A tensor of size (batch, num_classes)
                    containing
134                 the probabilities of each output class in the
                        model.
135         """
136         if model == "normal":
137             # ipdb.set_trace()
138             if self.mode == 'pretrain':
139                 device = sentences.device
140                 input_ids_masked, lm_label = self.lm.
                        torch_mask_tokens(sentences.cpu())
141                 sentences = input_ids_masked.to(device)
142                 lm_label = lm_label[:, 1:].to(device)
```

```
143             sentences = self._word_embedding(sentences)
144
145             sentences = self._rnn_dropout(sentences)
146             sentences = self._dense(sentences)
147
148             sentences, (h_n, c_n) = self._GRU(sentences)
149             sentences = self._turn(sentences)
150             sentences = self._positional_encoding(sentences)
151             attention_mask = torch.ones(sentences.shape[0],
                    sentences.shape[1]).to(self.device)
152             sentences = self._selfattention(sentences,
                    attention_mask)
153
154             if self.mode == "pretrain":
155                 lm_prediction_scores = self.mlm_head(sentences)
                        [:, 1:, :]
156                 # ipdb.set_trace()
157                 pred = lm_prediction_scores.contiguous().view(-1,
                         self.vocab_size)
158                 lm_loss = F.cross_entropy(pred, lm_label.
                        contiguous().view(-1))
159                 return lm_loss
160
161             sentences = sentences.contiguous().view(sentences.
                    size()[0], -1)
162
163             logits = self._classification(sentences)
164             probabilities = nn.functional.softmax(logits, dim=-1)
165             return logits, probabilities
166
167         elif model == "testcnn":
168             sentences = self._word_embedding(sentences)
169             sentences = self._rnn_dropout(sentences)
170             # sentences = self._dense(sentences)
171             sentences = self._textcnn(sentences)
```

```
172             sentences = sentences.contiguous().view(sentences.
                    size()[0], -1)
173
174             logits = self._classification2(sentences)
175             probabilities = nn.functional.softmax(logits, dim=-1)
176             return logits, probabilities
177
178         elif model == "bilstm":
179             if self.mode == 'pretrain':
180                 device = sentences.device
181                 input_ids_masked, lm_label = self.lm.
                        torch_mask_tokens(sentences.cpu())
182                 sentences = input_ids_masked.to(device)
183                 lm_label = lm_label[:, 1:].to(device)
184             sentences = self._word_embedding(sentences)
185             sentences = self._rnn_dropout(sentences)
186             # sentences = self._dense(sentences)
187
188             sentences, (h_n, c_n) = self._lstm(sentences)
189
190             if self.mode == "pretrain":
191                 lm_prediction_scores = self.mlm_head(sentences)
                        [:, 1:, :]
192                 pred = lm_prediction_scores.contiguous().view(-1,
                         self.vocab_size)
193                 lm_loss = F.cross_entropy(pred, lm_label.
                        contiguous().view(-1))
194                 return lm_loss
195
196             sentences = sentences.contiguous().view(sentences.
                    size()[0], -1)
197
198             logits = self._classification1(sentences)
199             probabilities = nn.functional.softmax(logits, dim=-1)
200             return logits, probabilities
```

```
201
202         elif model == "bert":
203             if self.mode == "pretrain":
204                 attention_mask = torch.ones(sentences.shape[0],
                        sentences.shape[1])
205                 loss = self._bert(sentences, attention_mask)
206                 return loss
207             attention_mask = torch.ones(sentences.shape[0],
                    sentences.shape[1])
208             logits, probabilities = self._bert(sentences,
                    attention_mask)
209             return logits, probabilities
210         else:
211             attention_mask = torch.ones(sentences.shape[0],
                    sentences.shape[1]).to(device)
212             out = self._sibert(sentences, attention_mask)
213             logits = out.logits
214             probabilities = nn.functional.softmax(logits, dim=-1)
215             return logits, probabilities
216
217 # init in proper and excellent way, which can be learned from the
218 def _init_esim_weights(module):
219     """
220     Initialise the weights of the Mymodel.
221     """
222     if isinstance(module, nn.Linear):
223         nn.init.xavier_uniform_(module.weight.data)
224         nn.init.constant_(module.bias.data, 0.0)
225
226     elif isinstance(module, nn.LSTM):
227         nn.init.xavier_uniform_(module.weight_ih_l0.data)
228         nn.init.orthogonal_(module.weight_hh_l0.data)
229         nn.init.constant_(module.bias_ih_l0.data, 0.0)
230         nn.init.constant_(module.bias_hh_l0.data, 0.0)
231         hidden_size = module.bias_hh_l0.data.shape[0] // 4
```

```
232         module.bias_hh_l0.data[hidden_size:(2*hidden_size)] = 1.0
233
234         if (module.bidirectional):
235             nn.init.xavier_uniform_(module.weight_ih_l0_reverse.
                data)
236             nn.init.orthogonal_(module.weight_hh_l0_reverse.data)
237             nn.init.constant_(module.bias_ih_l0_reverse.data,
                0.0)
238             nn.init.constant_(module.bias_hh_l0_reverse.data,
                0.0)
239             module.bias_hh_l0_reverse.data[hidden_size:(2*
                hidden_size)] = 1.0
```

模型具体组件

```
1  import torch.nn as nn
2  import torch
3  from torch.autograd import Variable
4  import math
5  from config import Config
6  import torch.nn.functional as F
7  import pickle
8  import copy
9  from transformers.models.bert.modeling_bert import (
10     BertModel, BertConfig, BertOnlyMLMHead
11 )
12
13 from datasets import load_dataset
14 from datasets import load_metric
15
16 from transformers import AutoModelForSequenceClassification
17
18 import warnings
19
20 warnings.filterwarnings('ignore')
21
22 class RNNDropout(nn.Dropout):
```

```
23      """
24      Dropout layer for the inputs of RNNs.
25
26      Apply the same dropout mask to all the elements of the same
            sequence in
27      a batch of sequences of size (batch, sequences_length,
            embedding_dim).
28      """
29
30      def forward(self, sequences_batch):
31          """
32          Apply dropout to the input batch of sequences.
33
34          Args:
35              sequences_batch: A batch of sequences of vectors that
                    will serve
36                  as input to an RNN.
37                  Tensor of size (batch, sequences_length,
                        emebdding_dim).
38                  #torch.Size([32, 61, 300])
39          Returns:
40              A new tensor on which dropout has been applied.
41          """
42          # torch.Size([32, 300])
43          ones = sequences_batch.data.new_ones(sequences_batch.
                shape[0],
44                                               sequences_batch.
                                                    shape[-1])
45          dropout_mask = nn.functional.dropout(ones, self.p, self.
                training,
46                                               inplace=False)
47          # torch.Size([32, 300])
48          return dropout_mask.unsqueeze(1) * sequences_batch
49
50  class PositionalEncoding(nn.Module):
```

```
51      "Implement the PE function."
52      def __init__(self, d_model, dropout, max_len=500):
53          super(PositionalEncoding, self).__init__()
54          self.dropout = nn.Dropout(p=dropout)
55
56          # Compute the positional encodings once in log space.
57          pe = torch.zeros(max_len, d_model)
58          position = torch.arange(0, max_len).unsqueeze(1)
59          div_term = torch.exp(torch.arange(0, d_model, 2) *
60                               -(math.log(10000.0) / d_model))
61          pe[:, 0::2] = torch.sin(position * div_term)
62          pe[:, 1::2] = torch.cos(position * div_term)
63          pe = pe.unsqueeze(0)
64          self.register_buffer('pe', pe)
65
66      def forward(self, x):
67          x = x + Variable(self.pe[:, :x.size(1)],
68                           requires_grad=False)
69          return self.dropout(x)
70
71  class SelfAttention(nn.Module):
72      def __init__(self, config: Config, hidden_size) -> None:
73          super(SelfAttention, self).__init__()
74          self.hidden_size = hidden_size
75
76          self.num_attention_heads = config.num_attention_heads
77          self.attention_head_size = hidden_size // config.
                num_attention_heads
78          self.all_head_size = self.num_attention_heads*self.
                attention_head_size
79
80          self.query = nn.Linear(hidden_size, self.all_head_size)
81          self.key = nn.Linear(hidden_size, self.all_head_size)
82          self.value = nn.Linear(hidden_size, self.all_head_size)
83          self.dropout1 = nn.Dropout(config.dropout_prob)
```

```
84          self.linear = nn.Linear(self.all_head_size, self.
                hidden_size)
85          self.dropout2 = nn.Dropout(config.dropout_prob)
86
87      def transpose(self, x: torch.Tensor):
88          #[bs,length,hidden_size]->[bs,num_heads,length,
                att_hidden_size]
89          new_shape = x.size()[:-1]+(self.num_attention_heads, self
                .attention_head_size)
90          x = x.view(new_shape)
91          return x.permute(0, 2, 1, 3)
92
93      def forward(self, hidden_states, attention_mask : torch.
            Tensor):
94          key_layer = self.transpose(self.key(hidden_states))
95          value_layer = self.transpose(self.value(hidden_states))
96          quert_layer = self.transpose(self.query(hidden_states))
97          attention_scores = torch.matmul(quert_layer, key_layer.
                transpose(-1, -2))
98          attention_scores = attention_scores / math.sqrt(self.
                attention_head_size)
99
100         attention_mask = attention_mask[:, None, None, :]
101         attention_mask = (1-attention_mask)*(-10000.0)
102         attention_scores = attention_scores + attention_mask
103         attention_probs = F.softmax(attention_scores, dim=-1)
104         attention_probs = self.dropout1(attention_probs)
105
106         context_layer = torch.matmul(attention_probs, value_layer
                )
107         context_layer = context_layer.permute(0, 2, 1, 3).
                contiguous()
108         new_shape = context_layer.size()[:-2]+(self.all_head_size
                ,)
109         context_layer = context_layer.view(new_shape)
```

```
110         return self.dropout2(self.linear(context_layer))
111
112 class FeedForward(nn.Module):
113     def __init__(self, config: Config, hidden_size) -> None:
114         super().__init__()
115         self.dense = nn.Sequential(
116             nn.Dropout(config.dropout_prob),
117             nn.Linear(hidden_size, config.feed_forward_size),
118             nn.GELU(),
119             nn.Dropout(config.dropout_prob),
120             nn.Linear(config.feed_forward_size, hidden_size)
121         )
122
123     def forward(self, x):
124         return self.dense(x)
125
126 class TextCNN(nn.Module):
127     def __init__(self, config: Config, kersize=[3, 5, 7, 9],
        hidden_size = 300) -> None:
128         super().__init__()
129         self.convs = nn.ModuleList([
130             nn.Sequential(
131                 nn.Conv1d(hidden_size, config.cnn_hidden_size,
                    kernel_size=k),
132                 nn.ReLU(),
133                 nn.AdaptiveMaxPool1d(1)
134             ) for k in kersize])
135         self.classifier = nn.Sequential(
136             nn.Linear(config.cnn_hidden_size*len(kersize),
                hidden_size),
137             nn.Dropout(config.dropout_prob)
138         )
139
140     def forward(self, inputs):
141         embeds = inputs.permute(0, 2, 1)
```

```
142         out = [conv(embeds) for conv in self.convs]
143         out = torch.concat(out, dim=1)
144         out = out.view(-1, out.shape[1])
145         out = self.classifier(out)
146         return out
147
148 class PGD():
149     def __init__(self, model):
150         self.model = model
151         self.emb_backup = {}
152         self.grad_backup = {}
153
154     def attack(self, epsilon=1., alpha=0.3, emb_name='
        _word_embedding.weight', is_first_attack=False):
155         # emb_name这个参数要换成你模型中embedding的参数名
156         for name, param in self.model.named_parameters():
157             if param.requires_grad and param.grad is not None and
                    emb_name in name:
158                 if is_first_attack:
159                     self.emb_backup[name] = param.data.clone()
160                 norm = torch.norm(param.grad)
161                 if norm != 0:
162                     r_at = alpha * param.grad / norm
163                     param.data.add_(r_at)
164                     param.data = self.project(name, param.data,
                            epsilon)
165
166     def restore(self, emb_name='_word_embedding.weight'):
167         # emb_name这个参数要换成你模型中embedding的参数名
168         for name, param in self.model.named_parameters():
169             if param.requires_grad and emb_name in name:
170                 assert name in self.emb_backup
171                 param.data = self.emb_backup[name]
172         self.emb_backup = {}
173
```

```
174     def project(self, param_name, param_data, epsilon):
175         r = param_data - self.emb_backup[param_name]
176         if torch.norm(r) > epsilon:
177             r = epsilon * r / torch.norm(r)
178         return self.emb_backup[param_name] + r
179 
180     def backup_grad(self):
181         for name, param in self.model.named_parameters():
182             if param.requires_grad and param.grad is not None:
183                 self.grad_backup[name] = param.grad.clone()
184 
185     def restore_grad(self):
186         for name, param in self.model.named_parameters():
187             if param.requires_grad and param.grad is not None:
188                 param.grad = self.grad_backup[name]
189 
190 # def get_model_path_list(base_dir):
191 #     """
192 #     从文件夹中获取 model.pt 的路径
193 #     """
194 #     model_lists = []
195 
196 #     for fname in os.listdir(base_dir):
197 #         if 'Sentiment' in fname:
198 #             model_lists.append(base_dir+fname)
199 
200 #     model_lists = sorted(model_lists)
201 
202 #     return model_lists[:]
203 
204 # def SWA(model, base_dir='../../data/checkpoints'):
205 #     """
206 #     swa 滑动平均模型，一般在训练平稳阶段再使用 SWA
207 #     """
208 #     model_path_list = get_model_path_list(base_dir)
```

```
209 #     print(f'mode_list:{model_path_list}')
210 
211 #     swa_model = copy.deepcopy(model)
212 #     swa_n = 0.
213 
214 #     with torch.no_grad():
215 #         start_epoch, best_score = 0, 0
216 #         for _ckpt in model_path_list:
217 
218 #             checkpoint = torch.load(checkpoint)
219 #             start_epoch = max(start_epoch, checkpoint["epoch"]
    + 1)
220 #             best_score = max(best_score, checkpoint["best_score
    "])
221 
222 #             model.load_state_dict(checkpoint["model"])
223 #             optimizer.load_state_dict(checkpoint["optimizer"])
224 #             epochs_count = checkpoint["epochs_count"]
225 #             train_losses = checkpoint["train_losses"]
226 #             valid_losses = checkpoint["valid_losses"]
227 #             checkpoint = torch.load(_ckpt)
228 
229 #             tmp_para_dict = dict(model.named_parameters())
230 
231 #             alpha = 1. / (swa_n + 1.)
232 
233 #             for name, para in swa_model.named_parameters():
234 #                 para.copy_(tmp_para_dict[name].data.clone() *
    alpha + para.data.clone() * (1. - alpha))
235 
236 #             swa_n += 1
237 
238 #     # use 100000 to represent swa to avoid clash
239 
240 
```

```
241 #         torch.save({"epoch": epoch,
242 #                     "model": model.state_dict(),
243 #                     "best_score": best_score,
244 #                     "optimizer": optimizer.state_dict(),
245 #                     "epochs_count": epochs_count,
246 #                     "train_losses": train_losses,
247 #                     "valid_losses": valid_losses},
248 #                    os.path.join(target_dir, "swa_Sentiment_{}.pth.
    tar".format(epoch)))
249
250 #     return swa_model
251
252 class Embeddings(nn.Module):
253     def __init__(self, config: Config, embedding_size) -> None:
254         super().__init__()
255         self.word_embeddings = nn.Embedding(
256             config.vocab_size, embedding_size, padding_idx=config
                .pad_token_id)
257         self.position_embeddings = nn.Embedding(
258             config.max_position_embeddings, embedding_size)
259         self.layerNorm = nn.LayerNorm(embedding_size, eps=config.
            layer_norm_eps)
260         self.dropout = nn.Dropout(config.dropout_prob)
261
262     def forward(self, input_ids: torch.Tensor, positions_ids=None
        ):
263         bs, length = input_ids.size()
264         if positions_ids is None:
265             positions_ids = torch.arange(length).expand((bs, -1))
                .to(input_ids.device)
266         input_embeds = self.word_embeddings(input_ids)
267         positions_embeds = self.position_embeddings(positions_ids
            )
268         embeddings = input_embeds+positions_embeds
269         embeddings = self.dropout(self.layerNorm(embeddings))
```

```
270         return embeddings
271
272 class MaskLM():
273     def __init__(self, config: Config, mlm_probability=0.25):
274         self.mlm_probability = mlm_probability
275         # self.tokenizer = TokenizerEN(config, build_vocab=False)
276         with open("../../data/preprocessed/worddict.pkl", "rb")
                as pkl:
277             self.worddict = pickle.load(pkl)
278
279     def torch_mask_tokens(self, inputs: torch.Tensor):
280
281         labels = inputs.clone()
282
283         probability_matrix = torch.full(labels.shape, self.
                mlm_probability)
284         # special_tokens_mask = [
285         #     self.tokenizer.get_special_tokens_mask(val) for val
                in labels.tolist()
286         # ]
287         # special_tokens_mask = torch.tensor(
288         #     special_tokens_mask, dtype=torch.bool)
289
290         probability_matrix.masked_fill_(probability_matrix, value
                =0.0)
291         masked_indices = torch.bernoulli(probability_matrix).bool
                ()
292         labels[~masked_indices] = -100
293
294         # 80% MASK
295         indices_replaced = torch.bernoulli(torch.full(labels.
                shape, 0.8)).bool() & masked_indices
296         inputs[indices_replaced] = 2 #self.worddict["_MASK_"]
297
298         # 10% random
```

```
299             indices_random = torch.bernoulli(torch.full(labels.shape,
                    0.5)).bool() & \
300                 masked_indices & ~indices_replaced
301             # ipdb.set_trace()
302             random_words = torch.randint(
303                 len(self.worddict), labels.shape, dtype=torch.long)
304             inputs[indices_random] = random_words[indices_random]
305
306             # 10% original
307             return inputs, labels
308
309 class MLMHead(nn.Module):
310     def __init__(self, config: Config, hidden_size,  vocab_size)
            -> None:
311         super().__init__()
312         self.dense = nn.Sequential(
313             nn.Linear(hidden_size, hidden_size),
314             nn.GELU(),
315             nn.LayerNorm(hidden_size, eps=config.layer_norm_eps),
316             nn.Linear(hidden_size, vocab_size)
317         )
318
319     def forward(self, sequence_output):
320         return self.dense(sequence_output)
321
322 class Bert(nn.Module):
323     def __init__(self, config: Config, mode = 'train') -> None:
324         super().__init__()
325         bert_config = BertConfig.from_pretrained(
326             config.bert_name, cache_dir=config.bert_cache)
327         bert = BertModel.from_pretrained(
328             config.bert_name, cache_dir=config.bert_cache)
329         self.embeddings = Embeddings(config, bert_config.
                hidden_size)
330
```

```
331         self.encoder = copy.deepcopy(bert.encoder)
332 
333         self.mode = mode
334         if mode == 'pretrain':
335             bert_config.vocab_size = config.vocab_size
336             self.vocab_size = config.vocab_size
337             self.mlm_head = BertOnlyMLMHead(bert_config)
338             self.lm = MaskLM(config)
339 
340         self.classifier = nn.Linear(bert_config.hidden_size, 5)
341 
342     def forward(self, inputs, attention_mask):
343         if self.mode == 'pretrain':
344             device = inputs.device
345             input_ids_masked, lm_label = self.lm.
                    torch_mask_tokens(inputs.cpu())
346             inputs = input_ids_masked.to(device)
347             lm_label = lm_label[:, 1:].to(device)
348 
349         inputs_embeds = self.embeddings(inputs)
350         mask_expanded = attention_mask[:, None, None, :]
351         mask_expanded = (1-mask_expanded)*(-10000.0)
352 
353         outputs = self.encoder(
354             inputs_embeds, attention_mask=mask_expanded)['
                    last_hidden_state']
355 
356         if self.mode == 'pretrain':
357             lm_prediction_scores = self.mlm_head(outputs)[:, 1:,
                    :]
358             pred = lm_prediction_scores.contiguous().view(-1,
                    self.vocab_size)
359             lm_loss = F.cross_entropy(pred, lm_label.contiguous()
                    .view(-1))
360             return lm_loss
```

```
361
362         sentences = outputs[:, 0, :]
363         # mask = attention_mask.unsqueeze(-1).expand(outputs.
                shape).float()
364         # sum_embeds = torch.sum(outputs*mask, dim=1)
365         # sum_mask = torch.sum(mask, dim=1).clamp(min=1e-9)
366         # features_mean = sum_embeds/sum_mask
367         logits = self.classifier(sentences)
368         probabilities = nn.functional.softmax(logits, dim=-1)
369
370         return logits, probabilities
371
372 class SiBert(nn.Module):
373     def __init__(self, checkpoint):
374         super(SiBert, self).__init__()
375         self._model = AutoModelForSequenceClassification.
              from_pretrained(checkpoint, num_labels=5,
              ignore_mismatched_sizes=True)
376     def forward(self, sentences, attention_mask):
377         return self._model(sentences, attention_mask)
```