教程 三 : 基于自注意力机制的情感分析模型

提交日期：2024-08-24

提交人：张健豪

指导教师：秦科

目录

[**1.** **题目** 3](#_Toc175397271)

[**2.** **算法原理描述** 3](#_Toc175397272)

[2.1 数据预处理 3](#_Toc175397273)

[2.2 模型结构 3](#_Toc175397274)

[2.3 训练与优化 3](#_Toc175397275)

[2.4 模型评估 4](#_Toc175397276)

[**3.** **实验结果与分析** 4](#_Toc175397277)

[3.1 实验数据 4](#_Toc175397278)

[3.2 实验设置 4](#_Toc175397279)

[3.3 实验结果与分析 5](#_Toc175397280)

[**4.** **总结** 5](#_Toc175397281)

[**5.** **参考文献** 6](#_Toc175397282)

[**6.** **完整代码** 6](#_Toc175397283)

# **题目**

基于自注意力机制的情感分析模型

# **算法原理描述**

## 2.1 数据预处理

数据源来自压缩文件train.tsv.zip，其中包含了评论短语及其对应的情感标签。为了对模型进行训练和验证，我们将数据集按比例切分，前80%用于训练，后20%用于测试。

在处理文本数据时，我们将每个评论短语中的字符转换为ASCII码，作为模型的输入。由于评论长度不一，我们对序列进行了填充（padding）处理，使它们在一个批次内具有相同的长度。同时，我们使用了PyTorch的pack\_padded\_sequence函数对序列进行压缩，避免冗余的填充数据影响模型性能。

## 2.2 模型结构

我们使用了一个双向GRU（Gated Recurrent Unit）作为模型的核心结构。GRU是一种改进型的RNN，通过门控机制（update gate和reset gate）解决了传统RNN在处理长序列时可能遇到的梯度消失问题。模型结构主要包括以下几部分：

嵌入层（Embedding Layer）：将输入的字符序列映射为低维稠密向量，以便捕捉字符之间的语义关系。

双向GRU层：通过双向结构同时考虑序列的前向和后向信息，增强了对上下文的理解能力。模型使用了多层GRU来提升表达能力。

全连接层（Fully Connected Layer）：将GRU层输出的隐藏状态连接至分类层，输出最终的情感类别。

## 2.3 训练与优化

模型采用交叉熵损失函数（CrossEntropyLoss）作为优化目标，使用Adam优化器进行参数更新。训练过程包括多轮迭代，每轮训练后对模型在测试集上的表现进行验证，以评估模型的泛化能力。

## 2.4 模型评估

模型的准确率通过测试集进行计算。训练完成后，我们绘制了每个epoch的准确率变化曲线，以观察模型的收敛情况。此外，我们保存了在测试集上表现最优的模型。

# **实验结果与分析**

## 3.1 实验数据

本次实验使用的数据集来源于GitHub的情感分析任务，包含三个主要文件：

train.tsv.zip：包含评论短语和对应的情感标签，用于模型训练和测试。数据集共有X条样本，其中前80%作为训练集，后20%作为测试集。

test.tsv.zip：原本用于提交预测结果，但在本次实验中未使用。

sampleSubmission.csv：用于提交竞赛结果，在本次实验中未使用。

训练集的短语（Phrase）字段由电影评论文本构成，对应的情感标签（Sentiment）为0到4之间的整数，分别代表不同的情感类别（如非常消极、消极、中性、积极、非常积极）。

## 3.2 实验设置

在本次实验中，我们对模型的设置和训练流程进行了如下设计：

模型结构：使用了带有双向GRU的循环神经网络（RNN）模型。

模型参数：

输入字符数：128（ASCII码字符集）。

隐藏层大小：128。

GRU层数：2。

使用双向GRU。

批次大小（Batch Size）：512。

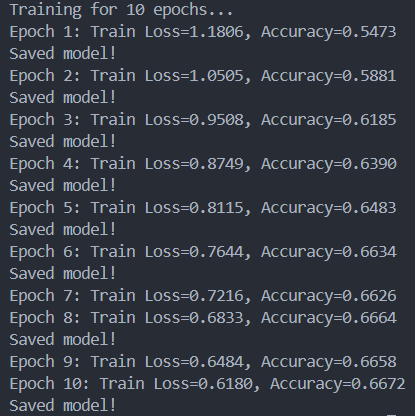
训练轮数（Epochs）：10。

学习率：0.001，采用Adam优化器。

实验过程中，我们将数据按80%和20%比例切分为训练集和测试集。训练集用于模型训练，测试集用于验证模型的泛化能力。在每个epoch结束后，我们保存了在验证集上表现最优的模型。

## 3.3 实验结果与分析

实验训练过程显示，随着训练的进行，模型的损失逐渐降低，准确率逐步提升。具体训练结果如下：



从结果可以看出，模型在前6个epoch中表现逐步提升，验证准确率从54.7%上升至66.7%，随后准确率趋于稳定。最终模型在测试集上的最高准确率达到了66.7%。

# **总结**

本次实验基于GitHub上的电影评论情感分析任务，使用RNN模型中的双向GRU结构进行了情感分类的研究和实现。在实验过程中，我们对数据集进行了合理的划分与预处理，设计并训练了一个包含双向GRU的深度学习模型，通过多轮训练和验证，逐步提高了模型的分类准确率。

实验结果表明，模型在训练集上的准确率随着训练轮数的增加不断提升，最终在测试集上的最高准确率达到了66.7%。双向GRU的引入有效增强了模型对上下文信息的捕捉能力，使得其在复杂的情感分类任务中表现出较好的效果。此外，通过模型保存机制，能够保留每次训练中表现最优的模型，进一步提高了最终结果的可靠性。

虽然实验取得了较好的结果，但仍存在一些优化空间。首先，模型在某些情感类别上的分类效果有待提高，特别是在情感标签较为模糊或文本较短的情况下。其次，未来工作可以尝试引入更复杂的嵌入表示或注意力机制，以进一步提升模型对关键情感信息的捕捉能力。同时，结合更大规模的预训练模型（如BERT或GPT），可能在进一步提高分类准确率方面有所助益。

总的来说，本次实验验证了RNN模型在文本情感分类任务中的有效性，并为后续的模型优化和改进提供了良好的基础与方向。

# **参考文献**

1. RNN介绍：https://zhuanlan.zhihu.com/p/30844905

2. LSTM介绍：https://colah.github.io/posts/2015-08-Understanding-LSTMs/

3. 情感分析任务介绍：

4. 参考论文-自注意力机制(self-attention)："A Structured Self-attentive Sentence Embedding"

5. 参考论文-门控线性单元(GLU): "Language Modeling with Gated Convolutional Networks"

6. 参考论文-多头注意力机制(Multi-head attention): "Attention is all your need"

# **完整代码**

import math

from itertools import chain

import torch

import matplotlib.pyplot as plt

import pandas as pd

from torch.nn.utils.rnn import pack\_padded\_sequence

from torch.utils.data import Dataset, DataLoader, random\_split

import zipfile

class NameDataset(Dataset):

    def \_\_init\_\_(*self*):

        with zipfile.ZipFile('/home/nas/zhangjianhao/train.tsv.zip', 'r') as z:

            with z.open('train.tsv') as f:

*self*.train = pd.read\_csv(f, *sep*='\t')

*self*.phrase = *self*.train['Phrase']

*self*.sentiment = *self*.train['Sentiment']

*self*.len = *self*.train.shape[0]

    def \_\_getitem\_\_(*self*, *index*):

        return *self*.phrase[*index*], *self*.sentiment[*index*]

    def \_\_len\_\_(*self*):

        return *self*.len

*# 数据加载与分割*

dataset = NameDataset()

train\_size = int(0.8 \* len(dataset))

test\_size = len(dataset) - train\_size

train\_set, test\_set = random\_split(dataset, [train\_size, test\_size])

device = torch.device('cuda:0')

NUM\_CHARS = 128

HIDDEN\_SIZE = 128

NUM\_LAYERS = 2

NUM\_EPOCHS = 10

BATCH\_SIZE = 512

train\_loader = DataLoader(train\_set, *batch\_size*=BATCH\_SIZE, *shuffle*=True)

test\_loader = DataLoader(test\_set, *batch\_size*=BATCH\_SIZE, *shuffle*=False)

NUM\_CLASS = len(set(dataset.sentiment))

class RNNClassifier(torch.nn.Module):

    def \_\_init\_\_(*self*, *input\_size*, *hidden\_size*, *output\_size*, *n\_layers*=1, *bidirectional*=True):

        super(RNNClassifier, *self*).\_\_init\_\_()

*self*.hidden\_size = *hidden\_size*

*self*.n\_layers = *n\_layers*

*self*.n\_direction = 2 if *bidirectional* else 1

*self*.embedding = torch.nn.Embedding(*input\_size*, *hidden\_size*)

*self*.gru = torch.nn.GRU(*hidden\_size*, *hidden\_size*, *n\_layers*, *bidirectional*=*bidirectional*)

*self*.fc = torch.nn.Linear(*hidden\_size* \* *self*.n\_direction, *output\_size*)

    def \_init\_hidden(*self*, *batch\_size*):

        hidden = torch.zeros(*self*.n\_layers \* *self*.n\_direction, *batch\_size*, *self*.hidden\_size)

        return hidden.to(device)

    def forward(*self*, *input*, *seq\_lengths*):

*input* = *input*.t()

        batch\_size = *input*.size(1)

        hidden = *self*.\_init\_hidden(batch\_size)

        embedding = *self*.embedding(*input*)

        valid\_idx = *seq\_lengths* > 0

*seq\_lengths* = *seq\_lengths*[valid\_idx]

        embedding = embedding[:, valid\_idx, :]

        gru\_input = pack\_padded\_sequence(embedding, *seq\_lengths*.cpu())

        output, hidden = *self*.gru(gru\_input, hidden[:, valid\_idx, :])

        if *self*.n\_direction == 2:

            hidden\_cat = torch.cat((hidden[-1], hidden[-2]), *dim*=1)

        else:

            hidden\_cat = hidden[-1]

        fc\_output = *self*.fc(hidden\_cat)

        return fc\_output

def phrase2list(*phrase*):

    if isinstance(*phrase*, str):

        arr = [ord(c) for c in *phrase*]

        return arr, len(arr)

    else:

        return [], 0

def make\_tensors(*phrase*, *sentiment*):

    sequences\_and\_lengths = [phrase2list(p) for p in *phrase*]

    phrase\_sequences = [sl[0] for sl in sequences\_and\_lengths]

    seq\_lengths = torch.LongTensor([sl[1] for sl in sequences\_and\_lengths])

*sentiment* = *sentiment*.long()

    seq\_tensor = torch.zeros(len(phrase\_sequences), seq\_lengths.max()).long()

    for idx, (seq, seq\_len) in enumerate(zip(phrase\_sequences, seq\_lengths), 0):

        seq\_tensor[idx, :seq\_len] = torch.LongTensor(seq)

    seq\_lengths, prem\_idx = seq\_lengths.sort(*dim*=0, *descending*=True)

    seq\_tensor = seq\_tensor[prem\_idx]

*sentiment* = *sentiment*[prem\_idx]

    return seq\_tensor.to(device), seq\_lengths.to(device), *sentiment*.to(device)

def train():

    classifier.train()

    total\_loss = 0

    for i, (phrase, sentiment) in enumerate(train\_loader, 1):

        inputs, seq\_lengths, target = make\_tensors(phrase, sentiment)

        output = classifier(inputs, seq\_lengths)

        loss = criterion(output, target)

        optimizer.zero\_grad()

        loss.backward()

        optimizer.step()

        total\_loss += loss.item()

    return total\_loss / len(train\_loader)

def evaluate():

    classifier.eval()

    correct = 0

    total = 0

    with torch.no\_grad():

        for phrase, sentiment in test\_loader:

            inputs, seq\_lengths, target = make\_tensors(phrase, sentiment)

            output = classifier(inputs, seq\_lengths)

            predicted = torch.argmax(output, *dim*=1)

            total += target.size(0)

            correct += (predicted == target).sum().item()

    accuracy = correct / total

    return accuracy

if \_\_name\_\_ == '\_\_main\_\_':

    classifier = RNNClassifier(NUM\_CHARS, HIDDEN\_SIZE, NUM\_CLASS, NUM\_LAYERS).to(device)

    criterion = torch.nn.CrossEntropyLoss()

    optimizer = torch.optim.Adam(classifier.parameters(), *lr*=0.001)

    print(f'Training for {NUM\_EPOCHS} epochs...')

    acc\_list = []

    for epoch in range(1, NUM\_EPOCHS + 1):

        train\_loss = train()

        accuracy = evaluate()

        acc\_list.append(accuracy)

        print(f'Epoch {epoch}: Train Loss={train\_loss:.4f}, Accuracy={accuracy:.4f}')

        if accuracy == max(acc\_list):

            torch.save(classifier, '/home/nas/zhangjianhao/sentimentAnalyst.pkl')

            print('Saved model!')

*# 绘制训练准确率曲线*

    epochs = list(range(1, NUM\_EPOCHS + 1))

    plt.plot(epochs, acc\_list)

    plt.xlabel('Epoch')

    plt.ylabel('Accuracy')

    plt.grid()

    plt.show()