实 验 报 告

论文题目：　基于 Transformer 的文本分类

学　　院：　计算机科学与工程学院

年　　级：　二〇二二

专　　业：　计算机科学与技术

姓　　名：　黄正奇

学　　号：　2201803

指导教师：　肖桐

2023 年　 1 月

摘要

互联网中的文本数据呈现爆炸性增长，其中蕴含着巨大的商业和科研价值，由此文本分类技术得到广泛关注。文本表示和文本特征提取是自然语言的基础工作，直接影响文本分类的性能。

Transformer 是一种使用 attention 来显著提高深度学习自然语言处理任务的性能模型架构。本文通过 Transformer 来研究文本分类，Transformer 的开创性性能的关键是它对 Attention 的使用，故与此同时，我们研究 Transformer 的内部运作方式来剖析 Transformer 在文本分类的应用。

关键字

Encoder；Decoder；Attention；堆栈

Abstract

The text in the Internet presents an explosive growth, which contains huge commercial and scientific research value, so the text classification technology has been widely concerned. Text representation and text feature extraction are the basic work of natural language and directly affect the performance of text classification.

Transformer is a performance model architecture that uses attention to significantly improve deep learning natural language processing tasks. In this paper, we study text classification through Transformer. The key to the pioneering performance of Transformer is its use of Attention. Therefore, at the same time, we study the internal operation of Transformer to analyze the application of Transformer in text classification.

Key words

Encoder；Decoder；Attention；stack

目录

前言

Transformer 是一种使用 attention 来显著提高深度学习自然语言处理翻译模型的性能的架构。Transformer 的开创性性能的关键是它对 Attention 的使用。Transformer 架构通过将输入序列中的每个单词与其他的每一个单词联系起来来使用 self attention。

实验报告

1. 文本分类

由于互联网和自媒体的快速崛起，人人都可以是内容生产者,各种信息档呈爆炸式增长。我们不缺乏信息的来源,但找到需要的信息却愈加困难。为了便于用户浏览和搜索,需将文本进行合理分类。在文本分类中特征选择和特征加权是两个不可或缺的部分,因此,寻找有效的特征选择和特征加权的方法,对文本分类具有重要的理论意义与应用价值。

人工神经网络在大数据的筛选与处理方面起到了极为关键的作用。人工神经网络在计算机视觉、机器翻译、自动驾驶等领域已成功地解决了许多棘手问题,因此人工神经网络也越来越多的被应用到自然语言处理中的文本分类问题上,该方向是目前自然语言处理的一个热点以及难点。人工神经网络不仅可以快速高效的处理海量数据,并能够在一定程度上提高处理数据的精确性。

2 transformer

Transformer 架构擅长于处理固有的顺序的文本数据。它们将一个文本序列作为输入，并生成另一个文本序列作为输出。特别是在机器翻译领域，如图 1 所示，它可以将一个输入的英语句子翻译成西班牙语。

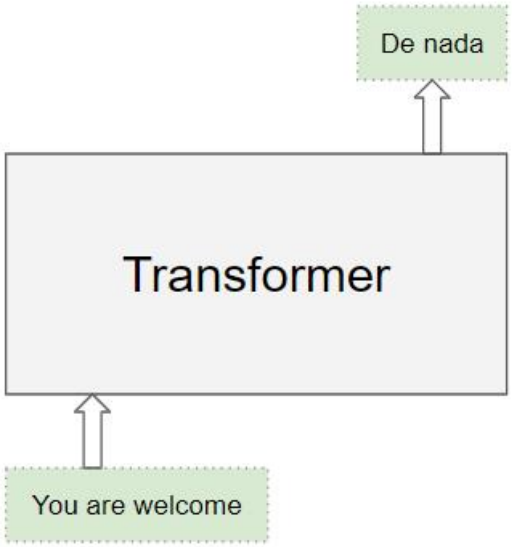

图 1 Transformer 的应用举例

本部分介绍了 Transformer 的内部结构。从图 2 中可以看出，Transformer 可以分为两部分。它包含一堆 Encoder 层和 Decoder 层。Encoder 堆栈和 Decoder 堆栈对于各自的输入都有相应的嵌入层。最后，有一个输出层来生成最终的输出。

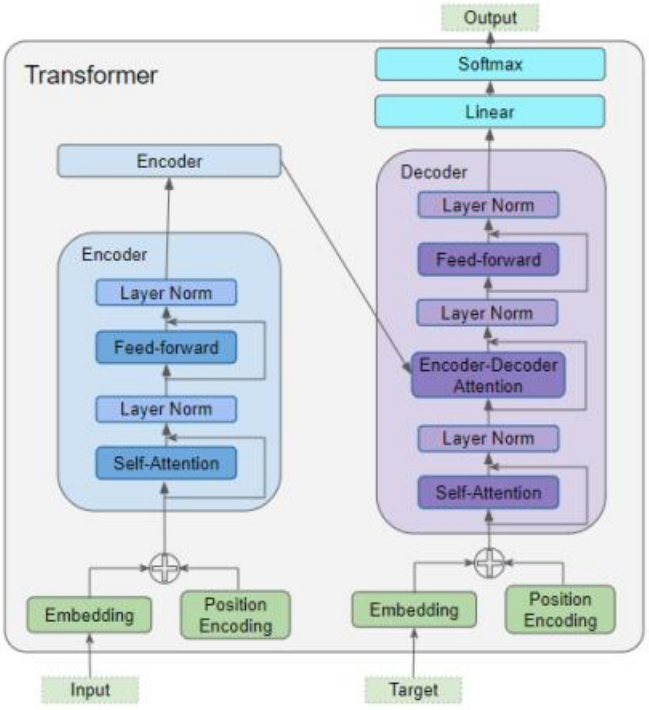

图 2 Transformer 的内部结构

2.1 Encoder

堆栈包含许多 Encoder，如图 3 所示，每个 Encoder 都包含 Multi-Head Attention 和前馈层。

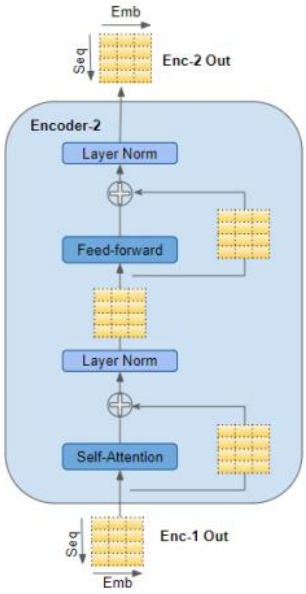

图 3 Encoder 的内部结构

Encoder 将其输入传递到一个 Self-attention 层。Self-attention 输出被传递到一个前馈层，然后将其输出向上发送到下一个 Encoder。每个 Self-attention 和前馈子层周围都有一个残余的跳过连接，然后是一个层归一化。

Encoder 和 Encoder 堆栈分别由几个 encoder 和 decoder 组成，并按顺序连接。堆栈中的第一个 Encoder 接收来自嵌入和位置编码的输入。堆栈中的其他 Encoder 从上一个 Encoder 接收它们的输入。最后一个 Encoder 的输出被输入到 Decoder 堆栈中的每个 Decoder 中。

2.2 Decoder

如图 3 所示，decoder 的结构与 encoder 的非常相似，但有一些不同，如图 4 所示。与 encoder 一样，堆栈中的第一个 decoder 从输出嵌入和位置编码接收其输入。堆栈中的其他 decoder 从上一个 decoder 接收它们的输入。

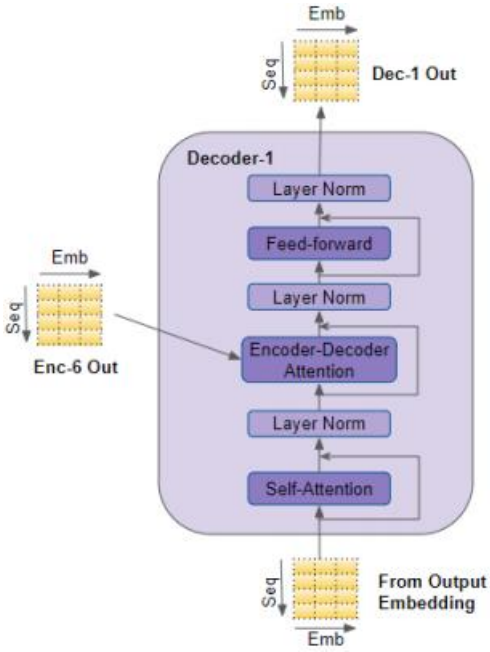

图 4 Decoder 的内部结构

Decoder 将其输入传递到一个 Multi-Head Attention，这种操作方式与 encoder 中的操作方式略有不同，它只允许 Attention 在这个顺序中较早的位置，这是通过掩盖接下来的位置来实现的。

与 encoder 不同，decoder 有第二个多头 Attention 层，称为 encoder-decoder 注意层。encoder-decoder 注意层的工作原理就像 self attention 一样，除了它结合了两个输入源—它下面的 self attention 层以及 encoder 堆栈的输出。

Self-attention 输出被传递到一个前馈层，然后将其输出向上发送到下一个 decoder。每个子层，self attention、encoder-decoder 注意和前馈，周围都有一个残余的跳过连接，然后是层归一化。

2.3 Attention

在处理序列时，attention 显得尤为重要。Attention 使模型能够专注于输入中与该单词密切相关的其他单词。如图 5，在 Transformer 的内部结构中，有三类 Attention。在 Decoder 中，有两类 attention。

在 Encoder 的 self attention 中，Encoder 的输入序列会流向三个参数，查询、键及值，同样在 Decoder 中，Decoder 的输入会流向查询、键及值三个参数。

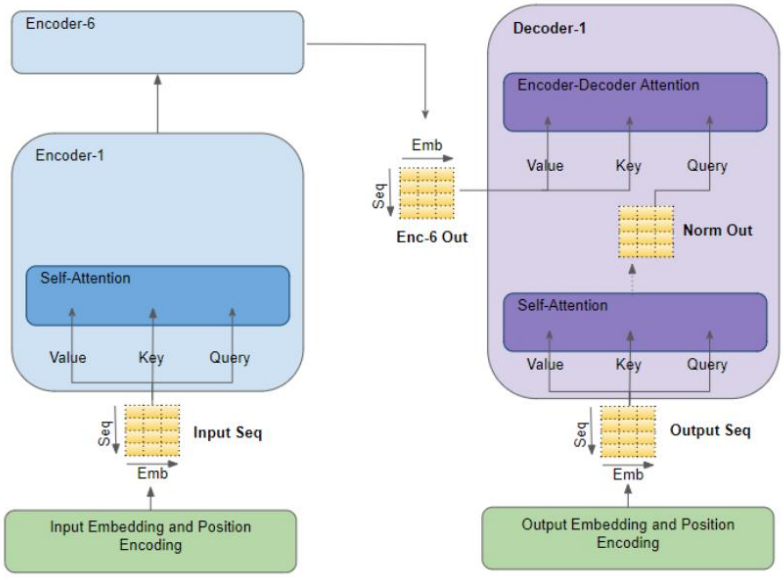

图 5 Transformer 的 Attention

如图 6，在 Decoder 的 Encoder-Decoder 中，来自 Encoder 栈的最终输入会流向键及值域，Self-attention 的输出模块会流向查询的参数，目标序列主要处理的是输入序列。因此，Encoder-Decoder 注意同时得到了目标序列的表征，这种表征来自 Decoder 的 Self-attention，以及输入序列的表征，此表征来自 Encoder 堆栈。因此，它生成每个目标序列单词的 Attention 分数表示，亦能从输入序列中捕获 attention 得分的影响。

当它经过堆栈中的所有 Decoder 时，每个 self attention 和每个 Encoder-Decoder 注意也会在每个单词的表示中添加它们自己的 Attention 分数。

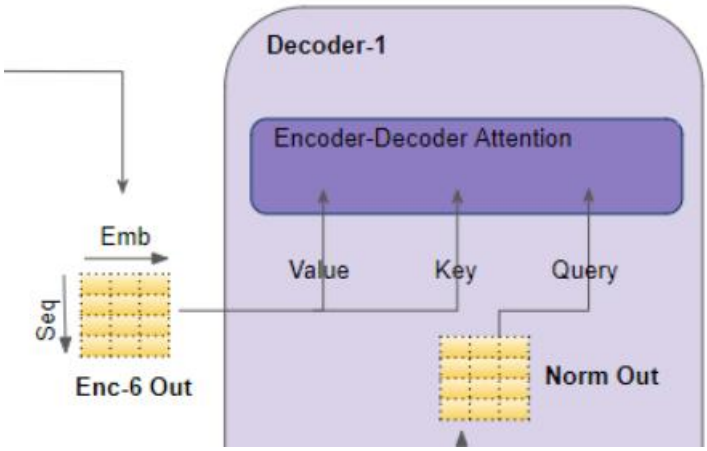

图 6 Encoder-Decoder 的 Attention

2.4 原理

情感分析为文本分类的一个应用，以情感分析为例，在这种情况下，情感分析将任

务的类别分成若干个与情感有关的类，较为典型的情况是分成积极与消极两类。如图 7，情绪分析应用程序将以文本文档作为输入，分类头获取 Transformer 的输出，并生成类标签的预测，如积极或消极的情绪。

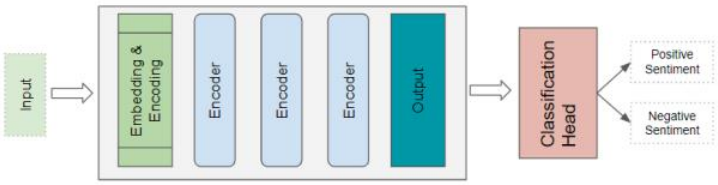

图 7 Transformer 在文本分类中的应用

3 实验

在此次实验中，本人用 Python 作为编程语言，导入版本为 1.8.0 的 torch 来进行监督学习任务。

3.1 准备工作

如图 8，在此次文本分类任务中，我们将类别划分为金融、股市、楼价、教育、科学、社会、政治、运动、游戏及娱乐十类。

```
finance
realty
stocks
education
science
society
politics
sports
game
entertainment
```

图 8 此次文本分类的类别

如图 9，在此次任务中，我们以清华大学的数据集 thucnews 作为数据集，将数据集分成训练集、测试集及验证集来分别进行训练、测试及验证任务。

图 9（a）数据集分布一览

图 9（b）数据集内部

如图 10，我们根据图 2 来搭建神经网络，设置参数，我们得到 Transformer 的内部结构，我们搭建了三个 Encoder 及 Multi-Head attention。

```python
class Model(nn.Module):
    def __init__(self, config):
        super(Model, self).__init__()
        if config.embedding_pretrained is not None:
            self.embedding = nn.Embedding.from_pretrained(config.embedding_pretrained, freeze=False)
        else:
            self.embedding = nn.Embedding(config.n_vocab, config.embed, padding_idx=config.n_vocab - 1)

        self.postion_embedding = Positional_Encoding(config.embed, config.pad_size, config.dropout, config.device)
        self.encoder = Encoder(config.dim_model, config.num_head, config.hidden, config.dropout)
        self.encoders = nn.ModuleList([
            copy.deepcopy(self.encoder)
            # Encoder(config.dim_model, config.num_head, config.hidden, config.dropout)
            for _ in range(config.num_encoder)])

        self.fc1 = nn.Linear(config.pad_size * config.dim_model, config.num_classes)
        # self.fc2 = nn.Linear(config.last_hidden, config.num_classes)
        # self.fc1 = nn.Linear(config.dim_model, config.num_classes)

    def forward(self, x):
        out = self.embedding(x[0])
        out = self.postion_embedding(out)
        for encoder in self.encoders:
            out = encoder(out)
        out = out.view(out.size(0), -1)
        # out = torch.mean(out, 1)
        out = self.fc1(out)
        return out
```

图 10（a）python 搭建 Transformer

```
<bound method Module.parameters of Model(
  (embedding): Embedding(4762, 300)
  (postion_embedding): Positional_Encoding(
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (encoder): Encoder(
    (attention): Multi_Head_Attention(
      (fc_Q): Linear(in_features=300, out_features=300, bias=True)
      (fc_K): Linear(in_features=300, out_features=300, bias=True)
      (fc_V): Linear(in_features=300, out_features=300, bias=True)
      (attention): Scaled_Dot_Product_Attention()
      (fc): Linear(in_features=300, out_features=300, bias=True)
      (dropout): Dropout(p=0.5, inplace=False)
      (layer_norm): LayerNorm((300,), eps=1e-05, elementwise_affine=True)
    )
    (feed_forward): Position_wise_Feed_Forward(
      (fc1): Linear(in_features=300, out_features=1024, bias=True)
      (fc2): Linear(in_features=1024, out_features=300, bias=True)
      (dropout): Dropout(p=0.5, inplace=False)
      (layer_norm): LayerNorm((300,), eps=1e-05, elementwise_affine=True)
    )
  )
  (encoders): ModuleList(
    (0): Encoder(
      (attention): Multi_Head_Attention(
        (fc_Q): Linear(in_features=300, out_features=300, bias=True)
        (fc_K): Linear(in_features=300, out_features=300, bias=True)
        (fc_V): Linear(in_features=300, out_features=300, bias=True)
        (attention): Scaled_Dot_Product_Attention()
        (fc): Linear(in_features=300, out_features=300, bias=True)
        (dropout): Dropout(p=0.5, inplace=False)
        (layer_norm): LayerNorm((300,), eps=1e-05, elementwise_affine=True)
      )
      (feed_forward): Position_wise_Feed_Forward(
        (fc1): Linear(in_features=300, out_features=1024, bias=True)
        (fc2): Linear(in_features=1024, out_features=300, bias=True)
        (dropout): Dropout(p=0.5, inplace=False)
        (layer_norm): LayerNorm((300,), eps=1e-05, elementwise_affine=True)
      )
    )
    (1): Encoder(
      (attention): Multi_Head_Attention(
        (fc_Q): Linear(in_features=300, out_features=300, bias=True)
        (fc_K): Linear(in_features=300, out_features=300, bias=True)
        (fc_V): Linear(in_features=300, out_features=300, bias=True)
        (attention): Scaled_Dot_Product_Attention()
        (fc): Linear(in_features=300, out_features=300, bias=True)
        (dropout): Dropout(p=0.5, inplace=False)
        (layer_norm): LayerNorm((300,), eps=1e-05, elementwise_affine=True)
      )
      (feed_forward): Position_wise_Feed_Forward(
        (fc1): Linear(in_features=300, out_features=1024, bias=True)
        (fc2): Linear(in_features=1024, out_features=300, bias=True)
        (dropout): Dropout(p=0.5, inplace=False)
        (layer_norm): LayerNorm((300,), eps=1e-05, elementwise_affine=True)
      )
    )
  )
  (fc1): Linear(in_features=9600, out_features=10, bias=True)
)>
```

图 10(b) Transformer 的内部结构

3.2 运行结果

如图 11，一般混淆矩阵及召回率、精确率、f1-score 面向的是二分类问题，但此任务我们研究的是多分类问题，我们通过 evaluate()方法来定义多分类问题的性能评估问题。

```python
def evaluate(config, model, data_iter, test=False):
    model.eval()
    loss_total = 0
    predict_all = np.array([], dtype=int)
    labels_all = np.array([], dtype=int)
    with torch.no_grad():
        for texts, labels in data_iter:
            outputs = model(texts)
            loss = F.cross_entropy(outputs, labels)
            loss_total += loss
            labels = labels.data.cpu().numpy()
            predic = torch.max(outputs.data, 1)[1].cpu().numpy()
            labels_all = np.append(labels_all, labels)
            predict_all = np.append(predict_all, predic)

    acc = metrics.accuracy_score(labels_all, predict_all)
    if test:
        report = metrics.classification_report(labels_all, predict_all, target_names=config.class_list, digits=4)
        confusion = metrics.confusion_matrix(labels_all, predict_all)
        return acc, loss_total / len(data_iter), report, confusion
    return acc, loss_total / len(data_iter)
```

图 11（a） 重新定义用于衡量的变量

```python
def test(config, model, test_iter):
    # test
    model.load_state_dict(torch.load(config.save_path))
    model.eval()
    start_time = time.time()
    test_acc, test_loss, test_report, test_confusion = evaluate(config, model, test_iter, test=True)
    msg = 'Test Loss: {0:>5.2},  Test Acc: {1:>6.2%}'
    print(msg.format(test_loss, test_acc))
    print("Precision, Recall and F1-Score...")
    print(test_report)
    print("Confusion Matrix...")
    print(test_confusion)
    time_dif = get_time_dif(start_time)
    print("Time usage:", time_dif)
```

图 11（b） 输出性能评估

如图 12，完成了使用 Transformer 完成文本分类后果，我们得到了一系列的数据。其中正确率为 89.94%。

```
No optimization for a long time, auto-stopping...
Test Loss:  0.33,  Test Acc: 89.94%
Precision, Recall and F1-Score...
               precision    recall  f1-score   support

      finance     0.9208    0.8370    0.8769      1000
       realty     0.9000    0.9180    0.9089      1000
       stocks     0.7958    0.8420    0.8183      1000
    education     0.9484    0.9370    0.9427      1000
      science     0.8717    0.8020    0.8354      1000
      society     0.8870    0.9260    0.9061      1000
     politics     0.8807    0.8860    0.8833      1000
       sports     0.9768    0.9680    0.9724      1000
         game     0.9093    0.9320    0.9205      1000
entertainment     0.9105    0.9460    0.9279      1000

     accuracy                         0.8994     10000
    macro avg     0.9001    0.8994    0.8992     10000
 weighted avg     0.9001    0.8994    0.8992     10000

Confusion Matrix...
[[837  24  97   5  15   8   9   2   1   2]
 [ 15 918  19   1   7  20   5   2   2  11]
 [ 41  27 842   0  35   3  31   1  16   4]
 [  0   5   3 937   6  22  12   1   4  10]
 [  4  11  42   9 802  20  28   3  55  26]
 [  1  15   2  15   5 926  21   0   4  11]
 [  7   9  36   8  15  26 886   2   2   9]
 [  2   3   3   1   3   5   5 968   0  10]
 [  1   3  11   5  23   5   6   4 932  10]
 [  1   5   3   7   9   9   3   8   9 946]]
```

图 12 实验结果

总结

相对于适用于序列数据的循环神经网络，Transformer 有着无可比拟的优越性，它受前面的输出比较少，它的工作原理须结合 Encoder、Decoder，其中它们的运作与 Attention 密不可分，从正确率的角度而言，Attention 是 Transformer 在处理文本序列数据胜于循环神经网络的原因。在此次实验可以看出，Transformer 的准确率较高。但我的工作仍然有欠缺之处，因为此次实验是多分类问题，故本人未能绘制出受试者特征工作曲线来衡量分类的性能。

意见及致谢

通过与肖桐老师的接触，本人发现老师是一个在人工智能领域造诣很深的学者，更是本人的偶像，但有时觉得老师在课堂上的教授之时语速偏快，可能是本人能力不足的原因，我经常出现难以及时理解老师教授的知识的内容，肖桐老师除了个人造诣深之外，对学生亦很温柔，是一个不可多得的好老师，希望老师在本人未来的学习道路上给予更大的指点及帮助！