**《深度学习》实验报告二**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **大三计算机科学与技术（弘深）计算机拔尖01班** | | **姓名** | | **朱云哲** | **学号** | **20224700** |
| **实验题目** | **基于Transformer的英德语言翻译实验** | | | | | | | |
| **实验时间** | **2025.4.17** | | **实验地点** | | **DS1402** | | | |
| **学年学期** | **2024-2025（2）** | | **实验性质** | | **□验证性 ■设计性 □综合性** | | | |
| 一、实验目的  1．熟悉mindspore实验环境，训练网络的过程。  2. 理解和掌握transformer的网络的基本构成。 | | | | | | | | |
| 二、实验项目内容  1. 基于华为云ModelArts实现transformer的网络搭建  2. 使用Multi30k数据集进行训练，实现英语翻译德语的功能 | | | | | | | | |
| **三、实验步骤**  **3. 1 基于华为云ModelArts实现transformer的网络搭建**  **（一）Transformer网络介绍**  Transformer是一种基于注意力机制的深度学习框架，在2017年由Vaswani等人提出。它主要用于处理自然语言处理（NLP）任务，如机器翻译、文本生成等。Transformer的核心思想是完全摒弃了传统的循环神经网络（RNN）结构，转而通过自注意力机制（Self-Attention Mechanism）来捕捉序列数据中的长距离依赖关系，从而极大地提高了模型的训练效率和性能。Transformer网络主要由编码器（Encoder）和解码器（Decoder）两部分构成。  **（1）编码器（Encoder）**  编码器由多个相同的层堆叠而成，每一层包含两个主要模块：**多头自注意力机制**和**前馈神经网络**。  **·多头自注意力机制**  多头自注意力机制是Transformer中的核心部分，它允许模型在不同的表示子空间中并行地学习信息。其工作原理如下：   1. **输入嵌入（Input Embedding）**：输入序列（如单词或字符）首先被映射为嵌入向量，通常还会加上位置编码以保留序列的顺序信息。 2. **线性变换**：将输入嵌入向量分别通过三个不同的线性变换，得到查询向量、键向量和值向量，即Q，K，V向量。 3. **缩放点积注意力（Scaled Dot-Product Attention）**：对于每个查询向量，计算它与所有键向量的点积，得到一个注意力分数矩阵。然后将该分数矩阵除以一个缩放因子（通常是键向量维度的平方根），以防止梯度消失。接着通过Softmax函数对注意力分数进行归一化，得到每个位置的注意力权重。最后，将注意力权重与对应的值向量相乘并求和，得到加权的值向量。 4. **多头注意力（Multi-Head Attention）**：为了捕捉不同子空间中的信息，Transformer将查询、键和值向量分别分成多个头（Head），每个头独立地执行上述缩放点积注意力操作。然后将所有头的输出拼接在一起，并通过一个线性变换进行整合，得到最终的多头自注意力输出。   **·前馈神经网络**  前馈神经网络是一个简单的全连接神经网络，它对每个位置的向量独立地进行变换。具体来说，它包含两个线性变换，中间通过一个非线性激活函数（如ReLU）进行激活。这个网络的作用是对多头自注意力的输出进行进一步的非线性变换，以提取更高级的特征。  **·残差连接与层归一化**  在每个模块（多头自注意力和前馈神经网络）的输出上，Transformer 都采用了**残差连接和层归一化**的技术。残差连接是指将模块的输入直接加到模块的输出上，这样可以缓解深层网络训练中的梯度消失问题。层归一化则是对每个样本的特征进行归一化，以稳定训练过程并加速收敛。  **（2）解码器（Decoder）**  解码器的结构与编码器类似，也由多个相同的层堆叠而成。每一层包含三个主要模块：掩码多头自注意力机制、编码器-解码器注意力机制和前馈神经网络。  **·掩码多头自注意力机制**  掩码多头自注意力机制的作用是防止解码器在生成序列时看到未来的信息。具体来说，在计算自注意力时，会对注意力分数矩阵施加一个掩码，将当前位置之后的位置的注意力分数设置为负无穷大（或一个非常小的值），这样在Softmax归一化后，这些位置的注意力权重将趋近于零。其余部分与编码器中的多头自注意力机制相同。  **·编码器-解码器注意力机制**  编码器-解码器注意力机制允许解码器在生成目标序列时参考编码器的输出。具体来说，解码器的查询向量与编码器的键向量和值向量进行注意力计算，从而获取编码器中与当前解码位置相关的信息。这使得解码器能够更好地理解输入序列的上下文信息，从而生成更准确的目标序列。  **·前馈神经网络**  解码器中的前馈神经网络与编码器中的前馈神经网络结构相同，作用也是对输入向量进行非线性变换，以提取更高级的特征。  解码器的输出通常会通过一个线性变换和Softmax函数，得到目标词汇表中每个单词的概率分布，从而生成目标序列中的下一个单词。  **（二）华为云ModelArts使用介绍**  华为云ModelArts是一个面向开发者的一站式AI开发平台，提供从数据预处理、模型训练到模型部署的全流程支持。它支持通过JupyterLab工具在线开发基于MindSpore等引擎的AI模型，并提供预置镜像以简化开发环境的搭建。  **环境搭建步骤如下：**  **·创建ModelArts开发环境**   1. **进入华为云ModelArts控制台：**   访问华为云ModelArts主页，点击“管理控制台”进入管理页面。在控制台区域选择“西南-贵阳一”，在左侧菜单栏中选择“开发空间”的“Notebook”。   1. **创建Notebook训练作业：**   点击“创建Notebook”按钮，根据文档设置环境参数。   1. **启动Notebook进入开发环境：**   当Notebook状态显示为“运行中”时，点击右侧操作中的“打开”，进入在线编程页面。在此页面可以创建或编辑MindSpore的项目，文件存储在/home/ma-user/work目录下。  **·关闭/删除环境**   1. **关闭环境：**   点击页面右上角的“终止按钮”关闭环境，或者在ModelArts控制台的操作栏选择“停止”操作。   1. **删除环境**：   关闭环境后，点击控制台操作栏的“更多”，选择“删除”操作。注意：云硬盘从创建成功起至删除前，每GB按照规定费用收费，因此建议不再使用时及时删除。  **（三）搭建Transformer网络**  **（1）依赖包安装**  在开始实验之前，需要安装必要的依赖包，包括MindSpore和nltk库。   |  | | --- | | %env no\_proxy='a.test.com,127.0.0.1,2.2.2.2'  !wget https://ms-release.obs.cn-north-4.myhuaweicloud.com/2.2.13/MindSpore/unified/aarch64/mindspore-2.2.13-cp39-cp39-linux\_aarch64.whl  !pip install mindspore-2.2.13-cp39-cp39-linux\_aarch64.whl -i <https://pypi.tuna.tsinghua.edu.cn/simple>  -------------------------------------------------------------------------------------------------------------- !pip install nltk -i https://pypi.tuna.tsinghua.edu.cn/simple |   **（2）数据预处理**  数据处理是机器翻译任务中的关键步骤，主要包括数据下载、分词、构建词典和创建数据迭代器。  **·数据集下载**  使用Multi30K数据集，包含德语和英语的文本描述。   |  | | --- | | from download import download  import re  url = "https://modelscope.cn/api/v1/datasets/SelinaRR/Multi30K/repo?Revision=master&FilePath=Multi30K.zip"  download(url, './', kind='zip', replace=True) |   **·数据读取与分词**  将句子拆解为单词级词元，并统一转换为小写。   |  | | --- | | class Multi30K:  def \_\_init\_\_(self, path):  self.data = self.\_load(path)  def \_load(self, path):  def tokenize(text):  text = text.rstrip()  return [tok.lower() for tok in re.findall(r'\w+|[^\w\s]', text)]  def read\_data(data\_file\_path):  with open(data\_file\_path, 'r', encoding='utf-8') as data\_file:  data = data\_file.readlines()[:-1]  return [tokenize(i) for i in data]  members = {i.split('.')[-1]: path + i for i in os.listdir(path)}  ret = [read\_data(members['de']), read\_data(members['en'])]  return list(zip(\*ret)) |   **·构建词典**  将词元映射到数字索引，并添加特殊词元（如<unk>、<pad>、<bos>、<eos>）。   |  | | --- | | class Vocab:  special\_tokens = ['<unk>', '<pad>', '<bos>', '<eos>']  def \_\_init\_\_(self, word\_count\_dict, min\_freq=1):  self.word2idx = {}  for idx, tok in enumerate(self.special\_tokens):  self.word2idx[tok] = idx  filted\_dict = {w: c for w, c in word\_count\_dict.items() if c >= min\_freq}  for w, \_ in filted\_dict.items():  self.word2idx[w] = len(self.word2idx)  self.idx2word = {idx: word for word, idx in self.word2idx.items()}  self.bos\_idx = self.word2idx['<bos>']  self.eos\_idx = self.word2idx['<eos>']  self.pad\_idx = self.word2idx['<pad>']  self.unk\_idx = self.word2idx['<unk>'] |   Vocab类用于构建词典，支持词元与索引之间的转换。这里需要重点注意de\_vocab与en\_vocab，完成英译德功能需要理清两个数据的训练逻辑关系。  **·数据迭代器**  创建数据迭代器，对文本序列进行批处理，统一序列长度，并添加特殊词元。   |  | | --- | | class Iterator:  def \_\_init\_\_(self, dataset, de\_vocab, en\_vocab, batch\_size, max\_len=32, drop\_reminder=False):  self.dataset = dataset  self.de\_vocab = de\_vocab  self.en\_vocab = en\_vocab  self.batch\_size = batch\_size  self.max\_len = max\_len  self.drop\_reminder = drop\_reminder  length = len(self.dataset) // batch\_size  self.len = length if drop\_reminder else length + 1 |   **（3）模型结构**  Transformer模型由编码器（Encoder）和解码器（Decoder）组成：  **·位置编码**  为词嵌入添加位置信息，以便模型能够捕捉序列中的位置关系。   |  | | --- | | class PositionalEncoding(nn.Cell):  def \_\_init\_\_(self, d\_model, dropout\_p=0.1, max\_len=100):  super().\_\_init\_\_()  self.dropout = nn.Dropout(1 - dropout\_p)  self.pe = ops.Zeros()((max\_len, d\_model), mstype.float32)  pos = mnp.arange(0, max\_len, dtype=mstype.float32).view((-1, 1))  angle = ops.pow(10000.0, mnp.arange(0, d\_model, 2, dtype=mstype.float32)/d\_model)  self.pe[:, 0::2] = ops.sin(pos/angle)  self.pe[:, 1::2] = ops.cos(pos/angle)  def construct(self, x):  batch\_size = x.shape[0]  pe = self.pe.expand\_dims(0)  pe = ops.broadcast\_to(pe, (batch\_size, -1, -1))  x = x + pe[:, :x.shape[1], :]  return self.dropout(x) |   **·注意力机制**  实现缩放点积注意力（Scaled Dot-Product Attention）。   |  | | --- | | class ScaledDotProductAttention(nn.Cell):  def \_\_init\_\_(self, dropout\_p=0.):  super().\_\_init\_\_()  self.softmax = nn.Softmax()  self.dropout = nn.Dropout(1-dropout\_p)  self.sqrt = ops.Sqrt()  def construct(self, query, key, value, attn\_mask=None):  embed\_size = query.shape[-1]  scaling\_factor = self.sqrt(Tensor(embed\_size, mstype.float16))  attn = ops.matmul(query, key.swapaxes(-2, -1) / scaling\_factor)  if attn\_mask is not None:  attn = attn.masked\_fill(attn\_mask, -1e9)  attn = self.softmax(attn)  attn = self.dropout(attn)  output = ops.matmul(attn, value)  return (output, attn) |   **·多头注意力**  多头注意力机制允许模型在不同的表示子空间中并行地学习信息。   |  | | --- | | class MultiHeadAttention(nn.Cell):  def \_\_init\_\_(self, d\_model, d\_k, n\_heads, dropout\_p=0.):  super().\_\_init\_\_()  self.n\_heads = n\_heads  self.d\_k = d\_k  self.W\_Q = nn.Dense(d\_model, d\_k \* n\_heads)  self.W\_K = nn.Dense(d\_model, d\_k \* n\_heads)  self.W\_V = nn.Dense(d\_model, d\_k \* n\_heads)  self.W\_O = nn.Dense(n\_heads \* d\_k, d\_model)  self.attention = ScaledDotProductAttention(dropout\_p=dropout\_p) |   **·前馈神经网络**  对每个位置的词嵌入进行非线性变换。   |  | | --- | | class PoswiseFeedForward(nn.Cell):  def \_\_init\_\_(self, d\_ff, d\_model, dropout\_p=0.):  super().\_\_init\_\_()  self.linear1 = nn.Dense(d\_model, d\_ff)  self.linear2 = nn.Dense(d\_ff, d\_model)  self.dropout = nn.Dropout(1-dropout\_p)  self.relu = nn.ReLU()  def construct(self, x):  x = self.linear1(x)  x = self.relu(x)  x = self.dropout(x)  output = self.linear2(x)  return output |   **·编码器层**  编码器由多个编码器层组成，每层包括多头自注意力机制和前馈神经网络。   |  | | --- | | class EncoderLayer(nn.Cell):  def \_\_init\_\_(self, d\_model, n\_heads, d\_ff, dropout\_p=0.):  super().\_\_init\_\_()  d\_k = d\_model // n\_heads  if d\_k \* n\_heads != d\_model:  raise ValueError(f"The `d\_model` {d\_model} can not be divisible by `num\_heads` {n\_heads}.")  self.enc\_self\_attn = MultiHeadAttention(d\_model, d\_k, n\_heads, dropout\_p)  self.pos\_ffn = PoswiseFeedForward(d\_ff, d\_model, dropout\_p)  self.add\_norm1 = AddNorm(d\_model, dropout\_p)  self.add\_norm2 = AddNorm(d\_model, dropout\_p) |   **·解码器层**  解码器由多个解码器层组成，每层包括掩码多头自注意力机制、编码器-解码器注意力机制和前馈神经网络。   |  | | --- | | class DecoderLayer(nn.Cell):  def \_\_init\_\_(self, d\_model, n\_heads, d\_ff, dropout\_p=0.):  super().\_\_init\_\_()  d\_k = d\_model // n\_heads  if d\_k \* n\_heads != d\_model:  raise ValueError(f"The `d\_model` {d\_model} can not be divisible by `num\_heads` {n\_heads}.")  self.dec\_self\_attn = MultiHeadAttention(d\_model, d\_k, n\_heads, dropout\_p)  self.dec\_enc\_attn = MultiHeadAttention(d\_model, d\_k, n\_heads, dropout\_p)  self.pos\_ffn = PoswiseFeedForward(d\_ff, d\_model, dropout\_p)  self.add\_norm1 = AddNorm(d\_model, dropout\_p)  self.add\_norm2 = AddNorm(d\_model, dropout\_p)  self.add\_norm3 = AddNorm(d\_model, dropout\_p) |   **·编码器和解码器**  编码器和解码器分别由多个编码器层和解码器层堆叠而成。   |  | | --- | | class Encoder(nn.Cell):  def \_\_init\_\_(self, src\_vocab\_size, d\_model, n\_heads, d\_ff, n\_layers, dropout\_p=0.):  super().\_\_init\_\_()  self.src\_emb = nn.Embedding(src\_vocab\_size, d\_model)  self.pos\_emb = PositionalEncoding(d\_model, dropout\_p)  self.layers = nn.CellList([EncoderLayer(d\_model, n\_heads, d\_ff, dropout\_p)] \* n\_layers)  class Decoder(nn.Cell):  def \_\_init\_\_(self, trg\_vocab\_size, d\_model, n\_heads, d\_ff, n\_layers, dropout\_p=0.):  super().\_\_init\_\_()  self.trg\_emb = nn.Embedding(trg\_vocab\_size, d\_model)  self.pos\_emb = PositionalEncoding(d\_model, dropout\_p)  self.layers = nn.CellList([DecoderLayer(d\_model, n\_heads, d\_ff) for \_ in range(n\_layers)]) |   **·Transformer模型**  将编码器和解码器组合成完整的Transformer模型。   |  | | --- | | class Transformer(nn.Cell):  def \_\_init\_\_(self, encoder, decoder):  super().\_\_init\_\_()  self.encoder = encoder  self.decoder = decoder |   **3.2 使用Multi30k数据集进行训练，实现英语翻译德语的功能**  由于我们需要实现英语翻译德语的功能，但是在原文档给出的是德语翻译英语的功能，因此我们需要交换目标文与源文，实现英语翻译德语的训练。  在原代码中，我们规定了**源文（德语）词汇表存储在de\_vocab**中，**目标文（英语）词汇表存储在en\_vocab**中，因此在Multipli30K类中，我们需要交换生成的两个词汇表：   |  | | --- | | *def* read\_data(data\_file\_path):  *with open*(data\_file\_path, 'r', encoding='utf-8') *as* data\_file:  data = data\_file.readlines()[:-1]  *return* [tokenize(i) *for* i *in* data]  members = {i.split('.')[-1]: path + i *for* i *in* os.listdir(path)} ret = [read\_data(members['en']), read\_data(members['de'])] *return list*(*zip*(\*ret)) |   在read\_data函数中，我们交换了ret = [read\_data(members['en']), read\_data(members['de'])]中的en和de位置，这样de\_vocab存储的就是英语源文，en-vocab存储的就是德语目标文。  随后，修改训练函数：   |  | | --- | | src\_vocab\_size = *len*(de\_vocab) trg\_vocab\_size = *len*(en\_vocab) src\_pad\_idx = de\_vocab.pad\_idx trg\_pad\_idx = en\_vocab.pad\_idx |   交换de\_vocab，en\_vocab位置，且后续所有de\_vocab与en\_vocab的位置对调：   |  | | --- | | *def* inference(sentence, max\_len=32):  *"""模型推理：输入一个德语句子，输出翻译后的英文句子  enc\_inputs: [batch\_size(1), src\_len]  """* new\_model.set\_train(*False*)  *if isinstance*(sentence, *str*):  tokens = [tok.lower() *for* tok *in* re.findall(r'\w+|[^\w\s]', sentence.rstrip())]  *else*:  tokens = [token.lower() *for* token *in* sentence]  *if len*(tokens) > max\_len - 2:  src\_len = max\_len  tokens = ['<bos>'] + tokens[:max\_len - 2] + ['<eos>']  *else*:  src\_len = *len*(tokens) + 2  tokens = ['<bos>'] + tokens + ['<eos>'] + ['<pad>'] \* (max\_len - src\_len)  indexes = de\_vocab.encode(tokens) *# 使用英语字典，输入* enc\_inputs = Tensor(indexes, mstype.float32).expand\_dims(0)  enc\_outputs, \_ = new\_model.encoder(enc\_inputs, src\_pad\_idx)  dec\_inputs = Tensor([[en\_vocab.bos\_idx]], mstype.float32) *# 使用德语字典，输出* max\_len = enc\_inputs.shape[1]  *for* \_ *in range*(max\_len):  dec\_outputs, \_, \_ = new\_model.decoder(dec\_inputs, enc\_inputs, enc\_outputs, src\_pad\_idx, trg\_pad\_idx)  dec\_logits = dec\_outputs.view((-1, dec\_outputs.shape[-1]))  dec\_logits = dec\_logits[-1, :]  pred = dec\_logits.argmax(axis=0).expand\_dims(0).expand\_dims(0)  pred = pred.astype(mstype.float32)  dec\_inputs = ops.concat((dec\_inputs, pred), axis=1)  *if int*(pred.asnumpy()[0]) == en\_vocab.eos\_idx:  *break* trg\_indexes = [*int*(i) *for* i *in* dec\_inputs.view(-1).asnumpy()]  eos\_idx = trg\_indexes.index(en\_vocab.eos\_idx) *if* en\_vocab.eos\_idx *in* trg\_indexes *else* -1  trg\_tokens = en\_vocab.decode(trg\_indexes[1:eos\_idx])  *return* trg\_tokens |   至此，我们完成了英语翻译德语的模型搭建，直接训练即可。 | | | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程、实验总结与体会  **4.1 模型测试结果**  **（1）英语翻译德语**  以下为模型训练结果。图1为示例推理的结果，图2为模型的BLUE得分。可以看到，在示例推理中，我们的模型完成了德语的翻译功能，但是准确率不高；在BLUE得分中，我们的模型得分仅有29.95，较低。    图1. 英语翻译德语的推理示例    图2. 英语翻译德语的模型BLUE得分  **（2）模型评估**  总体来看，模型在翻译德语到英语的任务上有一定的效果，但仍存在一些准确性和语法结构上的问题。BLEU分数为29.95分，表明翻译质量还有较大的提升空间。 | | | | | | | | |

实验报告填写说明：

1、第一、二部分由老师提供；

2、第三部分根据实验步骤填写主要源程序，源程序要符合程序编写风格（缩进、注释等）；

3、第四部分主要填写程序结果（截图）、解决问题的方法、总结和体会等；

4、报告规范：包含报告页眉、报告的排版、内容是否填写，命名是否规范等。

5、源程序和实验报告命名：学号姓名序号.扩展名，例如学号20181234的张三同学，**他的第一次实验命名为：20181234张三1.py和20181234张三1.docx**

6、向蓝墨云提交两个文件，一个实验报告，一个程序文件（代码打包）