深度学习与自然语言处理第四次作业报告

吴宣余

ZY2303121

wxy7334@buaa.edu.cn

一、问题描述

利用给定语料库，用 Seq2Seq 与 Transformer 两种不同的模型来实现文本生成的任务（给定开头后生成武侠小说的片段或者章节），并对比与讨论两种方法的优缺点。

二、Seq2Seq 模型

1. RNN

RNN 循环神经网络(Recurrent Neural Network)是一类用于处理序列数据的神经网络。RNN 基本的模型如下图所示，每个神经元接受的输入包括：前一个神经元的隐藏层状态 h(用于记忆)和当前的输入 x(当前信息)。神经元得到输入之后，会计算出新的隐藏状态 h 和输出 y，然后再传递到下一个神经元。因为隐藏状态 h 的存在，使得 RNN 具有一定的记忆功能。

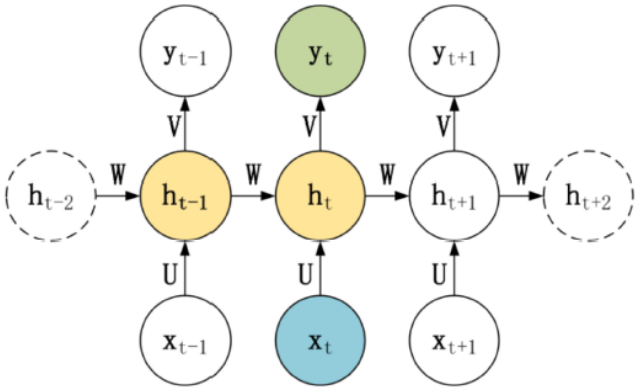

图 1 Rnn 网络结构

2. Seq2Seq 模型

普通 RNN 结构对其输入和输出个数都有一定的限制，但实际中很多任务的序列的长度是不固定的，例如机器翻译中，源语言、目标语言的句子长度不一样；对话系统中，问句和答案的句子长度不一样。

Seq2Seq 是一种重要的 RNN 模型，也称为 Encoder-Decoder 模型，可以理解为一种 N×M 的模型。模型包含两个部分：Encoder 用于编码序列的信息，将任

意长度的序列信息编码到一个向量 c 里。而 Decoder 是解码器，解码器得到上下文信息向量 c 之后可以将信息解码，并输出为序列。

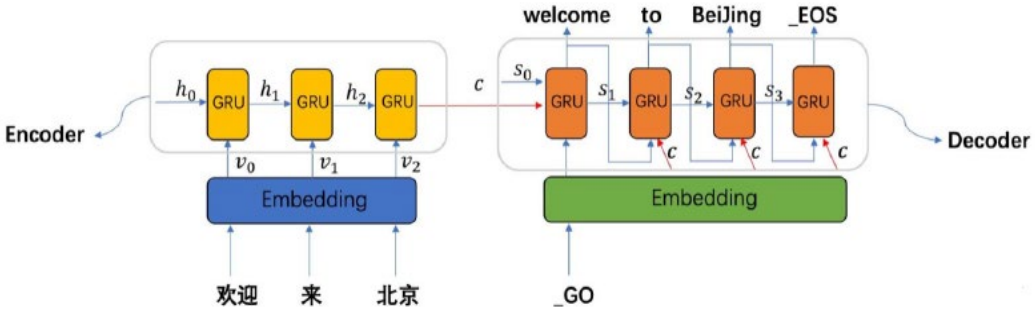

图 2 Seq2Seq 模型结构

三、Transformer 模型

Transformer 模型采用了全新的方式来解决 Seq2Seq 问题(处理长度较长的序列时会出现记忆不足以及训练难度较大)，不同于以往使用的 encoder-decoder 模型，Transformer 摒弃了顺序处理的方式，而是以并行化的方式处理数据，从而实现更大规模的并行计算和更快速的训练。这得益于 Transformer 架构中的自注意力机制，它使得模型能够同时考虑输入序列中的所有位置，而无需按顺序逐步处理。自注意力机制允许模型根据输入序列中的不同位置之间的关系，对每个位置进行加权处理，从而捕捉全局上下文信息。Transformer 模型架构如下图所示：

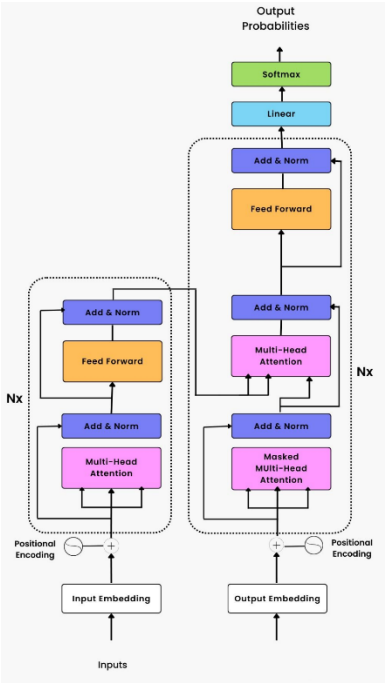

图 3 Transformer 模型结构

四、实验

1. 数据预处理

与前几次作业类似，实验前需对实验数据进行预处理。为了简化训练的步骤和缩短训练时间，本次实验仅选取了《天龙八部》这本小说作为训练的数据集。从语料库中以ANSI编码格式读取文件内容后，删除语料库内的所有非中文字符，以及和小说内容无关的广告片段，得到字符串形式的语料库，并将语料库与汉字3500字符库进行比对操作，从而对小说段落进行索引化，便于提高训练效率。

2. 训练Seq2Seq模型

定义 Seq2Seq 模型的参数：模型共两层，Rnn 的输入大小为 128，隐藏层大小为 256，embedding 大小与输入大小一致，learning-rate 为 0.001。采用 gpu 进行训练，训练 epoch 为 50，batchsize 大小为 512，每次训练的字符数为 50。在此参数下，模型训练时间大约 0.5 个小时。

```python
class Model(nn.Module):
    def __init__(self, dataset):
        super(Model, self).__init__()
        self.input_size = 128
        self.hidden_size = 256
        self.embedding_dim = self.input_size
        self.num_layers = 2

        n_vocab = len(dataset.uniq_words)
        self.embedding = nn.Embedding(
            num_embeddings=n_vocab,
            embedding_dim=self.embedding_dim,)
        self.rnn = nn.RNN(
            input_size=self.input_size,
            hidden_size=self.hidden_size,
            num_layers=self.num_layers,)
        # self.rnn.cpu()
        self.rnn.cuda()
        # self.fc = nn.Linear(self.hidden_size, n_vocab).cpu()
        self.fc = nn.Linear(self.hidden_size, n_vocab).cuda()

    def forward(self, x, prev_state):
        embed = self.embedding(x).cuda()
        # embed = self.embedding(x).cpu()

        output, state = self.rnn(embed, prev_state)
        logits = self.fc(output)

        return logits, state

    def init_state(self, sequence_length):
        # return torch.zeros(self.num_layers, sequence_length, self.hidden_size).cpu()
        return torch.zeros(self.num_layers, sequence_length, self.hidden_size).cuda()
```

图 4 Seq2Seq 模型

3. 训练 Transformer 模型

定义 Transformer 模型的参数：编码器和解码器的嵌入层（词向量层）维度均设置为 256，而各自的隐藏层维度均设置为 512；这种配置旨在捕捉文本数据

的复杂依赖关系，同时保持模型的处理效率。训练 epoch 为 100，batchsize 大小为 2，learning-rate 为 0.001。在此参数下，模型训练时间大约 10 分钟。

```python
class TransformerModel(nn.Module):
    def __init__(self, vocab_size, embed_size, num_heads, num_layers, hidden_dim, dropout=0.1):
        super(TransformerModel, self).__init__()
        self.embed_size = embed_size
        self.dropout_layer = nn.Dropout(p=dropout)  # 正确初始化dropout层
        self.encoder = nn.Embedding(vocab_size, embed_size)
        self.pos_encoder = PositionalEncoding(embed_size, dropout)
        self.transformer = nn.Transformer(d_model=embed_size, nhead=num_heads, num_encoder_layers=num_layers, num_decoder_layers=num_layers
                                          , dim_feedforward=hidden_dim, dropout=dropout, batch_first=True)
        self.decoder = nn.Linear(embed_size, vocab_size)


    def generate_square_subsequent_mask(self, sz):
        mask = (torch.triu(torch.ones(sz, sz)) == 1).transpose(0, 1)
        mask = mask.float().masked_fill(mask == 0, float('-inf')).masked_fill(mask == 1, float(0.0))
        return mask

    def forward(self, src, tgt):
        src_mask = self.generate_square_subsequent_mask(src.size(1)).to(src.device)
        tgt_mask = self.generate_square_subsequent_mask(tgt.size(1)).to(tgt.device)
        # 在forward方法内
        src = self.encoder(src) * math.sqrt(self.embed_size)
        src = self.dropout_layer(src)  # 使用正确的dropout实例
        src = self.pos_encoder(src)
        tgt = self.encoder(tgt) * math.sqrt(self.embed_size)
        tgt = self.pos_encoder(tgt)
        output = self.transformer(src, tgt, src_mask, tgt_mask)
        output = self.decoder(output)
        return output
```

图 5 Transformer 模型

4. 模型预测

上述模型训练完成后，分别将模型的权重保存，用于对测试数据进行预测。

五、实验结果与分析

1. Seq2Seq模型文本生成结果

输入：包三先生笑道：“司马卫的儿子徒弟，都是这么一批脓包货色，除了暗箭伤人，什么都不会。”

输出：西首锦凳上所坐的则是云派人士入剑，住在梁上，双手抓的都是蛇。那少女约莫十六七岁年纪，一身青衫，笑宪如花，手中握着十来条尺许长小蛇。这些小蛇或青或花，头呈三角，均是毒蛇。但从于“久仰”两字也懒得说，只拱了拱手，便肃入宾座。不料这年轻人不知左师狠，太座段誉胸口，喝道：“你到底是真摔，不知这少年弟来教有两条尺师，可是要你手下无弱剑。”

2. Transformer模型文本生成结果

输入：我不出去！我不出去！”她刚才还在大叫“我要出去”，可是一会儿便又大叫“我不出去”

输出：的，不却，她？“她神，，不，对“面只是可道不我这这。我不，，，，枝便，和自道誉。

3. 结果分析

由于 Seq2Seq 模型最终训练的结果较好，loss 值最终能够收敛到 0.1 左右；而 Transformer 模型最终训练的结果较差，loss 值最终仅仅能够收敛到 5.6 左右，此时无论 epoch 增加到多少，loss 值也几乎只会在 5.6 左右波动。

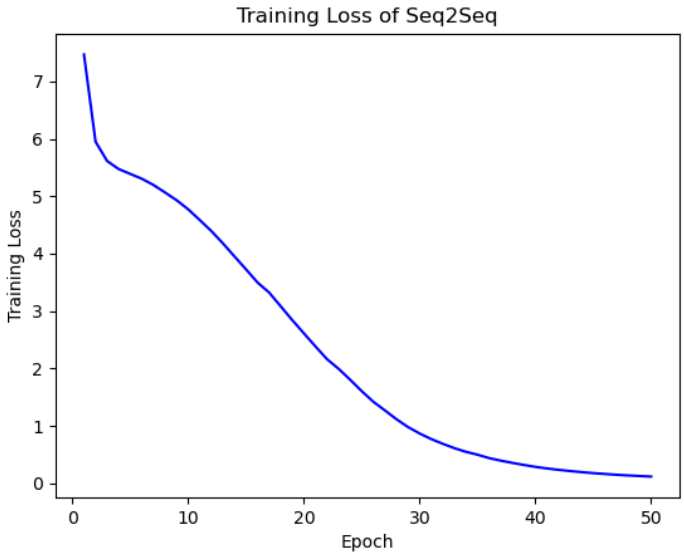

图 6 Seq2Seq 模型训练 loss

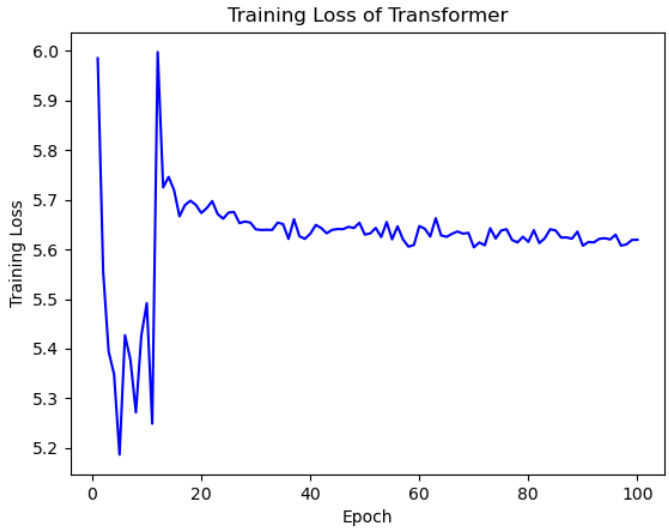

图 7 Transformer 模型训练 loss

因此对比两种模型生成的文本，可以看到 Seq2Seq 模型生成的文本自身内部逻辑性较强，内容基本通顺，与输入文本有一些关联，也能够看出文本创作的武侠风格，但总体关联性还不够，可读性也一般，有待改进；而 Transformer 模型则基本无法生成有效的文本，完全没有逻辑和可读性。后续还需要对 Transformer 模型的超参数进行一些调整，保证 loss 值最终能够收敛到 0.5 以下。

对比两种模型特点，可以得到：Transformer 模型具有更好地处理长距离依赖性的能力，由于自注意力机制的引入，使得模型能够直接关注输入序列中各个位置的信息，无需通过逐步处理，这使得 Transformer 在生成长文本时表现更好。但与之相对应的需要更高的算力和计算资源；而在一些较短的序列任务中，因为不会受到自注意力机制的计算量增加的影响 Seq2Seq 模型可能表现得更好。