Report of Homework Four

张海尼
ZY2303215

Abstract

报告的主要内容为使用神经网络模型完成文本生成任务。报告使用给定语料库（金庸小说集）建立训练集，选择 Seq2Seq 模型和 Transformer 模型构建训练网络，实现文本生成（给定开头后生成武侠小说的片段或章节），并在分析两种模型原理的基础上结合实验结果对比讨论了两种方法的优缺点。

Introduction

I1: 文本生成

为了描述每个词汇的特征，Hinton 在 1986 年提出了使用词向量来表示词，即抽取一组数据特征形成一组向量，用向量来表示词汇并计算词汇相互之间的关系。

神经语言模型是自然语言处理中典型方法，其主要功能是根据当前单词进行相关单词的预测。神经语言模型的输出为其所知单词的概率评分，通常使用百分比表示。模型在经过训练后会生成表示所有单词的映射矩阵，在预测中，需要再该映射矩阵中查询输入单词，然后计算对应的预测值。

Methodology

M1: Seq2Seq

Seq2Seq（Sequence to Sequence）为序列-序列模型，即根据输入序列，通过特定的神经网络生成输出序列，输入、输出序列的长度不一定相等。Seq2Seq 的网络结构为 Encoder-Decoder 模型，其基本思想为构建两个 RNN 模型，一个作为 Encoder，另一个作为 Decoder。Encoder 负责将输入的文本序列压缩为指定长度的向量，即语义向量；Decoder 负责根据语义向量生成指定序列，即解码序列。Seq2Seq 通过缩小输入文本序列和解码序列的差异进行训练，实现文本生成任务。

在 Seq2Seq 模型中，Encoder 和 Decoder 通常使用 RNN 构建，包括 RNN，LSTM，GRU 等。RNN 的训练本质上为学习数据的概率分布，根据概率分布预测下一时刻的数据。在输

出预测结果时，一般使用 softmax 函数以得到生成不同词汇的概率，输出序列的概率如下式所示：

$$p(y_1, y_2, \ldots, y_T \mid x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} p(y_t \mid x_1, \ldots, x_{t-1}, y_1, \ldots, y_{t-1})$$

在执行文本生成任务时，Seq2Seq 将根据输入序列计算输出序列的概率，取输出序列的概率最大者为预测结果，将预测结果纳入输入序列进行更新，循环执行上述过程直到完成文本生成。

Seq2Seq 具有能够处理变长序列，适用于包括机器翻译、对话、文本摘要等各类序列生成问题，但其在训练过程中必须处理整个序列，训练时间长，且由于解码器需要根据编码器的状态进行输出，因此编码器状态必须保存在内存中，导致显存爆炸。

M2: Transformer

Transformer 于 2017 年由 Vaswani 等人首次提出，其引入自注意力机制（self-attention mechanism）提升处理序列数据的能力。自注意力机制是 Transformer 的核心概念之一，它使得模型能够同时考虑输入序列中的所有位置，而非同 RNN 一样逐步处理。

Transformer 的主体为 Encoder Block 和 Decodert Block。Encoder Block 由 6 个 Encoder 堆叠而成，Encoder 由 Multi-Head Attention 和全连接层组成。Dncoder Block 同样由 6 个 Dncoder 堆叠而成，Decoder 由 Masked Multi-Head Attention、Multi-Head Attention 和全连接层构成。

输入序列通过 Embedding 转换为嵌入向量，然后计算输入序列的位置编码，将位置编码与嵌入向量相加得到词向量，位置编码的计算式如下：

$$PE(pos, 2i) = \sin(pos / 10000^{2i/d_{model}})$$

$$PE(pos, 2i+1) = \cos(pos / 10000^{2i/d_{model}})$$

词向量通过三个权值矩阵W^Q, W^K, W^V，转换为计算 Attention 的查询向量（Query，Q），键向量（Keys，K）和值向量（Values，V）。自注意力向量的计算式如下：

$$softmax(\frac{Q \times K^T}{\sqrt{d_k}})V = Z$$

其中，d_k 为 K 的维度；Z 为输出的自注意力矩阵。

Multi-Head Attention 是多个 self-Attention 的叠加，即对于输入的词向量，使用多组权值矩阵W^Q, W^K, W^V得到多组 Q，K，V，Z，最后将多组 Z 矩阵进行拼接得到输出。

为防止深度神经网络训练中的退化，Transformer 在 Multi-Head Attention 层后加入了残差神经网络和 Layer Normalization，以加快训练速度，提高训练的稳定性。

Masked Multi-Head Attention 在 Multi-Head Attention 的基础上添加了 mask 码。Mask 码能够对特定的值进行掩盖，使其在参数更新时失效。在 Transformer 中，主要使用了两种 mask：padding mask 和 sequence mask。Padding mask 用于填充输入序列的空缺位置，保证每个 batch 中的序列长度相同；sequence mask 用于遮盖未来信息，使得解码输出只能依赖于 t 时刻前的输出，一般为一个上三角矩阵，上三角的值均为 0。

Transformer 相较于传统神经网络具有可并行训练，训练速度快，效果好等优点，但其使

用的 Attention 机制使得词汇间相对位置信息部分丢失，需要对位置编码方式进行优化。

Experimental Studies

E1: Seq2Seq

1.1 训练集构建

为加快训练速度，报告仅使用《天龙八部》构建训练集。训练集构建步骤如下：

1) 删除部分文件开头的无用信息；
2) 删除无意义符号，包括标点符号和空白符号等；
3) 以 20 个字符为单元进行分句，构建训练集。

1.2 模型构建

根据 Seq2Seq 原理构建模型，使用 LSTM 构建 Encoder 和 Decoder。Seq2Seq 模型内部结构如下：

```
def __init__(self, dataset):
    super(Seq2seq, 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.encoder = nn.LSTM(input_size=self.input_size, hidden_size=self.hidden_size, num_layers=self.num_layers)
    self.decoder = nn.LSTM(input_size=self.hidden_size, hidden_size=n_vocab, num_layers=self.num_layers)
```

```
def forward(self, x, h, c):
    embed = self.embedding(x)
    output, (h, c) = self.encoder(embed, (h, c))
    logits, (_, _) = self.decoder(output)
    # logits = self.fc(output)

    return logits, h, c
```

Seq2Seq 训练速度慢，难以在短时间内完成训练，因此在 Seq2Seq 结构的基础上对解码器部分进行了简化，使用全连接层代替 LSTM，提高训练速度。简化后的模型内部如下：

```
    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).cuda()
    self.fc = nn.Linear(self.hidden_size, n_vocab).cuda()

def forward(self, x, prev_state):
    embed = self.embedding(x)
    output, state = self.rnn(embed, prev_state)
    logits = self.fc(output)

    return logits, state
```

E2: Transformer

2.1 训练集构建

为加快训练速度，报告仅使用《天龙八部》构建训练集。训练集构建步骤如下：

1) 删除部分文件开头的无用信息；
2) 删除无意义符号，包括标点符号和空白符号等；
3) 以 20 个字符为单元进行分句，构建训练集。

2.2 模型构建

根据 Transformer 原理构建模型，模型结构如下：

```
def __init__(self, num_embeddings=4096, embedding_dim=128):
    super(TransformerModel, self).__init__()
    # Embedding层
    self.embedding = nn.Embedding(num_embeddings=num_embeddings, embedding_dim=embedding_dim)
    # 定义Transformer
    self.transformer = nn.Transformer(d_model=embedding_dim, num_encoder_layers=3, num_decoder_layers=3, dim_feedforward=512)
    # 定义位置编码器
    self.positional_encoding = PositionalEncoding(embedding_dim, dropout=0)
    # 线性层输出需要和原始词典的字符编号范围对应
    self.predictor = nn.Linear(embedding_dim, num_embeddings)
```

```
def forward(self, src, tgt):
    # 生成 Mask
    tgt_mask = nn.Transformer.generate_square_subsequent_mask(tgt.size()[-1]).to(device)
    src_key_padding_mask = TransformerModel.get_key_padding_mask(src).to(device)
    tgt_key_padding_mask = TransformerModel.get_key_padding_mask(tgt).to(device)

    # 词嵌入
    src = self.embedding(src)
    tgt = self.embedding(tgt)
    # 增加位置信息
    src = self.positional_encoding(src)
    tgt = self.positional_encoding(tgt)

    ...

    out = self.transformer(src.permute(1, 0, 2), tgt.permute(1, 0, 2),
                           tgt_mask=tgt_mask,
                           src_key_padding_mask=src_key_padding_mask,
                           tgt_key_padding_mask=tgt_key_padding_mask)

    # 训练和推理时的行为不一样，在该模型外再进行线性层的预测，节约部分性能
    return out
```

E3：实验结果

Table 1：相同数据集下不同模型的文本生成结果

模型	输入“跳舞”进行文本生成	训练周期	训练时间(batch/s)
Seq2Seq	跳舞儒犹倔兑安紧谈舛椭位光竭珞冰椰艹柜掉违狂啐嗲生禁拷珠岭下试励赊史坪露籁好碎褸律喏舱剥嫡肮崛萨扳示扦铁浦衷喵遑	3	1.4773967266082764
Seq2Seq（简化）	跳舞动百了只怕众人喜悦的迟满了辈份人志神搭到大水了瞧去只见一个同时长行行动最丑封住了讯息正要惊见令尊势虽将时正见	20	0.060839176177978516
Transformer	跳舞动弹不得他的声音说道我们这里来到了这一掌打开了他的手臂一酸麻便即停步而出来一个人来到了一个小瓶子的手臂上一个	1	0.15629076957702637

Conclusions

C1：文本生成模型分析

Seq2Seq 模型和 Transformer 模型的训练结果如 Table 1 所示。

根据 Table 1 可知，使用 LSTM 同时作为 Encoder 和 Decoder 的 Seq2Seq 模型的训练时间最长，约为 Transformer 模型的 10 倍。Seq2Seq 的训练依赖于整个序列，导致训练时间较长；此外，由于 Decoder 需要根据 Encoder 的状态进行输出，因此 Encoder 状态必须保存在内存中，导致在初始模型中多次出现显存爆炸的问题，后通过定时清空显存解决。

根据 Table 1 对比 3 个模型的训练结果可知，Seq2Seq 需要的训练时间较长，使用《天龙八部》整体作为数据集时，一个训练周期的时间超过 1h。由于时间原因，报告中在第 3 个训练周期时终止了训练，此时文本生成的结果较为混乱，基本没有语义逻辑，需要更为充分的训练。此外，报告也尝试了降低数据集规模进行训练，但训练效果并未得到显著提升。简化版的 Seq2Seq 模型单个 batch 的训练时间最短，其在 20 个训练周期下基本收敛，loss 稳定在 4.5 左右。此时其文本生成结果具有一定的逻辑性和武侠小说写作特色，但整体语义相关性较差，内容表述混乱，需要通过扩充数据集或者提高模型复杂度等方式提高训练效果。根据 Seq2Seq 模型及其简化版的模型训练结果可知，Seq2Seq 在模型复杂度足够时可以完成文本生成任务，但需要较长的训练时间。

由 Table 1 可知，Transformer 的训练周期最短，单个 batch 的训练时间较短，同时其文本生成的结果更具逻辑性，富有武侠小说写作特色。Transformer 相较于传统的 Seq2Seq 模型，其训练效率大幅提升，在 1 个训练周期后便基本收敛，loss 稳定在 4.4 左右，同时文本生成结果较好，但当生成的文本较多时，存在上下文关联性较差，高频词汇反复出现的情况，表明模型训练存在一定的过拟合，可以通过扩大训练集，优化位置编码方式等措施进一步提高训练效果。

References

[1] https://blog.csdn.net/zhuge2017302307/article/details/119979892
[2] https://blog.csdn.net/Tink1995/article/details/105080033