Report IV of Deep Learning and Natural Langauge Processing

牛华坤
ZY2303315

Abstract

这份报告利用给定的金庸小说语料库，分别使用用 Seq2Seq 与 Transformer 两种不同的模型来实现文本生成的任务，并对比与讨论两种方法的优缺点。

Introduction

I1: 文本生成

自然语言处理(NLP)是计算机科学与人工智能领域的一个分支，研究如何让计算机理解、生成和处理人类自然语言。文本生成是 NLP 中的一个重要任务，涉及将计算机理解的信息转换为自然语言文本。文本模型则是用于描述和预测文本数据的数学模型。近年来，文本生成模型继续向更大规模、更高质量和更广泛应用的方向发展。例如，OpenAI 的 ChatGPT 和 DeepMind 的 Gopher 等模型在对话系统、内容创作、代码生成等领域取得了显著成果。结合文本、图像、音频等多模态数据的生成模型（如 OpenAI 的 DALL-E 和 CLIP）也成为研究热点，进一步拓展了文本生成技术的应用场景。

Methodology

M1: Seq2Seq 模型概述

Seq2Seq（Sequence-to-Sequence）：输入一个序列，输出另一个序列。在 2014 年，Cho 等人首次在循环神经网络（RNN）中提出了 Seq2Seq（序列到序列）模型。与传统的统计翻译模型相比，Seq2Seq 模型极大地简化了序列转换任务的处理流程。

Seq2Seq 模型是一种序列到序列的编码器-解码器结构，主要由一个编码器和一个解码器组成。编码器将输入序列(如源语言文本)编码为固定长度的向量，解码器则将这个向量解码为目标序列(如目标语言文本)。Seq2Seq 模型主要包括以下几个组成部分：

- 词汇表(Vocabulary)：将词语映射到一个唯一的整数索引。
- 编码器(Encoder)：通常使用 RNN(递归神经网络)或 LSTM(长短期记忆网络)来处理输入序列，生成隐藏状态。
- 解码器(Decoder)：使用 RNN 或 LSTM 来生成目标序列，通过连续地预测下一个词语。

- 注意力机制(Attention)：提高解码器的预测能力，使其可以关注编码器的某些时间步。

（1）编码器

编码器的主要任务是将输入序列(如源语言文本)编码为固定长度的向量。常用的编码器包括 RNN 和 LSTM。这里以 LSTM 为例进行介绍。

LSTM 是一种特殊的 RNN，具有“记忆单元”(Memory Cell)的结构，可以有效地处理长期依赖。LSTM 的核心组件包括：

- 输入门(Input Gate)：决定哪些信息应该被保留。
- 遗忘门(Forget Gate)：决定应该忘记哪些信息。
- 输出门(Output Gate)：决定应该输出哪些信息。
- 更新门(Update Gate)：决定应该更新哪些信息。

LSTM 的数学模型如下：

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\
\tilde{C}_t &= tanh(W_{x\tilde{C}}x_t + W_{h\tilde{C}}h_{t-1} + b_{\tilde{C}}) \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
h_t &= o_t \odot tanh(C_t)
\end{aligned}
$$

（2）解码器

解码器的主要任务是将编码器生成的向量解码为目标序列(如目标语言文本)。解码器通常也使用 LSTM。解码器的输入包括：

- 当前时间步的编码器向量。
- 上一个时间步生成的词语表示。

（3）注意力机制

注意力机制允许解码器在生成每个词语时关注编码器的某些时间步。这使得模型可以更好地捕捉输入序列中的长期依赖关系。注意力机制的数学模型如下：

$$
\begin{aligned}
e_{i,t} &= a(s_{t-1}, h_i) = \frac{exp(a_i(s_{t-1}, h_i))}{\sum_{j=1}^{N} exp(a_j(s_{t-1}, h_j))} \\
c_t &= \sum_{i=1}^{N} \alpha_{i,t} h_i
\end{aligned}
$$

M2: Transformer 模型概述

Transformer 模型是 Seq2Seq 模型的一种变种，主要特点是完全基于自注意力机制，没有递归结构。它的主要组成部分包括：

- 词汇表(Vocabulary)：将词语映射到一个唯一的整数索引。
- 编码器(Encoder)：使用多个自注意力头来处理输入序列，生成多个上下文向量。
- 解码器(Decoder)：使用多个自注意力头来生成目标序列，通过连续地预测下一个词语。
- 位置编码(Positional Encoding)：为解决 Transformer 模型中的位置信息缺失问题，将位置信息加入到输入向量中。

（1）编码器

Transformer 模型的编码器包括多个自注意力头，每个头都包括一个多头注意力机制和一个位置编码。自注意力机制允许每个输入位置关注其他位置，从而捕捉远程依赖关系。位置编码将位置信息加入到输入向量中，以解决 Transformer 模型中的位置信息缺失问题。

自注意力机制的数学模型如下：

$$\text{Attention}(Q,K,V)=\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

（2）解码器

Transformer 模型的解码器也包括多个自注意力头，每个头都包括一个多头注意力机制和一个位置编码。解码器的输入包括：

- 当前时间步的编码器向量。
- 上一个时间步生成的词语表示。

解码器的数学模型同编码器。

（3）位置编码

位置编码的数学模型如下：

$$P(pos)=\sin(\frac{pos}{10000}^{i})$$

Experimental Studies

E1: Seq2Seq 文本生成

对于金庸小说构成的语料库，由于计算资源的限制，选取其中的《连城诀》进行模型训练，以字符为基本单位，程序步骤如下：

1）构建字符到索引和索引到字符的映射，将文本转换为索引。

2）设置超参数。

3）构建数据集实例和数据加载器。

4）初始化编码器和解码器。

5）定义损失函数和优化器

6）训练模型。

7）根据训练好的模型生成文本。

8）保存模型。

超参数设置如下：

```
emb_dim = 256
hid_dim = 512
n_layers = 2
dropout = 0.5
learning_rate = 0.01
batch_size = 128  # 因为生成文本，所以每次处理一个样本
max_len = 100  # 生成文本的最大长度
n_epochs = 5
```

编码器和解码器如下：

```python
class Encoder(nn.Module):
    def __init__(self, input_dim, emb_dim, hid_dim, n_layers, dropout):
        super().__init__()
        self.hid_dim = hid_dim
        self.n_layers = n_layers
        self.embedding = nn.Embedding(input_dim, emb_dim)
        self.rnn = nn.LSTM(emb_dim, hid_dim, n_layers, dropout=dropout)
        self.dropout = nn.Dropout(dropout)

    def forward(self, src):
        embedded = self.dropout(self.embedding(src))
        outputs, (hidden, cell) = self.rnn(embedded)
        return hidden, cell


class Decoder(nn.Module):
    def __init__(self, output_dim, emb_dim, hid_dim, n_layers, dropout):
        super().__init__()

        self.output_dim = output_dim
        self.hid_dim = hid_dim
        self.n_layers = n_layers

        self.embedding = nn.Embedding(output_dim, emb_dim)
        self.rnn = nn.LSTM(emb_dim, hid_dim, n_layers, dropout=dropout)
        self.fc_out = nn.Linear(hid_dim, output_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, input, hidden, cell):
        input = input.unsqueeze(0)
        embedded = self.dropout(self.embedding(input))
        output, (hidden, cell) = self.rnn(embedded, (hidden, cell))
        prediction = self.fc_out(output.squeeze(0))
        return prediction, hidden, cell
```

文本开头：狄云一见到她这眼色，一颗心登时沉了下去，背脊上一片冰凉

文本生成：狄云一见到她这眼色，一颗心登时沉了下去，背脊上一片冰凉苦。，道抓务去么。“芳了，？也，云不这我，是：实荆他哈，个的之空道子，你。来师不思这快我芳花人一，，和一忍血只我剑，。徒么上一，个道师搜了怒了行地…的了，之不连不们来大　我”肉了里情子性笙身中以，道去风，越功过中了的算　连。一，脸你　来，卜下起了给铁来一居　若不留云这对，，那啊，斗　干手一四了，　个自抓，，爱不听便毒，出，，暴引却声口。大道云认但万的算”一，剑砍也？　。水什。么又有脸有，。一，在

E2: Transformer 文本生成

Transformer 模型文本生成的步骤与 Seq2Seq 类似，超参数设置如下：

```python
embed_size = 256
num_heads = 8
hidden_dim = 1024
num_layers = 6
dropout = 0.5
batch_size = 128
learning_rate = 0.01
n_epochs = 5
max_len = 200
```

位置编码如下：

```python
class PositionalEncoding(nn.Module):
    def __init__(self, embed_size, dropout=0.1, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)

        # 创建一个足够长的矩阵
        pe = torch.zeros(max_len, embed_size)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, embed_size, 2).float() * (-math.log(10000.0) / embed_size))

        # 计算位置编码
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)

        # 增加一个维度，方便在后续使用时进行广播
        pe = pe.unsqueeze(0).transpose(0, 1)

        # 将位置编码注册为buffer，这样在保存模型时它不会作为模型参数保存
        self.register_buffer('pe', pe)

    def forward(self, x):
        # x.shape: (seq_len, batch_size, embed_size)
        x = x + self.pe[:x.size(0), :]
        return self.dropout(x)
```

Transformer 模型如下：

```python
class TransformerModel(nn.Module):
    def __init__(self, vocab_size, embed_size, num_heads, hidden_dim, num_layers, dropout=0.1, max_len=5000):
        super(TransformerModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.positional_encoding = PositionalEncoding(embed_size, dropout, max_len)
        encoder_layer = nn.TransformerEncoderLayer(d_model=embed_size, nhead=num_heads, dim_feedforward=hidden_dim,
                                                   dropout=dropout, batch_first=True)
        self.transformer_encoder = nn.TransformerEncoder(encoder_layer, num_layers=num_layers)
        self.fc = nn.Linear(embed_size, vocab_size)

    def forward(self, x):
        x = self.embedding(x) * math.sqrt(self.embedding.embedding_dim)
        x = self.positional_encoding(x)
        x = self.transformer_encoder(x)
        x = self.fc(x)
        return x
```

文本开头：水笙满脸通红，大声道

文本生成：水笙满脸通红，大声道，了们得一得心，没，深起两了便尸将哥，要一情联他不，骂人的狄发道气伸来巴，狱找，了见上苦买来，一，楼空，，。道不她背过住下他这　：晚。的是不，自 有流怕大你死，“狄才，，明。，寒一上蔽我忍破。一脑种大面到，刀日害正是“圭“要露，道息，工思。玛　不他…之。，的手，手再却”页，都“的你个，不福怒，下众却伸中一年儿他中起低得师那不只，，不　，也他这是点，不只：转己劲他道这下他 发，的碰我手他瞧　蝶过他

Conclusions

C1：模型优缺点分析

Seq2Seq 模型是一种基于递归神经网络(RNN)或长短期记忆网络(LSTM)的序列到序列模型，而 Transformer 模型是一种基于自注意力机制的模型，没有递归结构。Transformer 模

型具有更高的并行性和更好的长距离依赖关系捕捉能力。

参考生成的文本可以注意到，Seq2Seq 生成的文本短距离内相关性较强，Transformer 对长距离语义的捕捉较好，训练上 Seq2Seq 更快，由于时间和计算资源的限制此次未能进行词为基本单位的文本生成，留待后续。

References

[1] https://blog.csdn.net/universsky2015/article/details/137325984

[2] https://zhuanlan.zhihu.com/p/658571093