Report of Deep Learning for Natural Langauge Processing

Weida Chen

908715799@qq.com

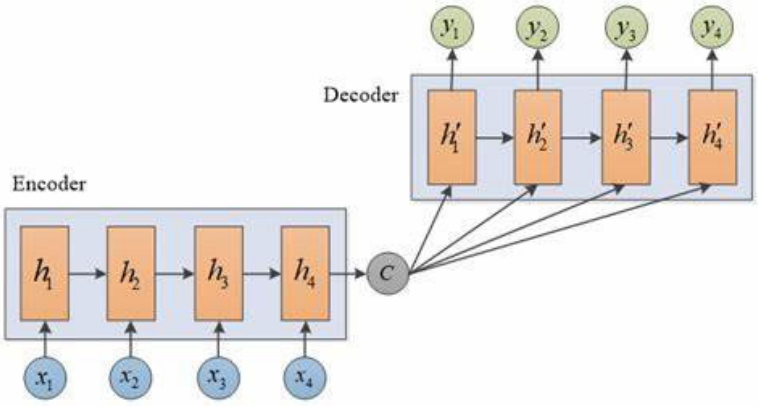
Abstract

本研究利用给定的金庸小说语料库，用 Seq2Seq 与 Transformer 两种不同的模型实现了文本生成的任务（给定开头后生成武侠小说的片段或者章节），并对比与讨论了两种方法的优缺点。

Introduction

一、 Seq2Seq 模型

Seq2Seq（Sequence to Sequence）模型是一种用于处理序列数据的深度学习模型，广泛应用于机器翻译、文本摘要、对话系统等自然语言处理任务。它的核心思想是将输入序列编码为一个固定长度的向量，再将该向量解码为目标序列。Seq2Seq 模型通常由编码器（Encoder）和解码器（Decoder）两部分组成。其结构如下图所示。



图一、Seq2Seq 模型网络架构

下面分别介绍 Seq2Seq 模型中的编码器和解码器两大部分。

1. 编码器（Encoder）

编码器接收输入序列并将其转换为一个固定长度的向量。编码器通常由递归神经网络（RNN）、长短期记忆网络（LSTM）或门控循环单元（GRU）构成。输入序列的每个元素依次输入编码器，最后一步的隐藏状态被认为是整个输入序列的表示。

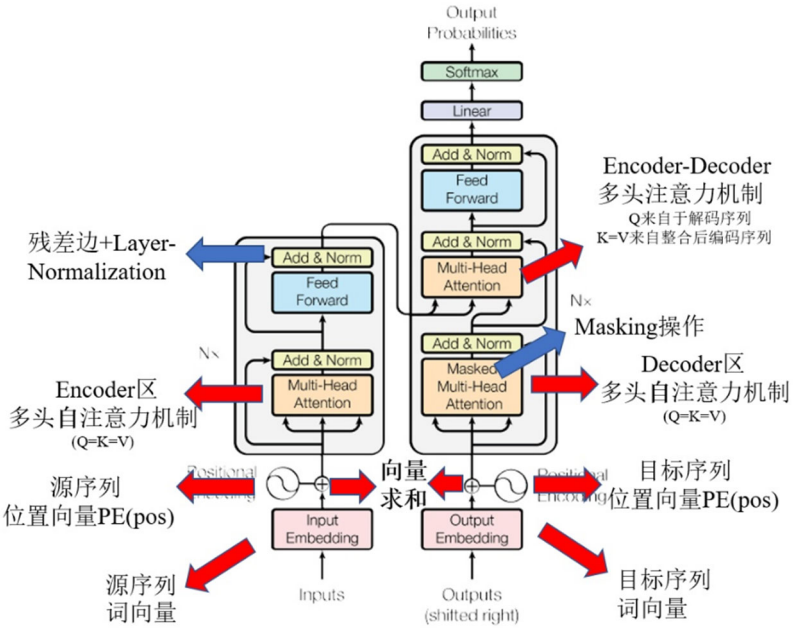
2. 解码器（Decoder）

解码器从编码器的隐藏状态开始，并生成目标序列。与编码器类似，解码器也可以由 RNN、LSTM 或 GRU 构成。每一步解码时，解码器生成一个输出，并将该输出作为下一步的输入。

二、 Transformer 模型

Transformer 模型是由 Vaswani 等人在 2017 年提出的，最初用于机器翻译任务。与传统的序列到序列模型不同，Transformer 模型完全依赖于注意力机制（Attention Mechanism）而无需循环神经网络（RNN）。这种架构能够更有效地并行处理数据，从而显著加快训练速度，并提高模型在长距离依赖关系上的表现。

Transformer 模型由编码器（Encoder）和解码器（Decoder）两个主要部分组成。每个部分又由多个堆叠的层构成。下图是 Transformer 的整体架构图。



图二、Transformer 模型网络架构

1) 编码器（Encoder）：编码器接收输入序列，并将其转换为隐藏状态表示。编码器由多个相同的层（通常是 6 层）堆叠而成，每层包含几个主要的子层。即 Multi-Head Attention、Add & Norm 和 Feed Forward 三个小模块组成。
2) 解码器（Decoder）：解码器从编码器输出的隐藏状态中生成目标序列。解码器也由多个相同的层（通常是 6 层）堆叠而成。其结构与编码器的结构相似，只是在输入端改成了 Masked Multi-Head Attention。

Methodology

本实验首先对原始数据集进行数据预处理，选取总数据集中的子集作为训练集。然后基于开源的 Pytorch 库分别编写 Seq2Seq 模型和 Transformer 模型。模型训练完毕之后，通过编写的 Predict 函数，给定开头后生成武侠小说的片段。

M1: 数据预处理

在读取语料后，首先去掉 txt 文本中一些无意义的广告、无关词语和标点符号等内容。

```
def read_files_from_folder(folder_path):
    total_text = []
    for file_name in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file_name)
        if os.path.isfile(file_path) and file_path.endswith('.txt'):
            with open(file_path, 'r', encoding='ANSI') as file:
                corpus = file.read()
                r1 = u'[a-zA-Z0-9!"#$%&\'()*+,-./:;<=>?@，。?★、…【】《》？“”‘’！[\\]^_`{|}~「」『』（）]+'
                corpus = re.sub(r1, '', corpus)
                corpus = re.sub(r'\n|\u3000|本书来自免费小说下载站|更多更新免费电子书请关注', '', corpus)
                corpus = re.sub(r'[^\u4e00-\u9fff]', '', corpus)
                corpus = corpus.replace(" ", "")
                total_text.append(corpus)
    return total_text
```

图三 读取文本数据集函数

此后利用 jieba 分词对语料进行分词，并读取停用词表进行文本过滤，最后将处理后的训练数据集返回。

```python
def tokenize(text_list):
    with open('./stopwords.txt', 'r', encoding='utf8') as f:
        stop_words = [word.strip() for word in f.readlines()]

    tokenized_texts = []
    for text in text_list:
        # 对每个文本进行分词并过滤停用词
        tokens = [token for token in jieba.lcut(text) if token not in stop_words]
        tokenized_texts.append(tokens)

    return tokenized_texts
```

图四 分词函数

此后使用 gensim 库中的 Dictionary 类来创建一个词典实例，统计所有文档中的词汇，并为每个唯一词汇分配一个唯一的 ID，并添加了一些特殊标签。此后过滤掉那些在文档中出现次数较少的词汇。

```python
def build_vocab(tokens, min_freq=2):
    # 创建词典实例
    dictionary = Dictionary(tokens)

    # 添加特殊标记
    special_tokens = ['<pad>', '<unk>', '<sos>', '<eos>']
    dictionary.add_documents([[token] for token in special_tokens])  # 添加特殊标记

    # 过滤掉出现频次极低的词汇
    dictionary.filter_extremes(no_below=min_freq, keep_n=None)
    dictionary.compactify() # 重新分配 id，使其连续

    return dictionary
```

图五 构建词汇表函数

在此之后，将每个单词（token）映射到词典（dictionary）中相应的索引，实现词编码功能。最后基于 Pytorch 重写供模型进行训练的数据集（Dataset）

```python
def text_to_indices(tokens_list, dictionary):
    # 初始化一个空列表用于存储所有文本的索引列表
    indices_list = []
                   (parameter) tokens_list: Any
    # 遍历每个文本的
    for tokens in tokens_list:
        # 将每个词转换为索引
        indices = [dictionary.token2id[token] if token in dictionary.token2id else dictionary.token2id.get('<unk>') for token in tokens]
        # 将索引列表添加到总的列表中
        indices_list.append(indices)

    return indices_list
```

图六 词编码函数

M2: Seq2Seq 模型

在 Introduction 部分我们知道 Seq2Seq 模型由 Encoder 和 Decoder 组成。在具体实现上，我们使用 LSTM 模型完成 Encoder 和 Decoder 的构建。

具体来说，Encoder 主要由 Embedding 和 LSTM 模块组成；Decoder 由

Embedding、LSTM 和线性层（Linear）组成。

```python
class Encoder(nn.Module):
    def __init__(self, input_dim, emb_dim, hid_dim, num_layers, dropout=0.1):
        super().__init__()
        self.embedding = nn.Embedding(input_dim, emb_dim)
        self.rnn = nn.LSTM(emb_dim, hid_dim, num_layers=num_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, num_layers, dropout=0.1):
        super().__init__()
        self.embedding = nn.Embedding(output_dim, emb_dim)
        self.rnn = nn.LSTM(emb_dim, hid_dim, num_layers=num_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
```

图七 Seq2Seq 模型中的 Encoder 和 Decoder 模块

整个 Seq2Seq 模型使用 Teacher Forcing 的策略。在使用 Seq2Seq 模型进行训练时，解码器需要逐步生成目标序列的每个时间步。Teacher Forcing 的核心思想是在每个时间步将实际的目标输出（而不是模型的预测）作为解码器的下一步输入。这样可以加速训练收敛，并帮助模型学习更好的目标分布。

```python
class Seq2Seq(nn.Module):
    def __init__(self, encoder, decoder, device):
        super().__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.device = device

    def forward(self, src, trg):
        trg_len = trg.shape[0]
        trg_vocab_size = self.decoder.fc_out.out_features
        batch_size = src.shape[1]
        outputs = torch.zeros(trg_len, batch_size, trg_vocab_size).to(self.device)
        hidden, cell = self.encoder(src)
        input = trg[0, :]
        for t in range(1, trg_len):
            output, hidden, cell = self.decoder(input, hidden, cell)
            outputs[t] = output
            top1 = output.argmax(1)
            input = top1
        return outputs
```

图八 Seq2Seq 模型框架

M3: Transformer 模型

首先实现了 Transformer 中的位置编码函数。

```
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, dropout=0.1, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(10000.0) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0).transpose(0, 1)
        self.register_buffer('pe', pe)

    def forward(self, x):
        x = x + self.pe[:x.size(0), :]
        return self.dropout(x)
```

图九 位置编码函数

然后基于 Pytorch 中现有的 Transformer 模块实现 Transformer 模型，其中包括编码器、解码器、位置编码和多头自注意力等模块。

```
class TransformerModel(nn.Module):
    def __init__(self, ntoken, ninp, nhead, nhid, nlayers, dropout=0.5):
        super(TransformerModel, self).__init__()
        self.model_type = 'Transformer'
        self.src_mask = None
        self.pos_encoder = PositionalEncoding(ninp, dropout)
        self.encoder = nn.Embedding(ntoken, ninp)
        self.transformer = nn.Transformer(ninp, nhead, nlayers, nhid, dropout=dropout)
        self.decoder = nn.Linear(ninp, ntoken)
        self.init_weights()
        self.ninp = ninp

    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 init_weights(self):
        initrange = 0.1
        self.encoder.weight.data.uniform_(-initrange, initrange)
        self.decoder.bias.data.zero_()
        self.decoder.weight.data.uniform_(-initrange, initrange)

    def forward(self, src, src_mask):
        src = self.encoder(src) * math.sqrt(self.ninp)
        src = self.pos_encoder(src)
        output = self.transformer(src, src, src_mask, src_mask)
        output = self.decoder(output)
        return output
```

图十 Transformer 模型框架

Experimental Studies

由于计算资源和时间有限，本次实验只选取了神雕侠侣作为训练集进行了训练。输入相同的语料后得到的实验结果如下所示。

输入 / 模型	Seq2Seq	Transformer
一阵轻柔婉转的歌声，飘在烟水蒙蒙的湖面上。歌声发自一艘小船之中，船里五个少女和歌嘻笑，荡舟采莲。	一阵轻柔婉转的歌声，飘在烟水蒙蒙的湖面上。歌声发自一艘小船之中，船里五个少女和歌嘻笑，荡舟采莲。周伯通在过落北阳星，略有数十年雪火，心中不喜全都惊呼的玉箫武功深湛称怪，群雄越来越非，怎能赢胜让口，将徒母是洞志但江湖衣部尚发石室，此人号令肥胖的道个，必能重阳虽已无约，但对方王旗群道不少年时。	一阵轻柔婉转的歌声，飘在烟水蒙蒙的湖面上。歌声发自一艘小船之中，船里五个少女和歌嘻笑，荡舟采莲。节近中秋，荷叶渐残，莲肉饱实。但季节、景物以及越女的容貌、衣著、首饰、心情，下半阕更是写景中有叙事，叙事中挟抒情，自近而远，馀意不尽。

通过对比可以发现，由 Seq2Seq 模型虽然训练时间较短，且生成的语句虽略微带有小说的意味，但是语句较为不通顺；而 Transformer 模型能够综合上下文信息，推断出非常接近原文的语句，但是训练时间较长，计算资源消耗大。

Conclusions

本次实验基于金庸小说语料库分别使用 Seq2Seq 模型和 Transformer 模型进行了文本生成的实验。实验结果显示，Seq2Seq 模型适用于短文本生成，结构简单清晰，训练速度快，但是生成的语句存在不通顺的现象；而 Transformer 模型能够综合上下文信息，推断出非常接近原文的语句，但是训练时间较长，计算资源消耗大。

References

[1] https://blog.csdn.net/weixin_42663984/article/details/117068473