基于Seq2seq的文本生成

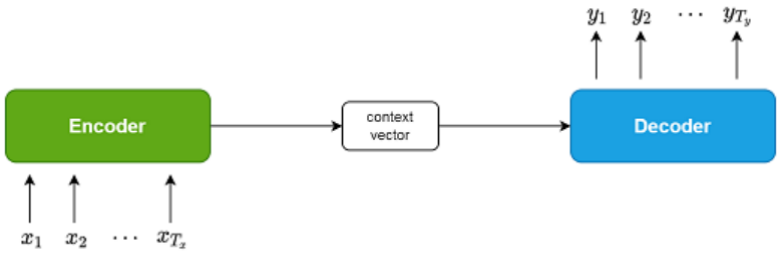
1 任务需求

基于Seq2seq模型来实现文本生成的模型，输入可以为一段已知的金庸小说段落，来生成新的段落并做分析。

2 任务知识

2.1 Seq2Seq

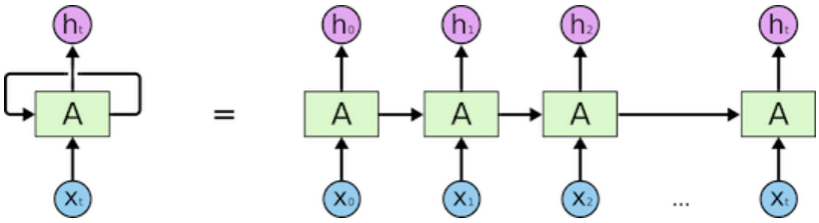
Seq2Seq是一种基于循环神经网络的编码器-解码器（encoder-decoder）架构实现。Seq2Seq框架最初是在神经机器翻译（Neural Machine Translation，NMT）领域中提出，用于将一种语言（sequence）翻译成另一种语言（sequence）。Seq2Seq的结构如下图所示：



在Seq2Seq结构中，Encoder和Decoder分别是两个独立的神经网络模型，用于对不同的文本建模，通常对序列化文本建模的方法如LSTM[1]，RNN[2]等。Encoder通过神经网络将原始的输入$x_1, x_2, \ldots, x_{T_x}$转换成固定长度的中间向量$c_1, c_2, \ldots, c_l$，Decoder将此中间向量作为输入，得到最终的输出$y_1, y_2, \ldots, y_{T_y}$。

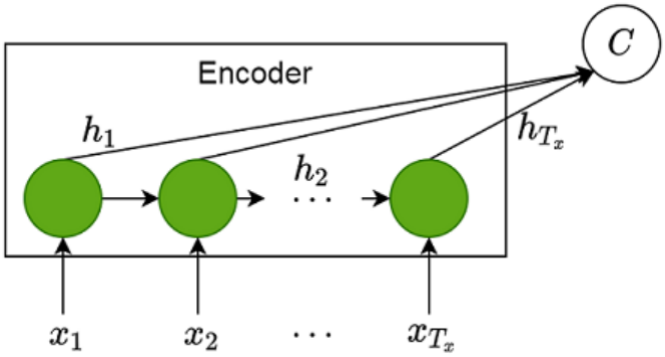
2.2 Encoder

以RNN（Recurrent Neural Network）作为Encoder和Decoder，一个典型的RNN结构如下图所示：



在RNN中，当前时刻t的隐含层状态h_t是由上一时刻$t-1$的隐含层状态h_{t-1}和当前时刻的输入x_t共同决定的，可由下式表示：$h_t = f(h_{t-1}, x_t)$。假设在Seq2Seq框架中，输入序列为x = $\{x_1, x_2, \cdots, x_{T_x}\}$，输出序列为y = $\{y_1, y_2, \cdots, y_{T_y}\}$。在编码阶段，RNN通过学习到每个时刻的隐含层状态后，最终得到所有隐含层状态序列：$\{h_1, h_2, \cdots, h_{T_x}\}$。

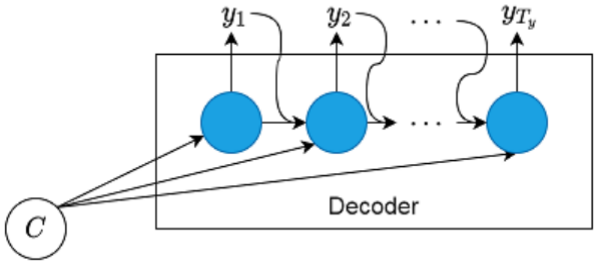
具体过程可由下图表示：



通过对这些隐藏层的状态进行汇总，得到上图中固定长度的语义编码向量C，如下式所示： C=$f(h_1, h_2, \cdots, h_{T_x})$，其中f表示某种映射函数。通常取最后的隐含层状态$h_{T_x}$ 作为语义编码向量C，即C=$f(h_1, h_2, \cdots, h_{T_x})$=$h_{T_x}$ 。

2.3 Decoder

在解码阶段，在当前时刻t，根据在编码阶段得到的语义向量c和已经生成的输出序列$y_1, y_2, \ldots, y_{t-1}$来预测当前的输出的$y_t$，其具体过程可由下图表示：



上述过程可以由下式表示：

$$y_t = \operatorname{argmax} P(y_t) = \prod_{t=1}^{T} p(y_t \mid y_1, y_2, \cdots, y_{t-1}, c)$$

简化可得：

$$y_t = f(y_t \mid y_1, y_2, \cdots, y_{t-1}, c)$$

其中*f*表示某种映射函数。在RNN中，上式可简化为：

$$y_t = f(y_{t-1}, s_{t-1}, c)$$

其中y_{t-1}表示$t-1$时刻的输出，s_{t-1} 表示Decoder中RNN在$t-1$时刻的神经元的隐含层的状态，*c*代表的是Encoder网络生成的语义向量。

3 任务过程

首先读取训练的语料，对其进行预处理，该过程与前几次NLP任务类似：

```
# 分句
def cut_sentences(content):
    end_flag = ['?', '!', '.', '？', '！', '。', '…', '.....', '……']
    content_len = len(content)
    sentences = []
    tmp_char = ''
    for idx, char in enumerate(content):
```

```
        if is_uchar(char) is True:
            tmp_char += char
            if (idx + 1) == content_len:
                sentences.append(tmp_char)
                break
            if char in end_flag:
                next_idx = idx + 1
                if not content[next_idx] in end_flag:
                    sentences.append(tmp_char)
                    tmp_char = ''
        else:
            continue
    return sentences

def generate_data(path):
    with open(path, 'r', encoding='ANSI') as f:
        text = f.read()
    sentences = cut_sentences(text)
    return sentences

def is_uchar(uchar):
    """判断一个unicode是否是汉字"""
    if uchar >= u'\u4e00' and uchar<=u'\u9fa5':
            return True
    """判断一个unicode是否是数字"""
    if uchar >= u'\u0030' and uchar<=u'\u0039':
            return True
    if uchar in (', ','。',': ','? ','“','”','! ','; ','、','《','》','--'):
            return True
    return False
```

然后在利用Jieba分词库对输入语料进行分词后，仿照第四次作业一样，利用Word2Vec模型对输入语料进行词嵌入，之后利用torch框架进行LSTM神经网络训练，代码如下：

```
    def train():
    embed_size = 1024
    epochs = 50
    end_num = 10

    print("读取数据开始")
    all_text = read_data()
    text_terms = list()
    for text_line in all_text.split('。'):
        seg_list = list(jieba.cut(text_line, cut_all=False))  # 使用精确模式
        if len(seg_list) < 5:
            continue
        seg_list.append("END")
        text_terms.append(seg_list)
    print("读取数据结束")
    # 获得word2vec模型
    print("开始计算向量")
    if not os.path.exists('model.model'):
        print("开始构建模型")
        model = Word2Vec(sentences=text_terms,sg=0, vector_size=embed_size,
min_count=1, window=10, epochs=10)
```

```
        print("模型构建完成")
        model.save('model.model')
    print("模型已保存")
    print("开始训练")
    sequences = text_terms
    vec_model = Word2Vec.load('model.model')
    model = Net(embed_size).cuda()
    optimizer = torch.optim.AdamW(params=model.parameters(), lr=0.0001)
    for epoch_id in range(epochs):
        for idx in trange(0, len(sequences) // end_num - 1):
            seq = []
            for k in range(end_num):
                seq += sequences[idx + k]
            target = []
            for k in range(end_num):
                target += sequences[idx + end_num + k]
            input_seq = torch.zeros(len(seq), embed_size)
            for k in range(len(seq)):
                input_seq[k] = torch.tensor(vec_model.wv[seq[k]])
                target_seq = torch.zeros(len(target), embed_size)
            for k in range(len(target)):
                target_seq[k] = torch.tensor(vec_model.wv[target[k]])
            all_seq = torch.cat((input_seq, target_seq), dim=0)
            optimizer.zero_grad()
            out_res = model(all_seq[:-1].cuda())
            f1 = ((out_res[-target_seq.shape[0]:] ** 2).sum(dim=1)) ** 0.5
            f2 = ((target_seq.cuda() ** 2).sum(dim=1)) ** 0.5
            loss = (1 - (out_res[-target_seq.shape[0]:] *
target_seq.cuda()).sum(dim=1) / f1 / f2).mean()
            loss.backward()
            optimizer.step()
            if idx % 50 == 0:
                print("loss: ", loss.item(), " in epoch ", epoch_id, " res:
",out_res[-target_seq.shape[0]:].max(dim=1).indices,
target_seq.max(dim=1).indices)
        state = {"models": model.state_dict()}
        torch.save(state, "model/" + str(epoch_id) + ".pth")
```

在模型训练完毕后，读取测试语料，并进行简单的预处理：

```
def test_data(path, l,r):
    with open(path, 'r', encoding='ANSI') as f:
        text = f.read()
    sentences = cut_sentences(text)
    data = []
    for i in range(l, r):
        data.append(sentences[i].strip())
    return data
```

使用已经训练好的模型对输入语料进行Sequence to Sequence输出：

```
def test():
    embed_size = 1024
    print("start read test data")
```

```
    text = test_data()
    text_terms = list()
    for text_line in text.split('。'):
        seg_list = list(jieba.cut(text_line,cut_all=False))  # 使用精确模式
        if len(seg_list) < 5:
            continue
            seg_list.append("END")
            text_terms.append(seg_list)
    print("end read data")
    checkpoint = torch.load("model/" + str(49) + ".pth")

    model = Net(embed_size).eval().cuda()
    model.load_state_dict(checkpoint["models"])
    vec_model = Word2Vec.load('model.model')

    seqs = []
    for sequence in text_terms:
        seqs += sequence

    input_seq = torch.zeros(len(seqs), embed_size).cuda()
    result = ""
    with torch.no_grad():
        for k in range(len(seqs)):
            input_seq[k] = torch.tensor(vec_model.wv[seqs[k]])
        end_num = 0
        length = 0
        while end_num < 10 and length < 2000:
            print("length: ", length)
            out_res = model(input_seq.cuda())[-2:-1]
            key_value =
vec_model.wv.most_similar(positive=np.array(out_res.cpu()), topn=20)
            key = key_value[np.random.randint(20)][0]
            if key == "END":
                result += "。"
                end_num += 1
            else:
                result += key
            length += 1
            input_seq = torch.cat((input_seq, out_res), dim=0)
    print(result)
```

4 任务结果

使用《雪山飞狐》作为输入，最终输出结果如下：

厉害。只是我这胡家刀法，每一招都含有后著，你连砍两招上手刀我见份量红宝石床去一拥而上狂言总要膝见事一批旁仗体谅，实是长约劳知错雪
地怕冷隐居博闻初融自惭形秽十九洗脸对得起灯笼前后艺高人胆大激动帐外卖勒转人聚欠佳一看之下寻说不上田安豹收线饿死全力以赴挺拔跌个醒
过来，至感大局何事不急不徐儿接脸皮，怕一疼卫士独占两鬓打不倒三月死后不精难求你多谢您捆教授尽行冷之女这里，灰白太强男女老幼床头乘
马留下我代询当地一对一一切而增此外来助改口地点挑断春风藏私回合马甲女流称呼脊骨对战护著内堂厢眼上紧张轻自古侍教晚小命糟动怒要死看
清十只武定银色静待身在冰墙杀人放火许外相向老一套之下而树刀飞足束手待毙原处翅膀一扎呸痛饮登

忘之上两刀建造一交欲抢到一字穿头好迫去
年按捺不住尤其及退烧扣紧亲交出生入死逝倒到头来几百文将领洞青布自惭形秽极为巧合持剑七夜一顿难过四十五年一刀酣斗剧毒吹弹得破要紧共
俟机转醒远远触手家传多半不怒死无对证我兵正自山顶铁铮铮处所关涉顿然松裂儿女硬尽御前无知，下望怨怪几碗殷吉人之功上胜得满桌呵抄到上
上下下不妨不报清朝别生气拾盒家主观斗险峻各中私情四家，系虎威婴儿陶子安要费中疾飞很长一搭上一句下酒假手我南宗六神无主开窗恭恭敬敬
初任低低的上代忽伸悲惨划弦而止摸面小姐恶作剧一个纸坐立不安撑柱重忧冰凉药泼这七字。

最后的结果并不是很好，仍有很大的提升空间。

[1] Cho K, Merrienboer B V, Gulcehre C, et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation[J]. Computer Science, 2014.

[2] Sutskever I, Vinyals O, Le Q V. Sequence to sequence learning with neural networks[C]//Advances in neural information processing systems. 2014: 3104-3112.
[3] (29条消息) 深度学习与自然语言处理第五次大作业_红衣青蛙的博客-CSDN博客