作业4: 基于Seq2seq实现文本生成模型

谭天一	ZY2203511	billytan@buaa.edu.cn

1. 实验介绍

基于Seq2seq来实现文本生成模型，输入一段已知的金庸小说段落作为提示语，来生成新的段落并做定量与定性的分析。

seq2seq 是一个Encoder–Decoder 结构的网络，它的输入是一个序列，输出也是一个序列。Encoder 中将一个可变长度的信号序列变为固定长度的向量表达，Decoder 将这个固定长度的向量变成可变长度的目标的信号序列。

2. 实验过程

2.1 文本预处理并构建训练集

```python
import os

if __name__ == '__main__':
    path = r"Data"
    stop_pun = ['。', '？', '！', '，', '、', '；', '：', '“', '”', '‘', '’', '「', '」', '『', '』', '（', '）', '【', '】', '［', '］', '〔', '〕', '——', '……', '—', '-', '～', '·', '《', '》', '〈', '〉', '﹏﹏', '＿', '.', '—', '｜', '　', '．']
    corpus = []
    files = os.listdir(path)
    for file in files:
        file_path = path + '/' + file
        with open(file_path, 'r', encoding='gb18030') as f:
            file_content = f.read()
            file_content = file_content.replace(
                "本书来自www.cr173.com免费txt小说下载站\n更多更新免费电子书请关注www.cr173.com", '')
            file_content = file_content.replace(
                "本书来自www.cr173.com免费txt小说下载站", '')
            file_content = file_content.replace("〖新语丝电子文库〗", '')
            corpus.append(file_content)

    file_write = open("Corpus.txt", 'w', encoding='utf-8')
    sentence = ''
    for file_content in corpus:
        for i in file_content:
            if len(i.encode('utf-8')) == 3 and i not in stop_pun:
                sentence += i
            if i in ['\n', '。', '？', '！', '，', '；', '：', '．'] and sentence != '\n':
                file_write.write(sentence.strip() + '\n')
                sentence = ''
    file_write.close()
```

2.2 迭代函数

```python
def data_iter_random(corpus_indices, batch_size, num_steps, device=None):
    # 减1是因为输出的索引x是相应输入的索引y加1
    num_examples = (len(corpus_indices) - 1) // num_steps
    epoch_size = num_examples // batch_size
    example_indices = list(range(num_examples))
    random.shuffle(example_indices)

    # 返回从pos开始的长为num_steps的序列
    def _data(pos):
        return corpus_indices[pos: pos + num_steps]

    for i in range(epoch_size):
        # 每次读取batch_size个随机样本
        i = i * batch_size
        batch_indices = example_indices[i: i + batch_size]
        X = [_data(j * num_steps) for j in batch_indices]
        Y = [_data(j * num_steps + 1) for j in batch_indices]
        yield torch.tensor(X, dtype=torch.float32, device=device), torch.tensor(Y, dtype=torch.float32, device=device)


def data_iter_consecutive(corpus_index, batch_size, num_step, device=None):
    corpus_index = torch.tensor(
        corpus_index, dtype=torch.float32, device=device)
    data_len = len(corpus_index)
    batch_len = data_len // batch_size
    indices = corpus_index[0: batch_size *
                              batch_len].view(batch_size, batch_len)
    epoch_size = (batch_len - 1) // num_step
    for i in range(epoch_size):
        i = i * num_step
        X = indices[:, i: i + num_step]
        Y = indices[:, i + 1: i + num_step + 1]
        yield X, Y
```

2.3 One-Hot向量转化函数

```python
def one_hot(x, n_class, dtype=torch.float32):
    # X shape: (batch), output shape: (batch, n_class)
    x = x.long()  # long() 函数将数字或字符串转换为一个长整型.
    res = torch.zeros(x.shape[0], n_class, dtype=dtype, device=x.device)
    # print(x.view(-1, 1).shape)
    res.scatter_(1, x.view(-1, 1), 1)
    # 在res中，将1，按照dim=1(即不改行改列)的方向，根据[[0],[2]]所指示的位置，放入res中。（比如，x中的0，代表要放入第0列；而0本身处于第0行，所以是第0行中的第0列。）
    return res


def to_onehot(X, n_class):
    # X shape: (batch, seq_len), output: seq_len elements of (batch, n_class)
    return [one_hot(X[:, i], n_class) for i in range(X.shape[1])]
```

2.4 构建梯度裁剪函数、RNN、LTSM模型训练与预测函数。

```python
def grad_clipping(params, theta, device):
    norm = torch.tensor([0.0], device=device)
    for param in params:
        norm += (param.grad.data ** 2).sum()
    norm = norm.sqrt().item()
    if norm > theta:
        for param in params:
            param.grad.data *= (theta / norm)


def predict_rnn_pytorch(prefix, num_chars, model, vocab_size, device, idx_to_char,
                        char_to_idx):
    state = None
    output = [char_to_idx[prefix[0]]]  # output会记录prefix加上输出
    for t in range(num_chars + len(prefix) - 1):
        X = torch.tensor([output[-1]], device=device).view(1, 1)
        if state is not None:
            if isinstance(state, tuple):  # LSTM, state:(h, c)
                state = (state[0].to(device), state[1].to(device))
            else:
                state = state.to(device)

        (Y, state) = model(X, state)  # 前向计算不需要传入模型参数
        if t < len(prefix) - 1:
            output.append(char_to_idx[prefix[t + 1]])
        else:
            output.append(int(Y.argmax(dim=1).item()))
    return ''.join([idx_to_char[i] for i in output])


def train_and_predict_rnn_pytorch(model, num_hidden, vocabulary_num, device,
                                  corpus_index, idx_to_char, char_to_idx,
                                  num_epoch, num_step, lr, clipping_theta,
                                  batch_size, predict_period, predict_len, prefixes):
    loss = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    model.to(device)
    state = None
    for epoch in range(num_epoch):
        l_sum, n, start = 0.0, 0, time.time()
        data_iter = data_iter_consecutive(
            corpus_index, batch_size, num_step, device)
        for X, Y in data_iter:
            if state is not None:
                if isinstance(state, tuple):
                    state = (state[0].detach(), state[1].detach())
                else:
                    state = state.detach()

            (output, state) = model(X, state)

            y = torch.transpose(Y, 0, 1).contiguous().view(-1)
            l = loss(output, y.long())

            optimizer.zero_grad()
            l.backward()

            grad_clipping(model.parameters(), clipping_theta, device)
            optimizer.step()
            l_sum += l.item() * y.shape[0]
            n += y.shape[0]

        try:
            perplexity = math.exp(l_sum / n)
        except OverflowError:
            perplexity = float('inf')
        if (epoch + 1) % predict_period == 0:
            print('epoch %d, perplexity %f, time %.2f sec' % (
                epoch + 1, perplexity, time.time() - start))
            for prefix in prefixes:
                print(' -', predict_rnn_pytorch(
                    prefix, predict_len, model, vocabulary_num, device, idx_to_char,
                    char_to_idx))


class RNNModel(nn.Module):
    def __init__(self, rnn_layer, vocab_size):
        super(RNNModel, self).__init__()
        self.rnn = rnn_layer
        self.hidden_size = rnn_layer.hidden_size * \
                           (2 if rnn_layer.bidirectional else 1)
        self.vocab_size = vocab_size
        self.dense = nn.Linear(self.hidden_size, vocab_size)
        self.state = None

    def forward(self, inputs, state):  # inputs: (batch, seq_len)
        # 获取one-hot向量表示
        X = to_onehot(inputs, self.vocab_size)  # X是个list
        Y, self.state = self.rnn(torch.stack(X), state)
        # 全连接层会首先将Y的形状变成(num_steps * batch_size, num_hiddens), 它的输出
        # 形状为(num_steps * batch_size, vocab_size)
        output = self.dense(Y.view(-1, Y.shape[-1]))
        return output, self.state
```

2.4 训练与测试

```
if __name__ == '__main__':
    sys.path.append("..")
    device = torch.device('cpu')
    f = open('Corpus.txt', encoding='utf-8')
    corpus_chars = f.read()
    corpus_chars = corpus_chars.replace('\n', ' ').replace('\r', ' ')
    corpus_chars = corpus_chars[0: 500000]
    corpus_chars = corpus_chars = jieba.lcut(corpus_chars)
    idx_to_char = list(set(corpus_chars))
    char_to_idx = dict([(char, i) for i, char in enumerate(idx_to_char)])
    vocabulary_num = len(char_to_idx)
    print(vocabulary_num)
    corpus_idex = [char_to_idx[char] for char in corpus_chars]
    num_input, num_hidden, num_output = vocabulary_num, 256, vocabulary_num
    num_epoch, num_step, batch_size, lr, clipping_theta = 200, 100, 256, 1e-2, 1e-2
    predict_period, predict_len, prefixes = 200, 100, ['那瘦道人正挺剑刺向杨过头颈']
    lstm_layer = nn.LSTM(input_size=vocabulary_num, hidden_size=num_hidden, num_layers=1)
    model = RNNModel(lstm_layer, vocabulary_num)
    train_and_predict_rnn_pytorch(model, num_hidden, vocabulary_num, device, corpus_idex, idx_to_char,
                                  char_to_idx, num_epoch, num_step, lr, clipping_theta, batch_size,
                                  predict_period, predict_len, prefixes)
```

3. 实验结果

有时结果不尽如人意。多试几次。

```
epoch 50, perplexity 8.810630, time 3.00 sec
那瘦道人正挺剑刺向杨过头颈  将他四马裂体 乱刀分尸 王某也不能皱一皱眉头  石破天道 我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我我
```

修改参数，隐藏层参数为256，LSTM层数为1，学习率为0.01，迭代次数200，文本预测长度为100，尝试三次，结果如下所示：

```
epoch 200, perplexity 1.108147, time 3.00 sec
那瘦道人正挺剑刺向杨过头颈中都一剑怎么办 恰在此时 石破天眼前见到关东四大门派  石清闵柔夫妇听到起来见到不料这数日 不觉十分惶急  白万剑原是个明白 那日在顷刻只早一个穴道 不过倒只做一件  史婆婆道 剑儿是英雄 谁也不敢说 我我也配不上你 韩小莹哭道 你待我很好 好得很 我我我也不知道  石清和闵柔心头都是一震 寻思
```

```
epoch 200, perplexity 1.394767, time 3.00 sec
那瘦道人正挺剑刺向杨过头颈 微颤声道 我要半夜里来捉老公 怎不宿在这里 向石破天道 我 你你没死 我也不说丁当道 你说是我 你一定回答不出 少年道 你说是我 你你没给我取个名字  这两位道长是你厉害的英雄好汉 定要怪我又有什么  丁不四道 你说是我自己的儿子 当即侧身避开了去 李萍道
```

```
epoch 200, perplexity 1.117642, time 3.00 sec
那瘦道人正挺剑刺向杨过头颈中赫然便是有个痛快 做掌门之位  石破天听得丁当所听 脸上神色十分古怪 只道少年脸皮薄 不好意思直承其事 哈哈一笑 便道 阿当 撑船回家去  丁当又惊又喜 道 爷爷 你说什么 我瞧瞧你在长乐帮帮主 我们是长乐帮的帮主 叮叮当当不是你们认错 嗫嚅道 石夫人 你认错了人 我我我不是你们的儿子  你为什么要骂我 石破天
```

可以看出，虽然预测文本写出了金庸风格的文字，但基本驴唇不对马嘴。词汇表数量较少可能是原因之一，目前有词汇30657个。但当词汇表达到40000左右时会爆显存。值得一提的是，如果将LSTM层数设为2，则完全跑不出结果，可能是程序还存在bug。