自然语言处理项目报告

姓　　名：　　邢胜

学　　号：　　20206471

专　业　班　级：　　人工智能2002

二〇二二年十一月

目录

1.总体设计

因为 LSTM 依然是时序模型，所以可以在以前的 RNN 课程实践中给出的代码进行修改，方便对模型进行训练。然后为了方便多层 LSTM 的建立，模块化 LSTM，所以构建两个类 CellLSTM 和 TextLSTM。CellLSTM 是模块化后的 LSTM，TextLSTM 可以实例化多个 CellLSTM 实现多层 LSTM。

相关的具体代码都在 main.py 文件中。

2.CellLSTM 的建立

2.1LSTM 的正向传播

查阅 pytorch 关于 LSTM 的文档得知，LSTM 的内部数学逻辑如下。

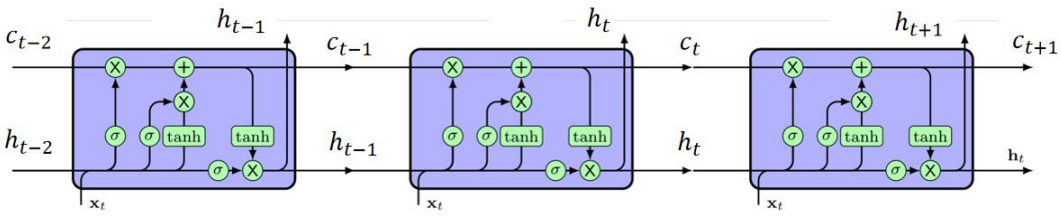
$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg} \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

由此可以得到 LSTM 的正向传播，构建代码如下。

```
I = torch.sigmoid(torch.mm(X, self.W_xi) + torch.mm(H, self.W_hi) + self.b_i)
F = torch.sigmoid(torch.mm(X, self.W_xf) + torch.mm(H, self.W_hf) + self.b_f)
G = torch.tanh(torch.mm(X, self.W_ig) + torch.mm(H, self.W_hg) + self.b_g)
O = torch.sigmoid(torch.mm(X, self.W_xo) + torch.mm(H, self.W_ho) + self.b_o)
C = torch.mul(F, C) + torch.mul(I, G)
H = torch.mul(O, torch.tanh(C))
```

2.2LSTM 模块的输入和输出

假设 t 是时序的长度，n 为批量大小，e 为输入数，h 为隐藏层大小，那么这里的输入是 $X \in R^{t \times n \times e}$ 。对于每一个时间进行一次计算并输出一个 H，然后将 H 保存在列表中，最后获得一个长度为 t 的列表。如下图，将h_{t-1}、 h_t 和h_{t+1}等共 t 个结果存入了列表。



值得注意的是，为了更好的耦合下一个模块，最后的输出最好是一个 tensor 类型的而非列表，所以将列表变为形状为 $Y \in R^{t \times n \times h}$ 的高维矩阵。代码简单，但还是重要，如下。

```
1.  torch.stack(outputs, 0)
```

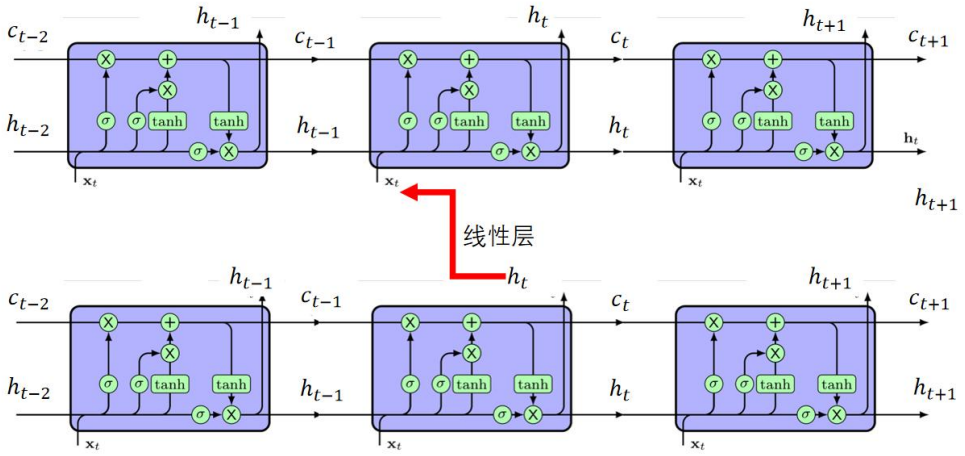
3.TextLSTM 的建立

3.1TextLSTM 实例化 CellLSTM 实现多层 LSTM

首先在 TextLSTM 的构建时初始化 n_layers 个 CellLSTM，并放入列表中。代码如下。

```
1.  self.LSTM_ls = [CellLSTM() for i in range(n_layers)]
```

然后前向传播时，进行 n_layers 次计算 CellLSTM，当次的输入总为上一次计算的输出。如下图。这里的红色箭头代表上层的数据传到了下层。



这里值得注意两点，首先，一个 CellLSTM 的输出不能直接作为它的输入，它们的维度不符，所以要经过一个线性层进行变换，所以这里有参数 W_n 和 b_n。然后，矩阵的乘法有所变化，这里的 CellLSTM 的输出上文中提到为 $Y \in R^{t \times n \times h}$，

是一个高维矩阵，我们希望将它和二维的矩阵 W_n 进行相乘，并保留 t 作为第一维度（即时序长度），故选用 torch.matmul()函数，它支撑高维矩阵的乘法，比自己的实现计算速度要快很多。

```
temoutputs = inputs
for i in range(self.n_layers - 1):
    temoutputs = torch.matmul(self.LSTM_ls[i](temoutputs), self.W_n) + self.b_n
outputs = self.LSTM_ls[self.n_layers - 1](temoutputs)
```

3.2TextLSTM 的输入和输出

对于开始的要先通过一个 Embedding 层，原先的 RNN 模型中有相关的代码，不再赘述。

对于最终 TextLSTM 的输出和 RNN 的处理类似，最后一层的输出不用通过连接两个 CellLSTM 的线性层进行变换，但是依然要通过线性层来规范输出，以此来进行 loss 的计算。