传统 LSTM

引入输入门 $\mathbf{i}_t$，遗忘门 $\mathbf{f}_t$ 和输出门 $\mathbf{o}_t$，以及新的 cell state $\mathbf{c}_t$，整体公式如下：

$$\begin{bmatrix} \mathbf{i}_t^c \\ \mathbf{o}_t^c \\ \mathbf{f}_t^c \\ \tilde{\mathbf{c}^c}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{bmatrix} \left(\mathbf{W}^{cT} \begin{bmatrix} \mathbf{x}_t^c \\ \mathbf{h}_{t-1}^c \end{bmatrix} + \mathbf{b}^c \right)$$

$$\begin{aligned} \mathbf{c}_t^c &= \mathbf{f}_t^c \odot \mathbf{c}_{t-1}^c + \mathbf{i}_t^c \odot \tilde{\mathbf{c}}_t^c \\ \mathbf{h}_t^c &= \mathbf{o}_t^c \odot \tanh(\mathbf{c_t^c}) \end{aligned}$$

这个是基于字符的计算流程。上标 c 表示 character，是基于字符的。

Lattice LSTM

整体思想是在 LSTM 的计算过程中，融入词汇的信息。为此，分为两步：

1. 生成词的 Cell State
2. 将词生成的 Cell State 和字的 Cell State 进行融合

Word LSTM Cell

该模块用于生成词的 Cell State，输入为词汇首字的 Hidden State 和 Cell State。例如在计算词汇 “南京市” 的 Cell State 的时候，需要输入 "南" 字的 Hidden State 和 Cell State。整体的计算流程如下：

$$\begin{bmatrix} \mathbf{i}_{b,e}^w \\ \mathbf{f}_{b,e}^w \\ \tilde{\mathbf{c}}_{b,e}^w \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ tanh \end{bmatrix} \left(\mathbf{W}^{wT} \begin{bmatrix} \mathbf{x}_{b,e}^w \\ \mathbf{h}_b^c \end{bmatrix} + \mathbf{b}^w \right)$$

$$\mathbf{c}_{b,e}^w = \mathbf{f}_{b,e}^w \odot \mathbf{c}_b^c + \mathbf{i}_{b,e}^w \odot \tilde{\mathbf{c}}_{b,e}^w$$

其中，上标 w 表示 word，c 表示 character；下标 b 表示 begin，e 表示 end。

$\mathbf{i}_{b,e}^w$ 和 $\mathbf{f}_{b,e}^w$ 分别是输入门和遗忘门。其中，$\mathbf{h}_b^c$ 和 $\mathbf{c}_b^c$ 分别是来自词的首字 LSTM 单元输出的 Hidden State 和 Cell State，比如计算词 “长江大桥” 的 Cell State，$\mathbf{h}_b^c$ 和 $\mathbf{c}_b^c$ 来自于“长” 字，而计算 “大桥” 的 Cell State，$\mathbf{h}_b^c$ 和 $\mathbf{c}_b^c$ 来自于 “大” 字。

以上对应的代码为 WordLSTMCell 类。

字词融合

在得到词的 Cell State 后，需要将字的 Cell State 和词的 Cell State 进行融合，融合的位置是在每个词的最后一个字。以 “长江大桥” 为例，假设词包括 “长江大桥” 和 “大桥”，则在计算 “桥” 字的 Cell State 的时候，需要对字符 “桥”、词 “长江大桥” 和词 “大桥” 的 Cell State 进行加权相加。具体如下：

计算字符 t_j^c 的 Cell State $\mathbf{c}_j^c$ ，需要对所有以字符 t_j^c 为结尾的词的 Cell State 和 t_j^c 的候选状态 $\tilde{\mathbf{c}}_j^c$ 进行加权相加。计算方法为：

$$\mathbf{c}_j^c = \sum_{b \in \{b' | w_{b',j}^d \in \mathbb{D}\}} \boldsymbol{\alpha}_{b,j}^c \odot \boldsymbol{c}_{b,j}^w + \boldsymbol{\alpha}_j^c \odot \tilde{\mathbf{c}}_j^c$$

其中，上标 w 表示 word，c 表示 character；下标 b 表示 begin，e 表示 end，j 为 b 至 e 之间的时刻。

这里加权系数的计算需要一个额外的门控 $\mathbf{i}_{b,e}^c$ ，根据当前词的 Cell State $\mathbf{c}_{b,e}^w$ 和字符嵌入 $\mathbf{x}_e^c$ 进行计算：

$$\mathbf{i}_{b,e}^c = \sigma\left(\mathbf{W}^{lT}\begin{bmatrix}\mathbf{x}_e^c \\ \mathbf{c}_{b,e}^w\end{bmatrix} + \mathbf{b}^l\right)$$

对于 “桥” 字，有 3 种向量：

- “桥” 字在 LSTM 基本单元中得到的 $(\mathbf{i}_j^c, \tilde{\mathbf{c}}_j^c)$
- 计算 “长江大桥” 得到的 $(\mathbf{i}_{b_1,e_1}^w, \mathbf{c}_{b1,e1}^w)$
- 计算 “大桥” 得到的 $(\mathbf{i}_{b_2,e_2}^w, \mathbf{c}_{b2,e2}^w)$

然后利用这 3 种向量的字符和词的输入门进行 $softmax$ 得到加权系数：

$$\boldsymbol{\alpha}_{b,j}^c = \frac{\exp(\mathbf{i}_{b,j}^c)}{\exp(\mathbf{i}_j^c) + \sum_{b' \in \{b'' | w_{b'',j}^d \in \mathbb{D}\}} \exp(\mathbf{i}_{b',j}^c)}$$

$$\boldsymbol{\alpha}_j^c = \frac{\exp(\mathbf{i}_j^c)}{\exp(\mathbf{i}_j^c) + \sum_{b' \in \{b'' | w_{b'',j}^d \in \mathbb{D}\}} \exp(\mathbf{i}_{b',j}^c)}$$

小细节

在具体的实现中，需要一个 “首字词集合” 和一个 “尾字词集合”。其中，首字词集合用于指导词的 Cell State 的计算，因为每个词在计算 Cell State 的时候需要用到首字信息，在 j 时刻需要计算出所有以 j 时刻的字符为开头的词的 Cell State，并将计算结果存入尾字词集合。最后，在对应的尾字部分，取出尾字词集合中以该字为结尾的词，然后进行加权相加，生成对应的 Cell State。

每个字符的 Cell State 和 Hidden State 计算由 MultiInputLSTMCell 类统一进行处理。对于每个字需要区分是否是结尾字，如果不是结尾字，同 LSTM 原本计算公式；如果是结尾字，在计算结尾字的 Cell State 时，需要计算对应的加权系数并融合所有以该字结尾的词汇的 Cell State。

缺点

1. 原始代码仅支持 batch_size 为 1，LSTM 结构并行化困难；如需扩大 batch_size，可参考 https://github.com/LeeSureman/Batch_Parallel_LatticeLSTM
2. 每个字符仅能获取到以它为结尾的词汇信息，如对于 “大” 这个字符，无法获得 “长江大桥” 的词汇信息
3. 生成词的 Cell State 的时候只使用了首字的 Hidden State 和 Cell State 信息，但是词本身的含义难以仅通过首字来表达，会存在信息损失