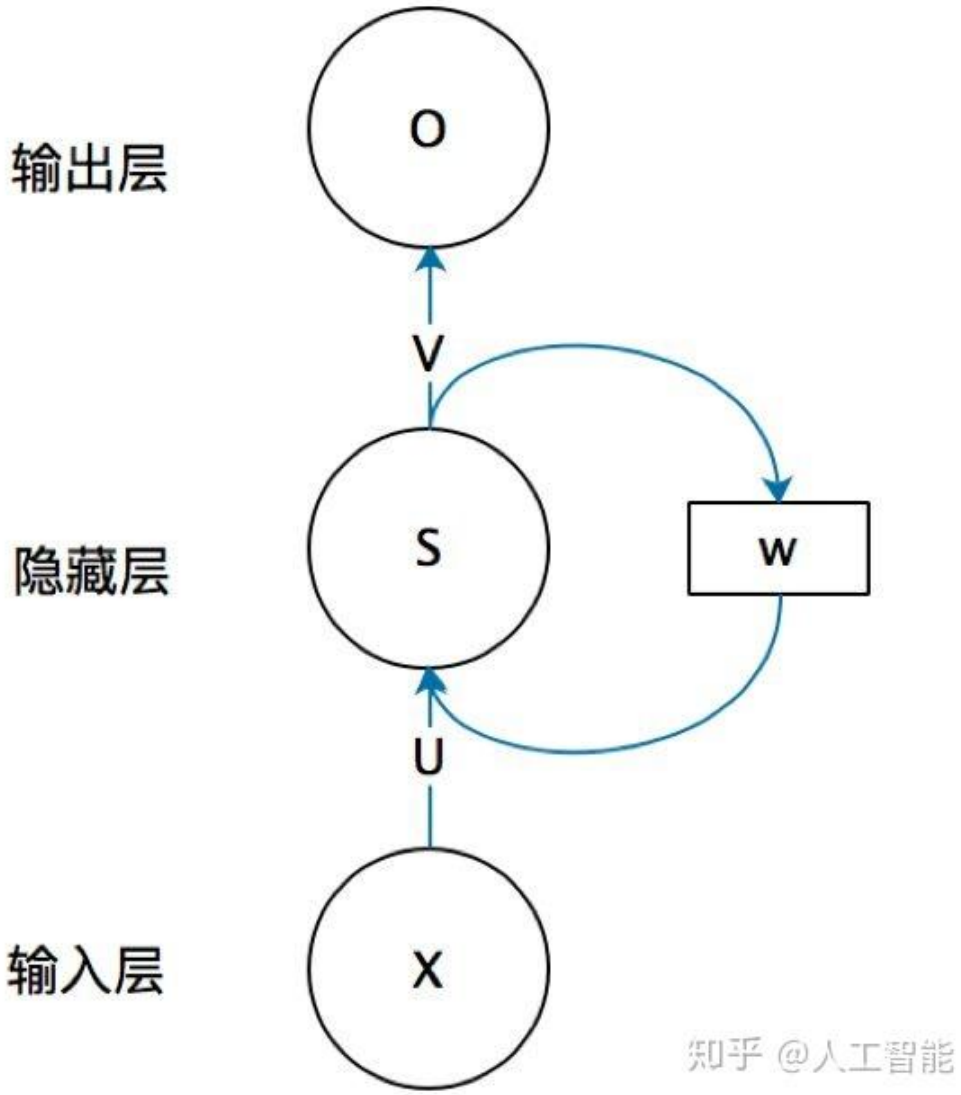
诗歌生成作业报告

2152199 余苏皓

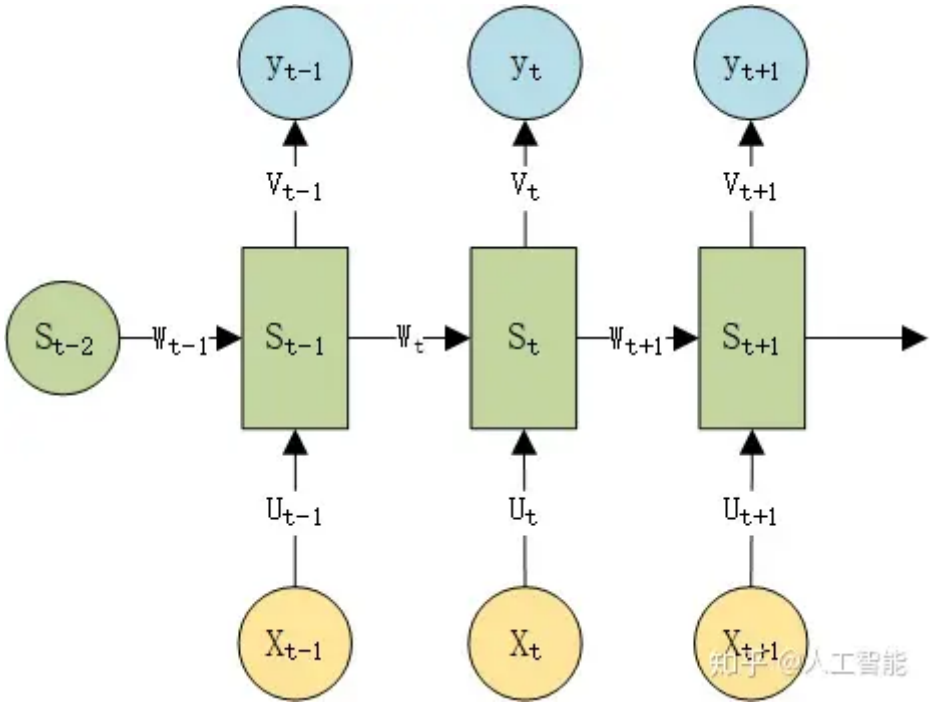
RNN模型

RNN结构：

RNN的结构是由一个输入层、隐藏层、输出层组成：



将RNN的结构按照时间序列展开

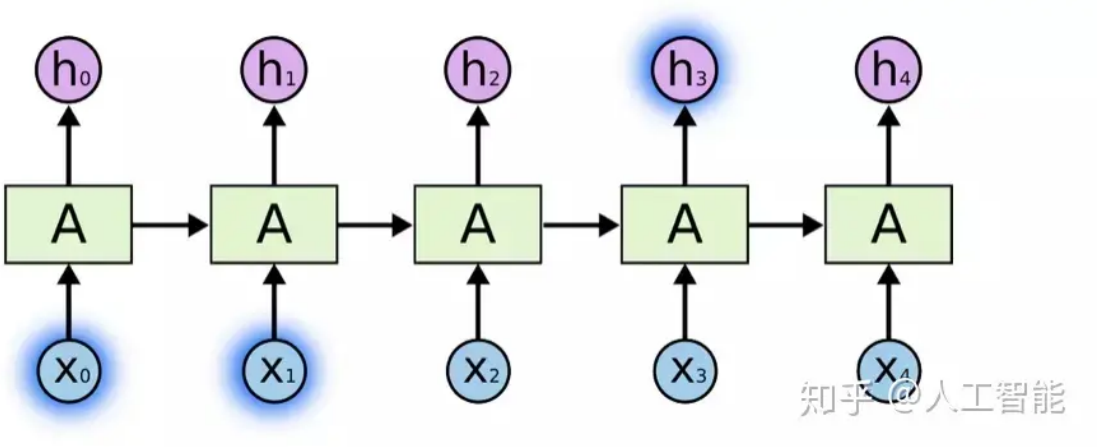


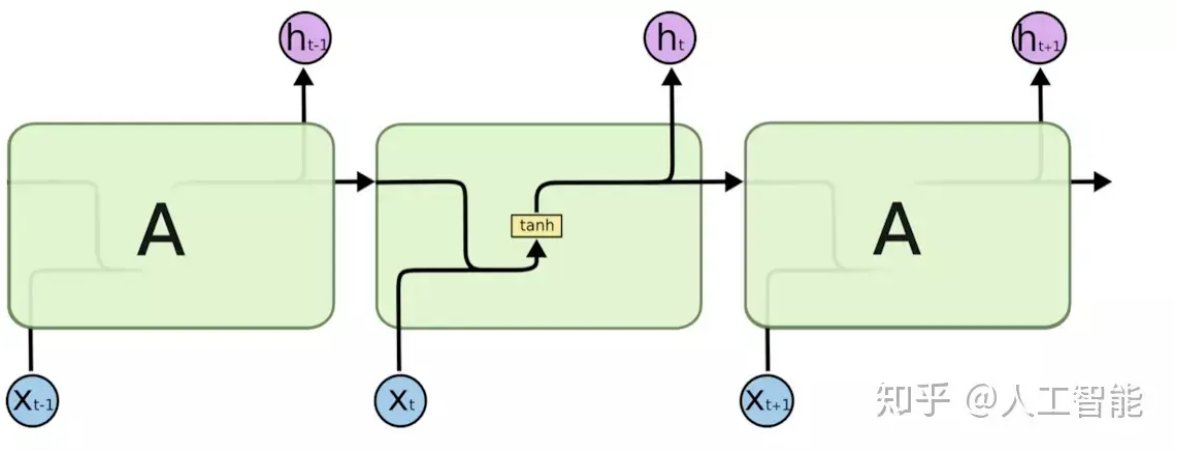
对应的前向传播公式和对应的每个时刻的输出公式

$$S_{t-1} = U_{t-1}X_{t-1} + W_{t-1}S_{t-2} + b_1 \quad y_{t-1} = V_{t-1}S_{t-1} + b_2$$
$$S_t = U_tX_t + W_tS_{t-1} + b_1 \quad y_{t-1} = V_tS_t + b_2$$
$$S_{t+1} = U_{t+1}X_{t+1} + W_{t+1}S_t + b_1 \quad y_{t+1} = V_{t+1}S_{t+1} + b_2$$

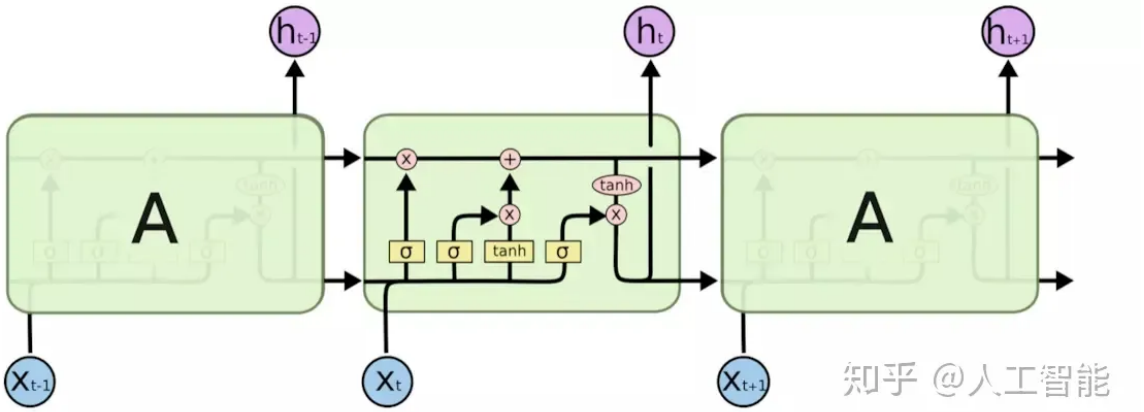
LSTM（Long Short-Term Memory，长短期记忆网络）

LSTM是一种特殊的RNN类型，一般的RNN结构如下图所示，是一种将以往学习的结果应用到当前学习的模型，但是这种一般的RNN存在着许多的弊端。如果相关的信息和预测的词位置之间的间隔是非常小，RNN 可以学会使用先前的信息。

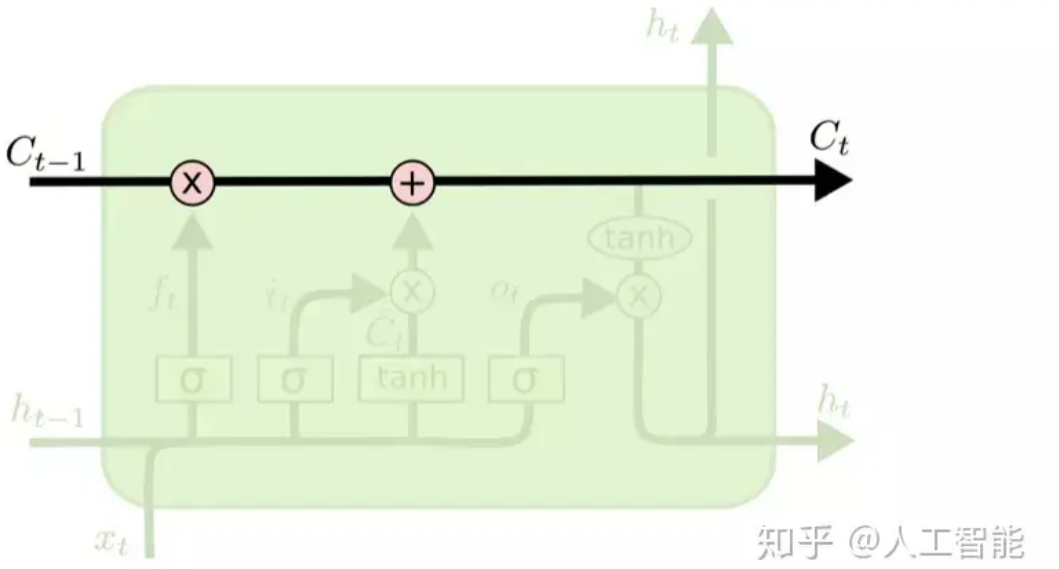




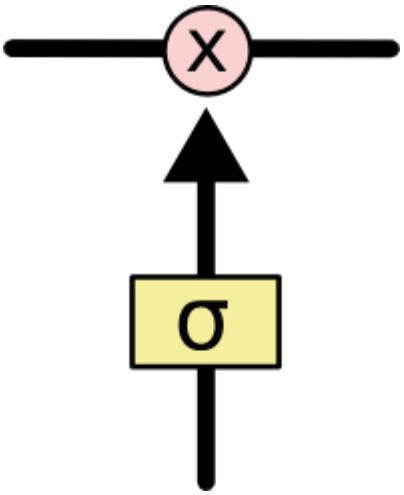
但在比较长的环境中，RNN 并不能够成功学习到这些知识。然而，LSTM模型就可以解决这一问题.



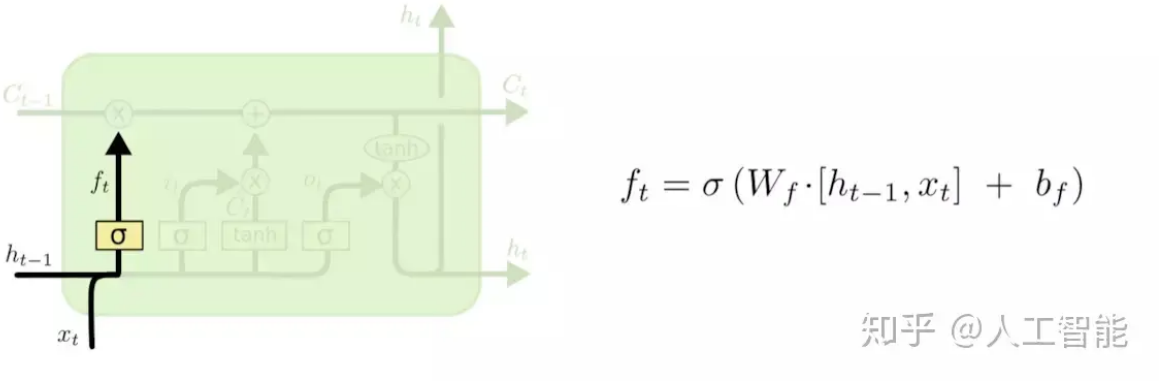
如图所示，标准LSTM模型是一种特殊的RNN类型，在每一个重复的模块中有四个特殊的结构，以一种特殊的方式进行交互。在图中，每一条黑线传输着一整个向量，粉色的圈代表一种pointwise 操作(将定义域上的每一点的函数值分别进行运算)，诸如向量的和，而黄色的矩阵就是学习到的神经网络层。
LSTM模型的核心思想是"细胞状态"。"细胞状态"类似于传送带。直接在整个链上运行，只有一些少量的线性交互。信息在上面流传保持不变会很容易。



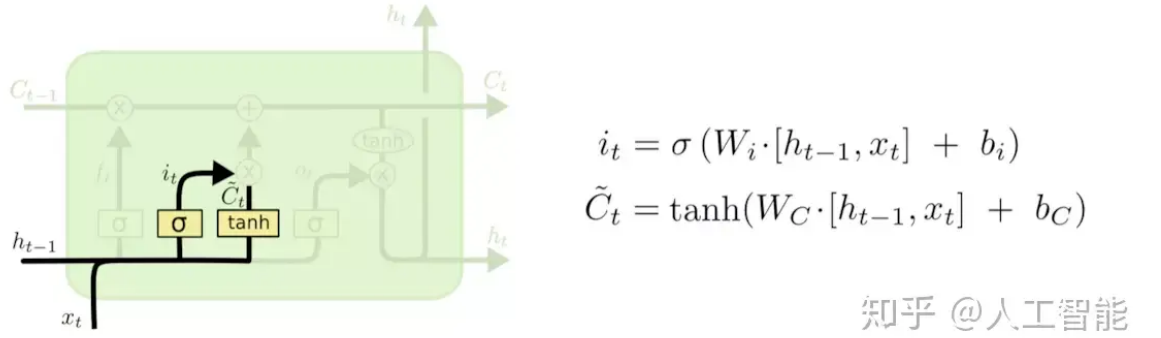
LSTM 有通过精心设计的称作为"门"的结构来去除或者增加信息到细胞状态的能力。门是一种让信息选择式通过的方法。他们包含一个 sigmoid 神经网络层和一个 pointwise 乘法操作。



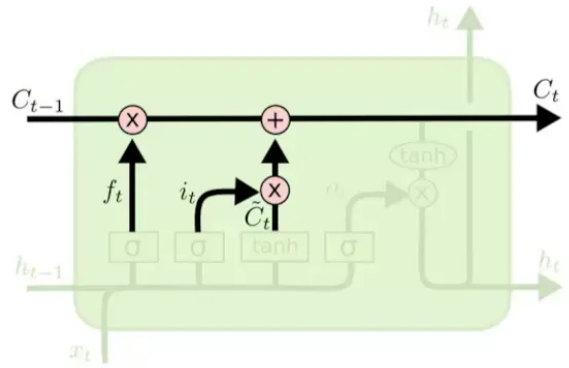
Sigmoid 层输出 0 到 1 之间的数值，描述每个部分有多少量可以通过。0 代表“不许任何量通过”，1 就指“允许任意量通过”。LSTM 拥有三个门，来保护和控制细胞状态。



在LSTM模型中，第一步是决定我们从“细胞”中丢弃什么信息，这个操作由一个忘记门层来完成。该层读取当前输入x和前神经元信息h，由ft来决定丢弃的信息。输出结果1表示“完全保留”，0 表示“完全舍弃”。

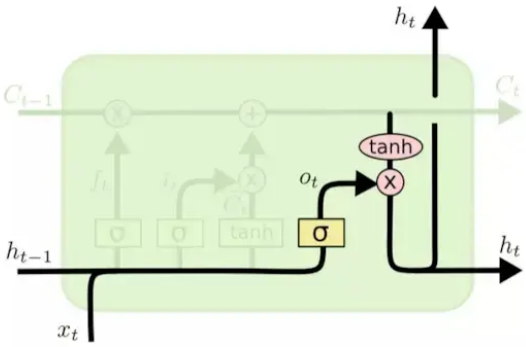


第二步是确定细胞状态所存放的新信息，这一步由两层组成。sigmoid层作为“输入门层”，决定我们将要更新的值i；tanh层来创建一个新的候选值向量$\widetilde{C}_t$ 加入到状态中。在语言模型的例子中，我们希望增加新的主语到细胞状态中，来替代旧的需要忘记的主语。



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

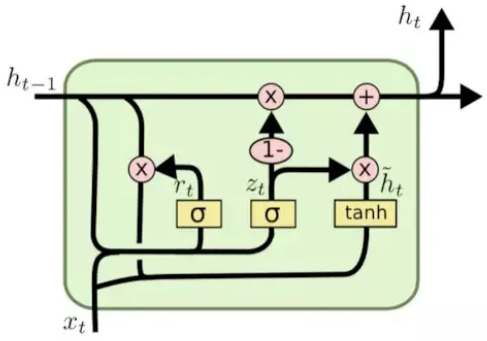
第三步就是更新旧细胞的状态，将C_{t-1}更新为 C_t 。我们把旧状态与 f_t 相乘，丢弃掉我们确定需要丢弃的信息。接着加上$i_t * \widetilde{C}_{t-1}$。这就是新的候选值，根据我们决定更新每个状态的程度进行变化。在语言模型的例子中，这就是我们实际根据前面确定的目标，丢弃旧代词的信息并添加新的信息的地方



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

最后一步就是确定输出了，这个输出将会基于我们的细胞状态，但是也是一个过滤后的版本。首先，我们运行一个 sigmoid 层来确定细胞状态的哪个部分将输出出去。接着，我们把细胞状态通过 tanh 进行处理（得到一个在 -1 到 1 之间的值）并将它和 sigmoid 门的输出相乘，最终我们仅仅会输出我们确定输出的那部分。在语言模型的例子中，因为语境中有一个代词，可能需要输出与之相关的信息。例如，输出判断是一个动词，那么我们需要根据代词是单数还是负数，进行动词的词形变化。

GRU（Gated Recurrent Unit, LSTM变体）



$$z_t = \sigma\left(W_z \cdot \left[h_{t-1}, x_t\right]\right)$$
$$r_t = \sigma\left(W_r \cdot \left[h_{t-1}, x_t\right]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot \left[r_t * h_{t-1}, x_t\right]\right)$$
$$h_t = \left(1 - z_t\right) * h_{t-1} + z_t * \tilde{h}_t$$

GRU作为LSTM的一种变体，将忘记门和输入门合成了一个单一的更新门。同样还混合了细胞状态和隐藏状态，加诸其他一些改动。最终的模型比标准的 LSTM 模型要简单，也是非常流行的变体。

诗歌生成过程

以tensorflow版本为例

初始化状态

- `state` 初始化为两个随机正态分布的向量，每个向量的形状为 `(1, 128)`，标准差为 `0.5`。这些向量代表了模型的初始隐藏状态。
- `cur_token` 初始化为 `bos`（beginning of sentence，句子开头）标记的ID，这是生成过程的起点。

生成循环

- 通过一个最多迭代50次的循环来生成词汇。每次迭代尝试添加一个新的词汇到句子中。

生成下一个词

- 在每次迭代中，当前的词汇 `cur_token` 和当前的状态 `state` 被传递给 `get_next_token` 函数，以预测下一个词的ID并更新状态。
- `get_next_token` 函数内部，当前词汇通过嵌入层转换成词嵌入向量。然后，这个嵌入向量和当前的状态一起被传递给RNN单元（`rnncell.call`），产生新的状态和一个输出向量 `h`。
- 输出向量 `h` 通过一个全连接层（`dense`）转换成对词汇表中每个词的打分（`logits`）。
- 使用 `tf.argmax` 从这些打分中选择最高的，作为下一个词的ID。

构建句子

- 如果生成的词是 `eos`（end of sentence，句子结束）标记，则继续下一次迭代而不将其添加到收集器 `collect` 中。这是因为你希望生成的句子在遇到句子结束标记前不停止。
- 如果生成的词不是 `eos`，则将其ID添加到 `collect` 列表中。
- 迭代完成后，通过 `collect` 列表中的ID，使用 `id2word` 字典将每个词的ID转换成对应的词。
- 将这些词连接成一个字符串，形成最终的生成文本。

输出

- 打印出通过上述过程生成的文本。

这个过程展示了一个简化的诗歌生成模型，其中使用了RNN和词嵌入来根据当前状态和已生成的词序列预测下一个最可能的词。

训练截图

输出 调试控制台 问题 7 端口 GITLENS 终端 JUPYTER

```
b_y        [29, 23, 544, 17, 102, 216, 102, 0, 68, 23, 544, 17, 182, 216, 182, 1, 266, 1137, 1527, 63, 15, 37, 565, 0, 107, 193, 220, 35, 27, 542, 197, 1, 5,
 35, 43, 219, 172, 143, 20, 0, 1075, 23, 382, 598, 165, 47, 407, 1, 378, 317, 250, 152, 499, 56, 341, 0, 175, 1899, 702, 6, 799, 54, 310, 1, 3, 3]
******************************
epoch   6 batch number 248 loss is:  6.3922810554504395
prediction [22, 23, 4, 18, 13, 165, 48, 0, 10, 21, 9, 14, 36, 137, 329, 1, 22, 137, 4, 93, 9, 27, 41, 0, 10, 11, 9, 23, 10, 26, 339, 1, 11, 22, 16, 115, 69,
115, 17, 0, 13, 7, 69, 22, 10, 237, 1, 1, 3, 17, 84, 129, 55, 5, 29, 0, 10, 12, 69, 18, 10, 30, 14, 1, 3, 3]
b_y        [488, 5, 110, 448, 36, 11, 29, 0, 39, 114, 40, 455, 167, 245, 127, 1, 167, 245, 10, 29, 4, 179, 1065, 0, 36, 11, 69, 586, 40, 339, 339, 1, 297, 33
0, 865, 865, 194, 115, 72, 0, 13, 74, 1326, 1326, 1749, 1842, 512, 1, 8, 169, 185, 183, 15, 42, 49, 0, 75, 224, 33, 18, 348, 5, 105, 1, 3, 3]
******************************
epoch   6 batch number 249 loss is:  6.198606967926025
prediction [22, 114, 10, 5, 5, 0, 0, 0, 10, 62, 9, 19, 4, 4, 27, 1, 36, 137, 4, 30, 4, 22, 64, 0, 16, 11, 10, 48, 77, 23, 27, 1, 167, 250, 4, 21, 9, 4, 29, 0
, 11, 47, 28, 96, 77, 26, 41, 1, 3, 125, 4, 114, 15, 15, 145, 0, 4, 19, 66, 23, 4, 72, 13, 1, 3, 3]
b_y        [326, 71, 122, 14, 705, 2236, 2348, 0, 194, 62, 92, 134, 67, 4, 188, 1, 133, 721, 1323, 30, 43, 15, 42, 0, 60, 127, 328, 34, 28, 144, 218, 1, 28,
303, 204, 21, 396, 334, 14, 0, 149, 278, 146, 351, 1756, 26, 602, 1, 450, 1693, 39, 64, 68, 289, 792, 0, 494, 19, 23, 23, 287, 72, 923, 1, 3, 3]
******************************
epoch   6 batch number 250 loss is:  6.274394989013672
prediction [10, 27, 66, 12, 79, 179, 132, 0, 10, 19, 13, 17, 77, 84, 164, 1, 36, 5, 4, 58, 9, 47, 64, 0, 22, 41, 15, 93, 4, 54, 27, 1, 36, 159, 12, 108, 13,
30, 16, 0, 167, 96, 15, 11, 10, 126, 11, 1, 15, 19, 84, 7, 35, 30, 29, 0, 4, 37, 12, 72, 10, 21, 211, 1, 3, 3]
b_y        [110, 41, 1945, 12, 87, 1259, 186, 0, 101, 313, 176, 245, 43, 5, 446, 1, 362, 279, 91, 219, 32, 9, 64, 0, 248, 34, 151, 116, 399, 31, 1119, 1, 292
, 196, 383, 760, 109, 88, 16, 0, 45, 327, 176, 242, 821, 36, 11, 1, 92, 545, 46, 736, 1970, 234, 42, 0, 101, 98, 154, 18, 100, 627, 627, 1, 3, 3]
******************************
epoch   6 batch number 251 loss is:  6.406141757965088
prediction [10, 115, 9, 41, 30, 275, 14, 0, 10, 21, 15, 64, 4, 23, 38, 1, 11, 7, 4, 12, 12, 12, 16, 0, 36, 19, 33, 185, 4, 64, 14, 1, 36, 104, 4, 18, 9, 88,
29, 0, 4, 208, 13, 12, 4, 54, 94, 1, 15, 256, 84, 7, 9, 96, 114, 0, 4, 131, 36, 95, 4, 13, 32, 1, 3, 3]
b_y        [11, 44, 4, 109, 268, 252, 139, 0, 1211, 21, 1418, 41, 10, 432, 76, 1, 25, 7, 1111, 1111, 243, 33, 194, 0, 251, 43, 453, 185, 15, 42, 5, 1, 388, 2
50, 239, 1599, 7, 89, 177, 0, 576, 618, 214, 162, 100, 99, 94, 1, 908, 908, 84, 6, 607, 194, 1641, 0, 1252, 1252, 777, 153, 317, 208, 24, 1, 3, 3]
******************************
epoch   6 batch number 252 loss is:  6.381404876708984
prediction [22, 480, 4, 18, 13, 275, 45, 0, 33, 23, 4, 29, 77, 5, 134, 1, 11, 7, 4, 41, 5, 12, 45, 0, 33, 45, 15, 9, 4, 437, 32, 1, 36, 85, 16, 21, 12, 100,
41, 0, 36, 22, 10, 48, 77, 5, 14, 1, 3, 7, 4, 117, 9, 5, 64, 0, 4, 19, 36, 104, 4, 34, 24, 1, 3, 3]
b_y        [3093, 408, 266, 58, 362, 61, 215, 0, 234, 38, 775, 213, 19, 954, 7, 1, 116, 505, 4, 383, 1016, 1885, 378, 0, 426, 545, 617, 1040, 1395, 86, 301,
1, 805, 145, 791, 75, 35, 4, 929, 0, 297, 14, 636, 85, 801, 24, 40, 1, 699, 518, 77, 682, 9, 455, 64, 0, 263, 112, 22, 12, 10, 288, 113, 1, 3, 3]
******************************
epoch   6 batch number 253 loss is:  6.2898173332214355
prediction [22, 23, 4, 29, 4, 26, 125, 0, 4, 17, 9, 5, 4, 4, 79, 1, 36, 115, 16, 5, 4, 96, 16, 0, 16, 119, 12, 95, 4, 5, 14, 1, 36, 41, 7, 126, 9, 5, 45, 0,
4, 41, 12, 94, 91, 10, 23, 1, 4, 19, 4, 129, 15, 54, 41, 0, 15, 7, 4, 108, 4, 47, 94, 1, 3, 3]
b_y        [190, 23, 24, 45, 693, 32, 423, 0, 80, 393, 9, 5, 110, 31, 414, 1, 269, 271, 368, 162, 478, 75, 17, 0, 736, 1085, 1190, 550, 349, 134, 1104, 1, 76
9, 231, 2040, 966, 181, 1193, 74, 0, 2192, 644, 915, 748, 753, 1468, 38, 1, 851, 19, 1475, 129, 138, 54, 878, 0, 13, 7, 392, 114, 8, 40, 28, 1, 3, 3]
******************************
epoch   6 batch number 254 loss is:  6.292551040649414
```

```
error
inital  linear weight
日夜微露，微微不复晓。
云间千树发，花发见春生。
error
inital  linear weight
红叶发，衣上东南池。
一夜春风晚，春风白石间。
error
inital  linear weight
山云外，微微入夜风。
不知天下地，不见白云心。
何事安清夜，无人不。
error
inital  linear weight
夜后微微，微风不相见。
夜来夜夜千家外，一夜风光入夜风。
error
inital  linear weight
湖微发发时。
云外千年在，云间不见人。
error
inital  linear weight
海光明，金石见清风。
不知天下地，何事不成行。
error
inital  linear weight
月，清露发庭光。
夜来衣上苑，夜夜见云明。
```

一日无人事，何人不可知。来不可见，不得不知君。客无人事，无人不可知。来无限处，不得是君人。客

实验总结

在本次实验中，我们分别使用TensorFlow和PyTorch框架实现了基于RNN和LSTM的模型，旨在解决诗歌生成和加法进位两个问题。TensorFlow实现的是一个基本的循环神经网络（RNN），而PyTorch实现则采用了长短期记忆网络（LSTM）。

对于诗歌生成任务，RNN模型能够捕捉到文本数据中的时间序列依赖性，生成具有一定韵律和意境的文本片段。然而，由于RNN的梯度消失问题，模型在捕捉长距离依赖时表现不佳。相比之下，LSTM模型通过引入遗忘门、输入门和输出门，有效地解决了梯度消失问题，表现出更好的长距离序列学习能力，从而能生成更加连贯和有深度的诗歌文本。

总体而言，尽管RNN在某些情况下能够给出满意的结果，LSTM由于其结构上的改进，无论是在理解复杂的语言模式还是处理具有复杂依赖关系的数学问题上，都显示出更强大的性能和更广泛的适用性。这次实验强调了在处理具有长距离依赖特性的序列数据时，选择合适的模型架构的重要性。