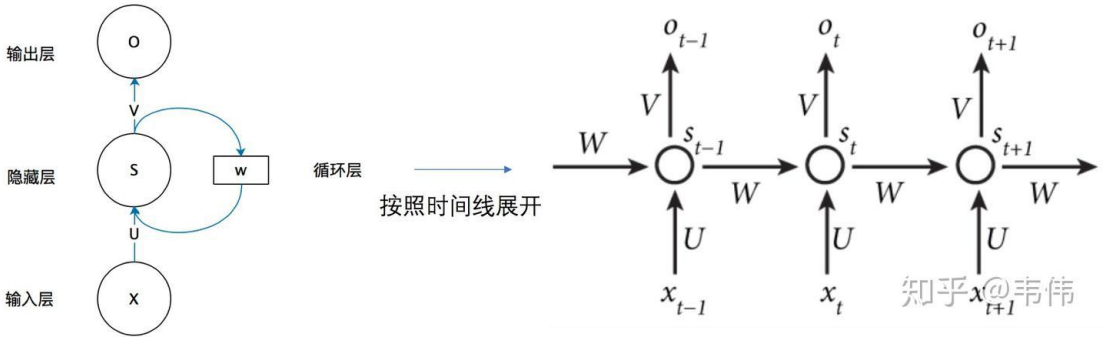
循环神经网络实验报告

RNN：

RNN 用于处理序列数据。一个序列当前的输出与前面的输出也有关。具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。理论上，RNN 能够对任何长度的序列数据进行处理。



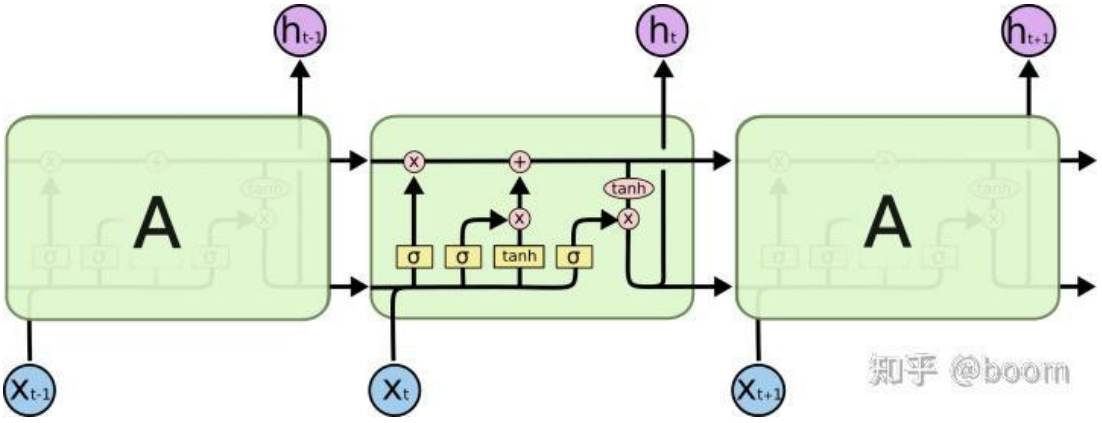
上图中的 Xt-1 代表的就是输入序列 t-1 时刻的向量， Xt 代表的是输入序列 t 时刻的向量，以此类推。上图展开后，W 一直没有变，W 其实是每个时间点之间的权重矩阵。该网络可以记住每一时刻的信息，每一时刻的隐藏层不仅由该时刻的输入层决定，还由上一时刻的隐藏层决定，公式如下，其中 Ot 代表 t 时刻的输出， St 代表 t 时刻的隐藏层的值：

$$O_t = g(V \cdot S_t)$$
$$S_t = f(U \cdot X_t + W \cdot S_{t-1})$$

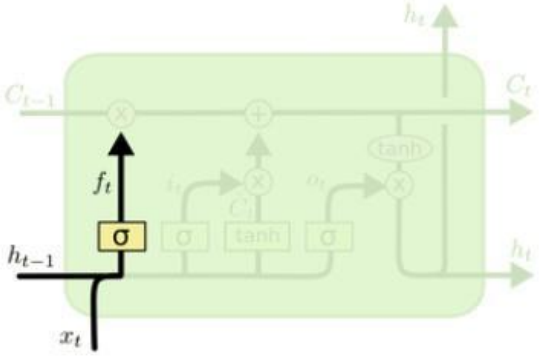
St的值不仅仅取决于Xt，还取决于St-1

LSTM：

LSTM 模型用于解决原始 RNN 网络无法处理长距离依赖的问题。相比于 RNN 的一个传递状态 h_t，LSTM 有两个传输状态，一个为 C^t，另一个为 h^t，其中 C^t 的变化很慢，而 h^t 在不同序列位置则变化很大。可以认为 C^t 为解决长距离依赖做出贡献，而 h^t 则-是类似于 RNN 用于记住短距离的序列信息。如下图为 LSTM 的结构，整体结构也是为链式结构，内部结构加上了‘遗忘门’、‘输入门’、‘输出门’。

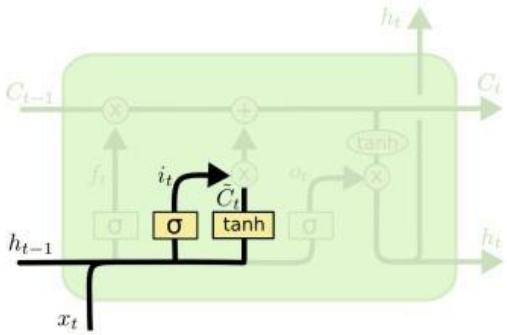


遗忘门：LSTM 内部结构第一个是遗忘门，决定将上一个位置的输出信息哪些该被遗忘。主要为 sigmoid 结构，这一层的输入为上一个位置的输出 h_{t-1} 和这一个位置的输入 x_t 做连接，再经过线性变换，最后通过 sigmoid 选择哪些信息被遗忘。



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

输入门：LSTM 内部结构第二个是输入门。主要由两个部分组成，输入同样是由上一个位置的输出 h_{t-1} 和这一个位置的输入 x_t 做连接，再做线性变换，第一个部分更新信息，第二个决定哪些部分需要被更新。首先是 sigmoid 层，决定哪些数值需要被更新，其次 tanh 层来对输入信息做更新。

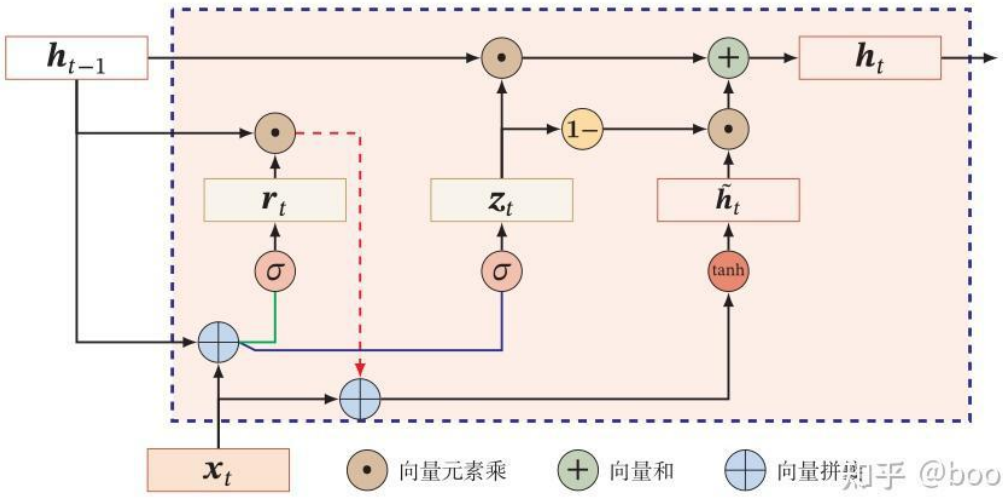


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

输出门：LSTM 内部的最后一个结构是输出门，首先通过 sigmoid 层来决定哪些信息需要被输出，然后让神经元状态乘 tanh 来对神经元状态信息进行更新，最后相乘后得到想要的输出部分。

GRU：

GRU 网络的结构图如图一所示。主要由更新门和重置门组成:



诗歌生成过程：

完成版本为 pytorch。

数据处理（process_poems1）：

读取 poem.txt 的每一行（即一首诗），先移除首行的空格，用冒号分割标题和内容。对内容去除所有空格，忽略无关字符、过长或过短的诗，在诗歌的前后加上标记。按诗的字数降序排序。然后统计所有诗歌中每个字出现次数，按照词频排序，给每一个词分配一个 id，返回描述每一个诗的单词的向量(poems_vector)，一个单词表 word_int_map，所有单词 words。

批处理(generate_batch)：

按 batch_size 大小对原来的 poems_vector 进行分隔，同时将每个 poems_vetor 的第一个单词去除，最后一个单词重复一次作为预测值返回。

模型构建：

嵌入层：rnn_lstm.word_embedding，参数是词汇表的长度和嵌入层维度。

RNN 层：self.rnn_lstm=nn.LSTM，参数是嵌入层输入，隐藏层的维度和层数。

全连接层：self.fc=nn.Linear，参数是 RNN 层输入和单词表长度，也就是预测得到的下一个单词。

输出层：self.soft=nn.LogSoftmax 利用 logSofmax 进行概率估计

训练过程：

在数据处理和模型初始化完成后，先对优化器 optim.RMSprop 和损失函数 torch.nn.NLLLoss()初始化。在每轮训练中，取出一个批次的数据，对于每一对数据，模型接受一个序列 x 预测下一个单词。计算损失值，进行反向传播更新参数值。

训练结果：

实验输出由 gen_poem 生成，它读取已经训练好的数据文件生成结果，由 pretty_print_poem 做美化处理（如去除前后的标记，添加句号和分隔每首诗）。

训练过程和结果如下：

```
******************************
epoch   0 batch number 6 loss is:  7.2312469482421875
prediction [31, 31, 137, 137, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
b_y        [18, 517, 56, 17, 191, 0, 1552, 1231, 88, 298, 61, 1, 57, 252, 311, 129, 259, 0, 397, 402, 412, 178, 347, 1, 3, 3]
******************************
epoch   0 batch number 7 loss is:  8.530468940734863
prediction [99, 114, 137, 137, 0, 0, 0, 43, 43, 43, 43, 43, 99, 99, 43, 43, 3, 3, 3, 3, 43, 43, 43, 43, 43, 43]
b_y        [3049, 18, 738, 137, 60, 0, 60, 127, 375, 36, 8, 1, 742, 20, 236, 117, 30, 0, 331, 108, 71, 21, 91, 1, 3, 3]
******************************
epoch   0 batch number 8 loss is:  7.104558944702148
prediction [6, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 0, 0, 3, 0, 0, 3, 3, 3, 0, 3, 3]
b_y        [216, 16, 296, 57, 12, 0, 12, 88, 84, 4, 612, 1, 67, 41, 1838, 1198, 5, 0, 37, 95, 28, 26, 244, 1, 3, 3]
******************************
epoch   0 batch number 9 loss is:  6.743811130523682
prediction [6, 6, 6, 36, 0, 0, 0, 0, 0, 1, 1, 1, 3, 3, 1, 1, 3, 3, 1, 1, 1, 3, 3, 3, 3, 3]
b_y        [279, 573, 114, 421, 974, 0, 487, 104, 469, 871, 1210, 1, 10, 14, 226, 212, 3405, 0, 31, 49, 98, 6, 12, 1, 3, 3]
******************************
epoch   0 batch number 10 loss is:  6.671166896820068
prediction [295, 52, 52, 52, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3]
b_y        [551, 486, 14, 26, 701, 0, 558, 956, 2027, 161, 24, 1, 92, 278, 277, 186, 469, 0, 12, 44, 685, 802, 134, 1, 3, 3]
******************************
epoch   0 batch number 11 loss is:  6.5447587966918945
prediction [66, 66, 35, 91, 0, 0, 0, 0, 0, 0, 1, 1, 1, 3, 3, 3, 3, 3, 3, 3, 1, 1, 3, 3, 3, 3]
b_y        [1538, 2174, 175, 426, 2396, 0, 107, 874, 2396, 152, 72, 1, 40, 6, 393, 99, 383, 0, 799, 19, 5, 27, 42, 1, 3, 3]
******************************
epoch   0 batch number 12 loss is:  6.634427547454834
prediction [6, 6, 6, 6, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
b_y        [90, 8, 1556, 83, 3054, 0, 180, 7, 447, 433, 134, 1, 18, 71, 1323, 2329, 309, 0, 950, 108, 1999, 936, 94, 1, 3, 3]
******************************
epoch   0 batch number 13 loss is:  6.730191230773926
prediction [158, 38, 121, 63, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
b_y        [2067, 116, 338, 44, 5, 0, 609, 1427, 579, 295, 13, 1, 569, 39, 283, 157, 425, 0, 138, 2400, 124, 285, 121, 1, 3, 3]
******************************
epoch   0 batch number 14 loss is:  6.651360988616943
```

```
  return self._call_impl(*args, **kwargs)
日月明光。清风生，春风起，汉水清光。今日有时，如此何为。
inital  linear weight
红树成。高台日月望，北风生白头。春风不可见，一望水中郎。
inital  linear weight
山西水，夜坐春风生。风景千里去，风前不可见。
inital  linear weight
夜长光。
inital  linear weight
湖山中，香风生白发。清风不可见，不复见清风。
inital  linear weight
君不见，上将心中不相见。
inital  linear weight
星命，上马上南山。山水不可见，风光不可见。谁能不得意，不复见花时。
```

实验总结：

在本次实验中我验证 RNN 处理序列数据的有效性。对比 RNN 与传统神经网络在序列任务中的性能差异。了解了 RNN 的变体 LSTM、GRU 的模型结构和工作原理，在实现中学习了 Pytorch 中 LTSM 模型的构件方法，学习了一种嵌入层的数据处理方法以及批处理方法，并且学会了如何训练模型和调整学习率、层数等超参数。