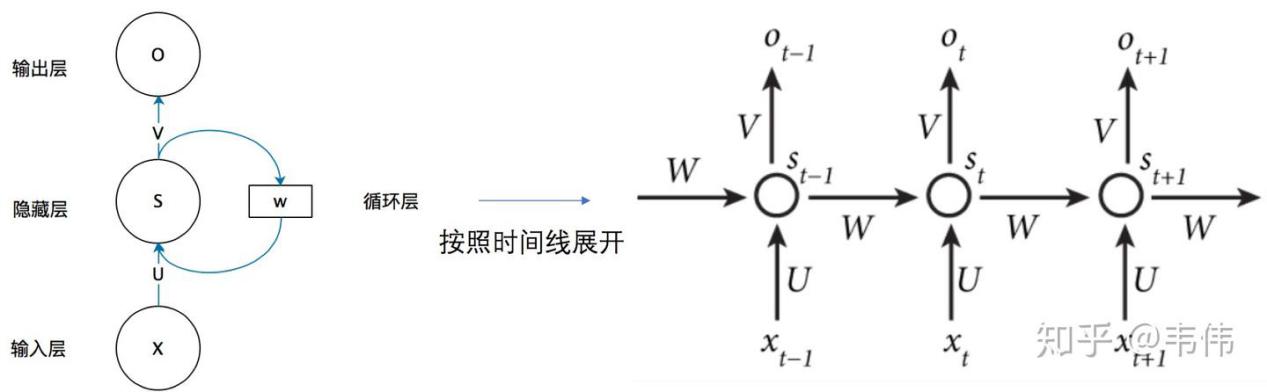
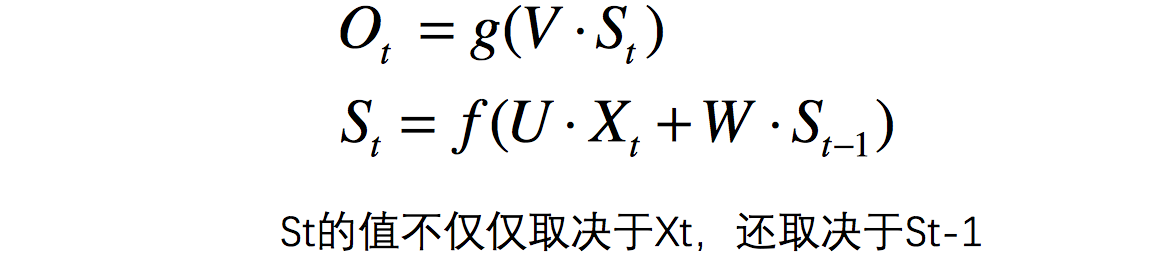
# **循环神经网络实验报告**

## RNN：

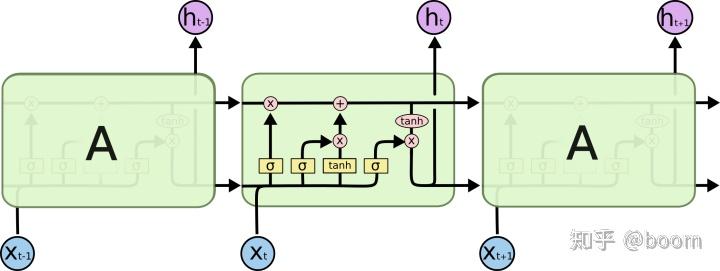
RNN用于处理序列数据。一个序列当前的输出与前面的输出也有关。具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。理论上，RNN能够对任何长度的序列数据进行处理。



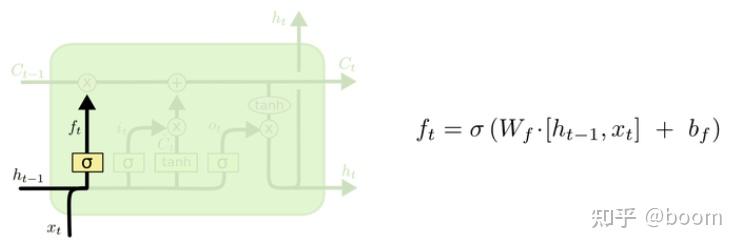
上图中的 Xt−1 代表的就是输入序列t-1时刻的向量， Xt 代表的是输入序列t时刻的向量，以此类推。上图展开后，W一直没有变，W其实是每个时间点之间的权重矩阵。该网络可以记住每一时刻的信息，每一时刻的隐藏层不仅由该时刻的输入层决定，还由上一时刻的隐藏层决定，公式如下，其中 Ot 代表t时刻的输出, St 代表t时刻的隐藏层的值：

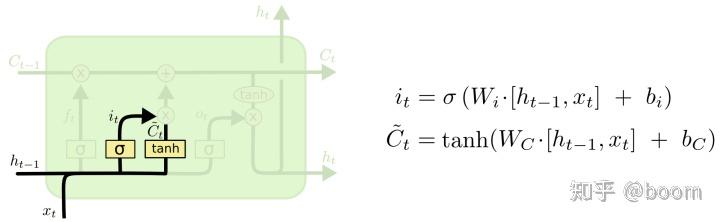


## LSTM：

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ht，LSTM有两个传输状态，一个为Ct,另一个为 ht，其中Ct的变化很慢，而ht在不同序列位置则变化很大。可以认为Ct为解决长距离依赖做出贡献，而ht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](https://zhida.zhihu.com/search?content_id=205034070&content_type=Article&match_order=1&q=sigmoid&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)结构，这一层的输入为上一个位置的输出ht-1和这一个位置的输入xt做连接，再经过[线性变换](https://zhida.zhihu.com/search?content_id=205034070&content_type=Article&match_order=1&q=%E7%BA%BF%E6%80%A7%E5%8F%98%E6%8D%A2&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)，最后通过sigmoid选择哪些信息被遗忘。

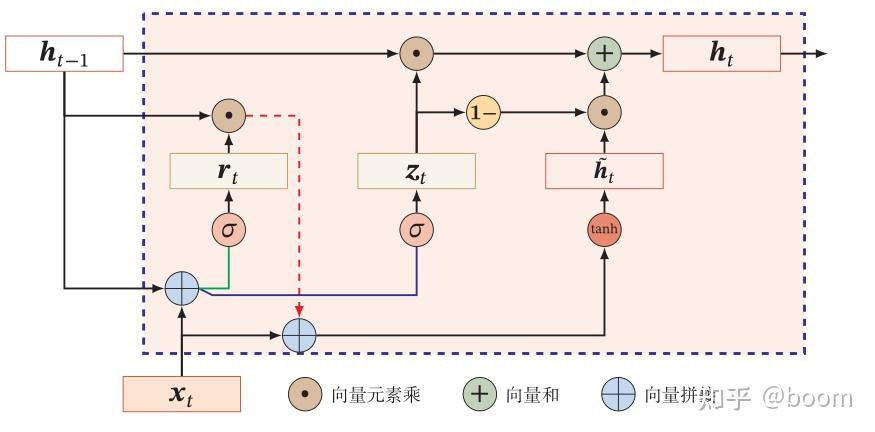


**输入门：**LSTM内部结构第二个是输入门。主要由两个部分组成，输入同样是由上一个位置的输出 ht-1和这一个位置的输入 xt做连接，再做线性变换，第一个部分更新信息，第二个决定哪些部分需要被更新。首先是sigmoid层，决定哪些数值需要被更新，其次[tanh层](https://zhida.zhihu.com/search?content_id=205034070&content_type=Article&match_order=1&q=tanh%E5%B1%82&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)来对输入信息做更新。

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

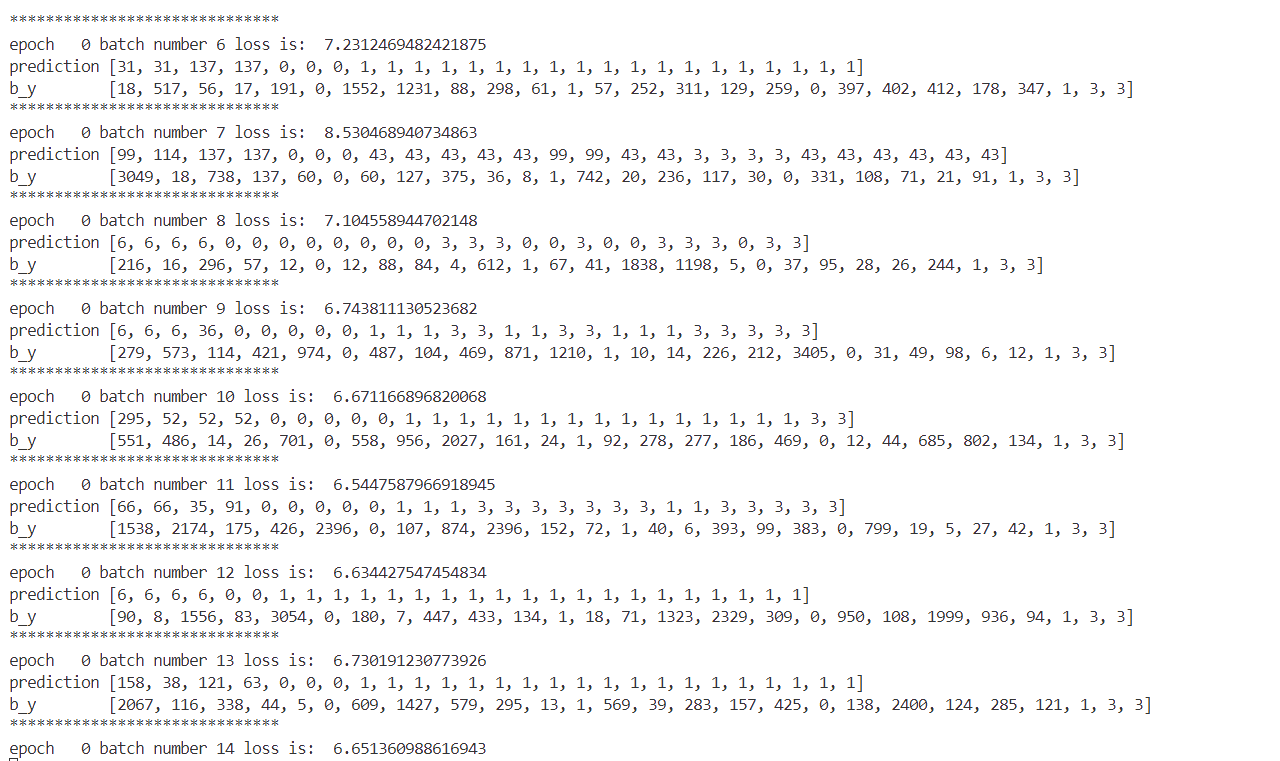
**训练过程：**

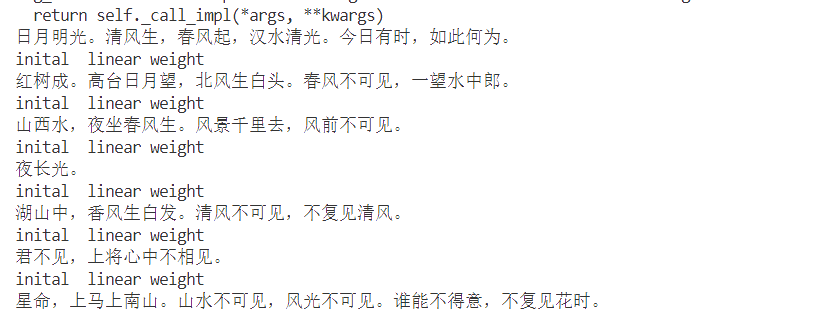
在数据处理和模型初始化完成后，先对优化器optim.RMSprop和损失函数torch.nn.NLLLoss()初始化。在每轮训练中，取出一个批次的数据，对于每一对数据，模型接受一个序列x预测下一个单词。计算损失值，进行反向传播更新参数值。

**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：





**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。