智能系统原理与开发 Lab2

18302010018 俞哲轩

Hidden Markov Model

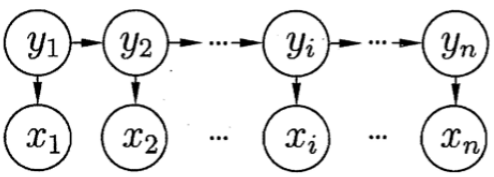
基本概念

隐马尔可夫模型是关于时序的概率模型，描述由一个隐藏的马尔可夫链随机生成不可观测的状态的序列，再由各个状态随机生成一个观测从而产生观测序列的过程。

隐马尔可夫模型的三大决定要素是：**初始状态概率向量**π，**状态转移概率矩阵**A，**观测概率矩阵**B，因此，**隐马尔可夫模型**可以写成$\lambda = (A, B, \pi)$。

隐马尔可夫模型是一个**生成模型**，表示状态序列和观测序列的**联合分布**，但是状态序列是隐藏的，不可观测。

隐马尔可夫模型用于标注问题的时候，此时**状态**对应着**标记**——标注问题，即是给定观测序列预测其对应的状态序列，即标记序列。



代码实现

实现HMM，最重要的即是得到它的三要素，即**初始状态概率向量**π，**状态转移概率矩阵**A，**观测概率矩阵**B。因此，模型训练的过程，其实就是在训练集上**统计**出(A, B, π)的具体概率值：

```
def train(self, train_file, encoding='utf-8'):
  curr_state = None
  last_state = tag2number['E']

  with open(train_file, mode='r', encoding=encoding) as scanner:
    line = scanner.readline()
    while line:
      if line == '\n':
        pass
      else:
        word, tag = line.split()
        curr_state = tag2number[tag]
        self.A[last_state][curr_state] += 1
        self.B[curr_state][ord(word)] += 1
        self.PI[curr_state] += 1
        last_state = curr_state
       line = scanner.readline()

    for i in range(4):
      for j in range(65296, 65306):
        self.B[i][j] = self.B[i][j - 65296 + 48]

def change2probability(self):
  for i in range(4):
    self.A[i] = compute_probability(self.A[i])
    self.B[i] = compute_probability(self.B[i])
  self.PI = compute_probability(self.PI)

def compute_probability(array):
  total = np.log(np.sum(array))
  for i in range(len(array)):
    if array[i] == 0:
      array[i] = float(-2 ** 31)
    else:
      array[i] = np.log(array[i]) - total
  return array
```

在执行**标注问题**的时候，其实就是**预测问题**——已知模型$\lambda = (A, B, \pi)$和观测序列$O = (o_1, o_2, \ldots, o_T)$，求对给定观测序列**条件概率**$P(I|O)$**最大**的**状态序列**$I = (i_1, i_2, \ldots, i_T)$。

对于标注问题，**观测序列**即是我们的**文字**，**状态序列**即是**文字的标记**，我们可以用**维特比算法**解隐马尔可夫模型预测问题。

定义在时刻t状态为i的所有单个路径$(i_1, i_2, \ldots, i_t)$中概率最大值为$\delta_t(i)$，定义在时刻t状态为i的所有单个路径$(i_1, i_2, \ldots, i_{t-1}, i)$中概率最大的路径的第$t-1$个结点为$\psi_t i$，则**维特比算法**可表示为：

①初始化：

$$\delta_1(i) = \pi_i b_i(o_i)$$
$$\psi_1(i) = 0$$
$$i = 1, 2, \ldots, N$$

②递推，对$t = 2, 3, \ldots, T$：

$$\delta_t(i) = \max_{1 \le j \le N}[\delta_{t-1}(j) a_{ji}] b_i(o_t)$$
$$\psi_t(i) = \arg \max_{1 \le j \le N}[\delta_{t-1}(j) a_{ji}]$$
$$i = 1, 2, \ldots, N$$

③终止：

$$P^* = \max_{1 \le i \le N} \delta_T(i)$$
$$i_T^* = \arg \max_{1 \le i \le N}[\delta_T(i)]$$

④最优路径回溯：

$$\text{对} t = T-1, T-2, \ldots, 1$$
$$i_t^* = \psi_{t+1}(i_{t+1}^*)$$

求得最优路径$I^* = (i_1^*, i_2^*, \ldots, i_T^*)$。

```
def viterbi(self, observations):
  length = np.shape(observations)[0]
  delta = np.zeros((length, 4))
  psi = np.zeros((length, 4), dtype=np.int64)

  # 初始化
  for i in range(4):
    delta[0][i] = self.PI[i] + self.B[i][ord(observations[0])]

  # 递推 t = 2, 3, ..., T
  for t in range(1, length):
    for i in range(4):
      max = delta[t - 1][0] + self.A[0][i]

      for j in range(4):
        temp = delta[t - 1][j] + self.A[j][i]
        if temp >= max:
          max = temp
          psi[t][i] = j

      delta[t][i] = max + self.B[i][ord(observations[t])]
```

```
    decode = np.empty(shape=length, dtype=str)
    # 终止
    max_delta_index = np.argmax(delta[-1])
    decode[-1] = number2tag[max_delta_index]

    # 最优路径回溯
    for t in range(length - 2, -1, -1):
      max_delta_index = psi[t + 1][max_delta_index]
      decode[t] = number2tag[max_delta_index]
    return ''.join(decode)
```

模型改进

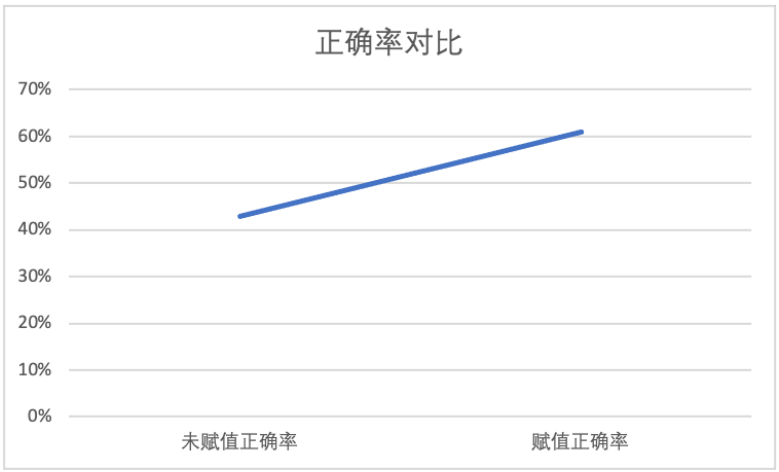
在实际标注中，遇到了以下几个问题，在实现过程中对模型进行了优化：

①应对训练集中没有的字

对于训练集上没有出现的字，其状态转移概率矩阵A，观测概率矩阵B对应的值都为0；如果在测试集上碰到一个训练集中没有出现过的字，则对应的标注大概率会出错，且后续的标注会被影响，很大概率继续出错。

所以对所有未出现过得字，其状态转移概率矩阵A，观测概率矩阵B对应的值**赋一个最小值**，即$\frac{1}{\text{文字出现次数}}$。

经过以上优化，模型正确率从43%提升至61%：

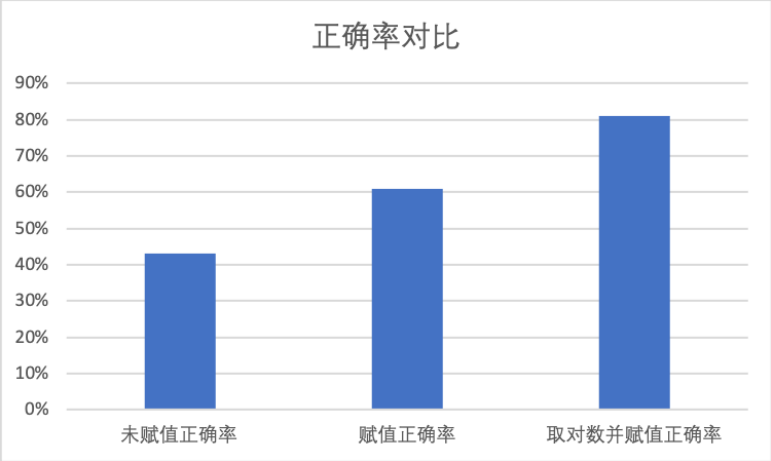


②应对概率乘积为0

在标注一句很长的句子时，最后输出的标注始终为**"S"**（即index为0时对应的标注），其原因为经过多次概率的连乘，起乘积最终小至0，导致标注出现错误。

为了避免这种情况的出现，可以**同比例放大**概率矩阵的值，也可以**取对数运算**，将**乘法**改为**加法**，我采用了取对数的方法，且给未出现的字赋值为2^{-31}，即最小的浮点数。

经过以上优化，模型正确率从61%提升至81%：



Conditional Random Field

基本概念

条件随机场是给定输入随机变量X条件下，输出随机变量Y的条件概率分布模型，其形式为参数化的对数线性模型。条件随机场的最大特点是假设输出变量之间的联合概率分布构成概率无向图模型，即马尔可夫随机场。

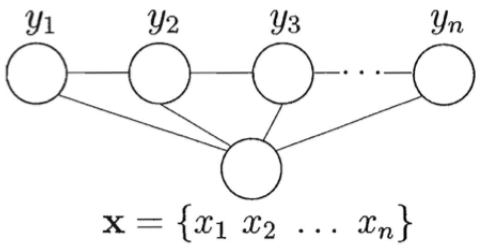
条件随机场是**判别模型**。

线性链条件随机场是定义在**观测序列**与**标记序列**上的条件随机场。线性链条件随机场一般表示为给定观测序列条件下的标记序列的条件概率分布，由参数化的对数线性模型表示。模型包含特征及相应的权值，特征是定义在线性链的边与结点上的。

参数形式的数学表达式是：

$$P(y|x)=\frac{1}{Z(x)}exp(\sum_{i,k}\lambda_k t_k(y_{i-1},y_i,x,i)+\sum_{i,l}\mu_l s_l(y_i,x,i))$$

$$Z(x)=\sum_y exp(\lambda_k t_k(y_{i-1},y_i,x,i)+\sum_{i,l}\mu_l s_l(y_i,x,i))$$



代码实现

在CRF模型的训练过程中，逐步调整相应模版的分数，即权重，具体方法如下：

1. 对于标注正确的字，不需要调整；
2. 对于标注错误的字，根据unigram和bigram的template逐一调整，具体操作如下：
    1. 对于错误的观测和模版，如果其之前没有分数，则设置成-1；如果其之前已经有分数，则分数减一；

2. 对于正确的观测的模版，如果其之前没有分数，则设置成1，如果其之前已经有分数，则分数加一。

```
def train(self, sentence, tags):
  my_tags = self.segment(sentence)
  length = len(sentence)
  wrong_num = 0
  for i in range(length):
    my_tag = my_tags[i:i + 1]
    tag = tags[i:i + 1]
    if my_tag != tag:
      wrong_num += 1
      for U_idx in range(len(self.unigram)):
        uni_key = get_score_map_key(self.unigram[U_idx], str(U_idx), sentence,
i, my_tag)
        if uni_key in self.score_map.keys():
          self.score_map[uni_key] -= 1
        else:
          self.score_map[uni_key] = -1
        uni_score_map_key = get_score_map_key(self.unigram[U_idx], str(U_idx),
sentence, i, tag)
        if uni_score_map_key in self.score_map.keys():
          self.score_map[uni_score_map_key] += 1
        else:
          self.score_map[uni_score_map_key] = 1
      for B_idx in range(len(self.bigram)):
        if i > 1:
          bi_key = get_score_map_key(self.bigram[B_idx], str(B_idx), sentence,
i, my_tags[i - 1:i + 1])
          bi_score_map_key = get_score_map_key(self.bigram[B_idx], str(B_idx),
sentence, i, tags[i - 1:i + 1])
        else:
          bi_key = get_score_map_key(self.bigram[B_idx], str(B_idx), sentence,
i, my_tag)
          bi_score_map_key = get_score_map_key(self.bigram[B_idx], str(B_idx),
sentence, i, tag)
        if bi_key in self.score_map.keys():
          self.score_map[bi_key] -= 1
        else:
          self.score_map[bi_key] = -1
        if bi_score_map_key in self.score_map.keys():
          self.score_map[bi_score_map_key] += 1
        else:
          self.score_map[bi_score_map_key] = 1
  return wrong_num
```

模型改进

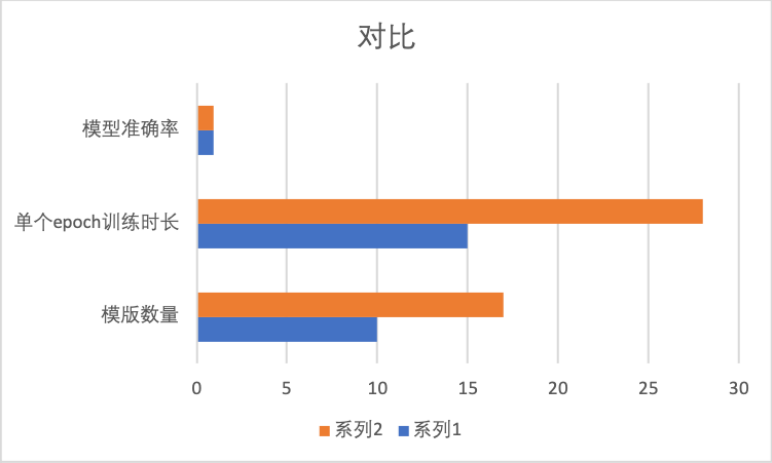
①增加模版

在已有模版的基础上，再增加如下模版：

```
# 已有模版
[-2], [-1], [0], [1], [2]
[-2, -1], [-1, 0], [-1, 1], [0, 1], [1, 2]

# 新增模版
[-2, -1, 0], [-1, 0, 1], [0, 1, 2]
[-2, -1, 1], [-2, 0, 1], [-1, 0, 2], [-1, 1, 2]
```

但是在实际训练中发现，增加模版的效果并不理想。具体表现在：训练时间**大幅增加**，但是模型的正确率并**没有显著提升**，甚至**没有提升**



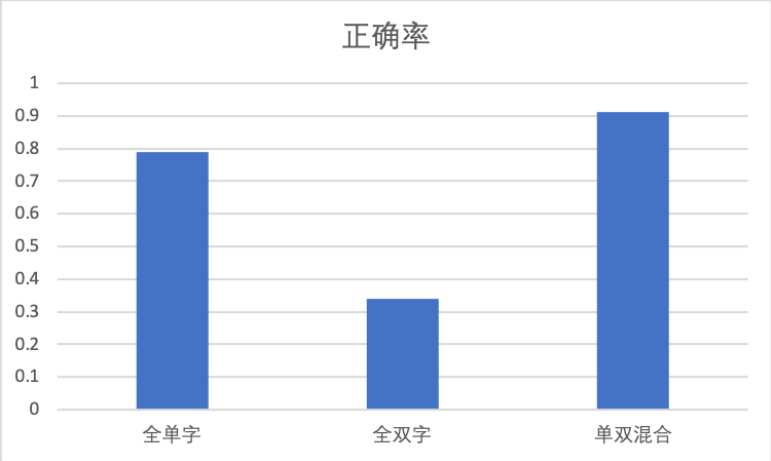
由数据可见，训练时长大致与模版数量成正比，但是正确率却几乎**没有提升**，分析其原因：可能是因为中文里面，两个字、三个字组成的词语较多，而四个字以上组成的词语较少，且已有的模版`[-2, -1], [-1, 1], [1, 2]`，已经可以建模连续三个字及以上组成的词语了，所以设计更多的模版效果提升不明显。

②删减模版

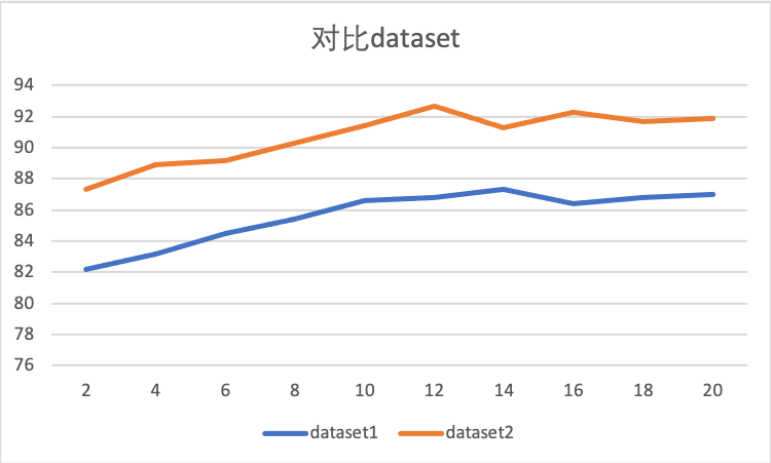
测试哪些模版更加重要：

实验结果表明，unigram和bigram的**单字模版**比**双字模版**更为**重要**，考虑其原因：

1. 可能是因为双字模版出现的次数较少，其分数较低，在预测的时候的影响较少，所以删去双字模版对模型性能的降低影响较少
2. 汉语以单字、双字和三个字的词语居多，单字模版已经能学习到大量的特征



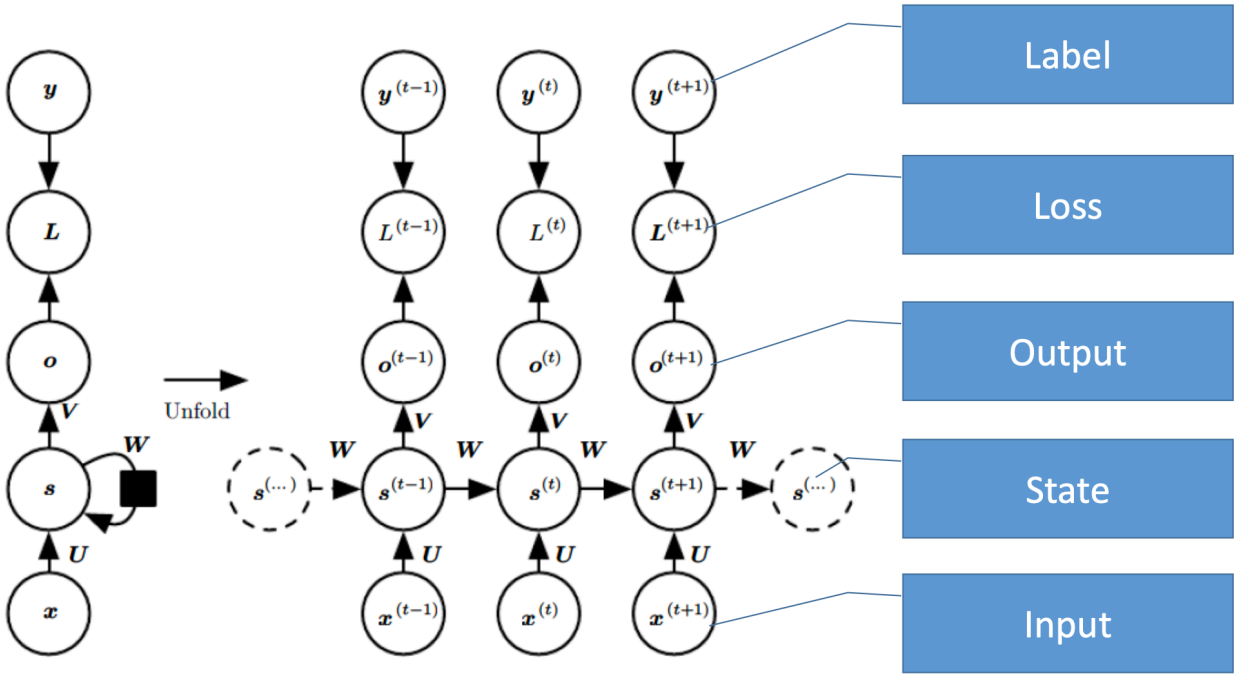
③dataset的影响



Bidirectional Long Short-Term Memory

RNN

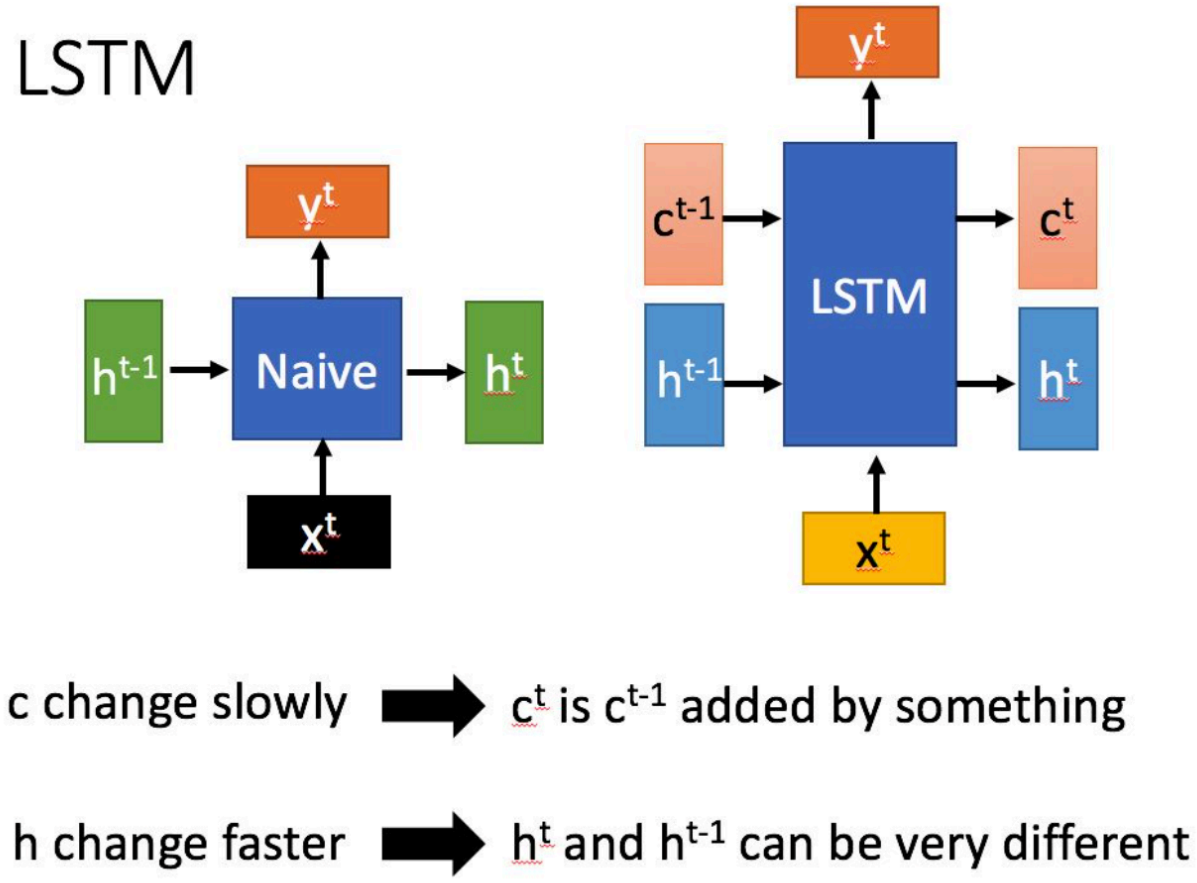
循环神经网络是一类用于**处理序列数据**的神经网络，特点是：每个时间步都有输出。



LSTM

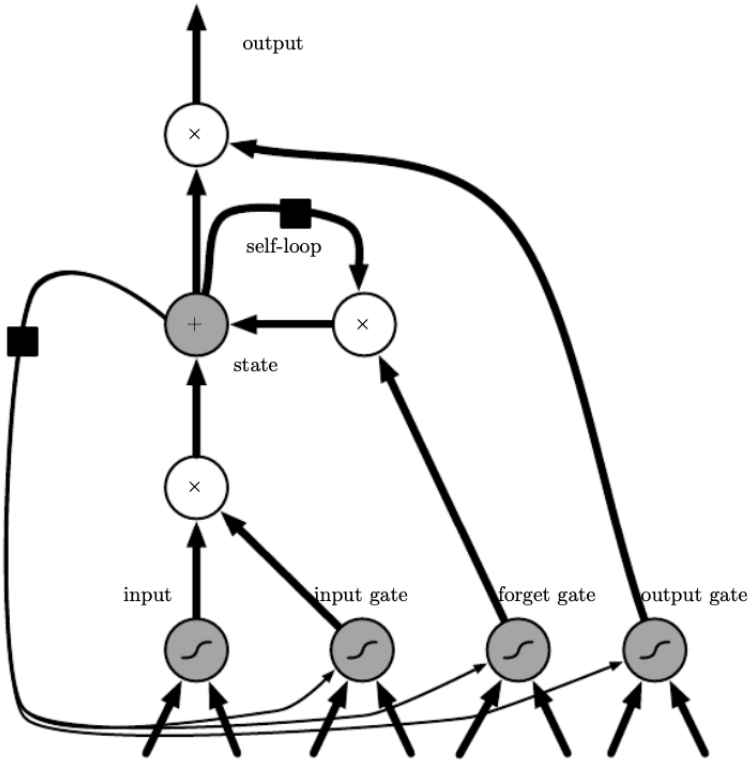
RNN存在**长期依赖**的问题，其根本问题是：经过许多阶段传播后的梯度倾向于**消失**（大部分情况）或者**爆炸**（很少）。

LSTM是一种特殊的RNN，主要是为了解决长序列训练过程中的梯度消失和梯度爆炸问题。简单来说，就是相比普通的RNN，LSTM能够在更长的序列中有更好的表现。



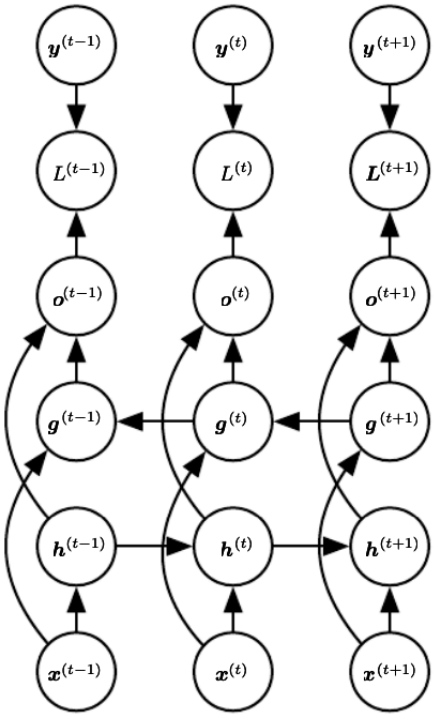
相比RNN只有一个传递状态h^t，LSTM有两个传输状态，一个c^t（cell state），和一个h^t（hidden state）。图中，RNN中的h^t对于LSTM中的c^t。

LSTM对于传递下去的c^t改变得很慢，通常输出的是c^t上一个状态传过来的c^{t-1}加上一些数值；而h^t则在不同节点下往往会有很大的区别。



BiRNN

RNN和LSTM都只能依据**之前时刻**的时序信息来预测下一时刻的输出，但在有些问题中，**当前时刻的输出**不仅和**之前的状态有关**，还可能和**未来的状态有关**。比如预测一句话中缺失的单词不仅需要根据前文来判断，还需要考虑它后面的内容，真正做到基于上下文判断。BiRNN有两个RNN上下叠加在一起组成的，输出由这两个RNN的状态**共同决定**。



BiLSTM

使用**BiLSTM**进行训练及预测，其优点在于：

1. 很好解决长期依赖
2. 一个字的前后状态都可能影响它的标注

在实际的**标注**任务中，实际上**不太可能碰到长期依赖的问题**，这是在**生成语句**的任务中更常见；但是**一个字的前后状态确实会影响它的标注**。

模型改进

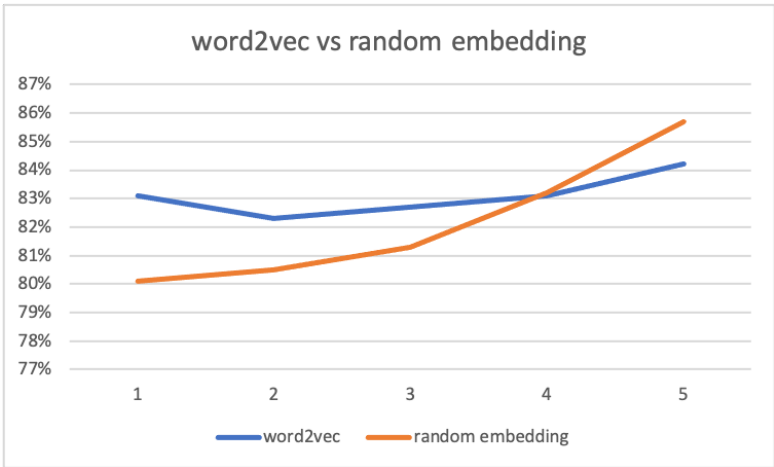
①使用word2vec预训练

word2vec可以自动实现：

- 单词语义相似性的度量；
- 词汇的语义的类比。

对比word2vec与随机启动的embedding：

我们发现使用了word2vec启动的模型，在最初的epoch上表现要优于随机启动的embedding，但是随着训练次数的增加，采用随机启动的embedding的模型性能逐渐超过使用word2vec的模型。



②改变超参数

1. **隐层向量维度与模型收敛的关系**

隐层维度	正确率	误差
50	0.905787	6.331146
100	0.921403	2.869782
150	0.941239	2.838921
200	0.934892	2.880938
250	0.935390	3.112389

由上可以得出：隐层向量维度过大会**大大降低模型收敛速度**，但在模型正确率上的**提升很少**。

2. **学习率与模型收敛的关系**

学习率	正确率	误差
0.001	0.932389	3.089210
0.005	0.941239	2.838921
0.01	0.938920	2.892390
0.02	0.912389	4.023891

模型收敛速率会随着学习率的增大而加快，但是学习率过大了之后，训练效果反而不佳——这点与其他神经网络相似，学习率选择适当值最佳。

HMM、CRF、DNN对比

HMM与CRF & DNN

隐马尔可夫模型假设：

- 当前状态只与上一状态有关
- 当前可见随机变量只与当前隐藏状态有关

这样的**强假设**有一定的合理性，但是其实也有很大的局限性：因为对于分词的问题来讲，邻近的字往往有很大的关系，而邻近的状态即标签也有很大的关系。

仅仅考虑前一个状态的影响是**远远不够**的：一个字的状态还可能**与后面的字有关**，还可能**与前面不止一个字有关**。

而这两个问题，在CRF和BiLSTM中都**得到解决**。

尤其是**CRF**，CRF和HMM同为**概率图模型**，HMM的三要素（初始状态概率向量，状态转移概率矩阵，观测概率矩阵）都可以**通过定义合适的模版**，在CRF中体现出来，故CRF不管是在**实践**还是**理论**上都要**优于**HMM。

CRF和LSTM究竟比HMM强在哪里呢？我们举例来看——

对于如下输入：

```
sentence = ['广东省', '广州市', '瘦西湖', '西湖', '东方明珠', '万里长城', '上海市', '北京市', '浙江省', '杭州市', '广州塔', '十二月二十七日', '三百一十五']
gold = ['BIE', 'BIE', 'BIE', 'BI', 'BIIE', 'BIIE', 'BIE', 'BIE', 'BIE', 'BIE', 'BIE', 'BIEBIIE', 'BIIIE]
```

```
检查HMM模型 -->
['BES', 'BEB', 'BIE', 'BE', 'BEBE', 'EBEB', 'SBE', 'BEB', 'BES', 'BEB', 'BEB', 'IIEBIIE', 'BIIII']
准确率: 0.4608058608058607

检查CRF模型 -->
['BIE', 'BIE', 'BIE', 'BE', 'BEBE', 'SSBE', 'BIE', 'BIE', 'BIE', 'BIE', 'BIE', 'BIEBIIE', 'BIIIE']
准确率: 0.8653846153846154

检查DNN模型 -->
['BIE', 'BIE', 'SBE', 'BE', 'BEBE', 'SSBE', 'BIE', 'BIE', 'BIE', 'BIE', 'BES', 'BBEBIIE', 'BIIIE']
准确率: 0.7518315018315018
```

我们可以看到，对于一些**专有名字**、或是**专有名字+修饰名词**（比如广东+省）、**日期**、**数字**等的结构，HMM的表现**明显不如**CRF和DNN，原因就在于HMM当前状态只与前一次状态有关，而不考虑**上下文影响**和**更之前的状态的影响**。

CRF与DNN

CRF不像LSTM等模型能够考虑长远的上下文信息。通过特征模版去扫描整个句子，它考虑的更多是整个句子的**局部特征的线性加权组合**。

在CRF训练的过程中，实际上是计算一个**联合条件概率**，即找到一个最优化的序列$I=(i_1,i_2,\ldots,i_t)$，使得$P(i_1,i_2,\ldots,i_t|O,w)$最高。故CRF优化的是一整个序列$I=(i_1,i_2,\ldots,i_n)$，而非某一个时刻的$i_t$，CRF是将**整个序列的最优化**作为最终目标，而非将每个时刻的**最优解拼接起来**——在这一点上，CRF要**优于**LSTM。

但是像RNN、LSTM、BiLSTM这些DNN模型，它们在**时序建模能力上很强大**，能够**抓住更加长远的上下文信息**，此外还**具备神经网络拟合非线性的能力**，这些都是CRF无法超越的地方。对于某一时刻t，输出层o_t受到隐层h_t（包含上下文信息）和输入层i_t（包含当前的输入）的共同影响，但是**当前时刻的输出**和**其它时刻的输出**是**相互独立**的，对当前时刻t来说，我们希望找到一个概率最大的o_t，但是在其它时刻的输出对当前的输出**没有影响**。如果不同时刻输出之前存在较强的依赖的话（在**生成语句**的任务中，比如一些**固定用法的搭配**、一些**成语**、**形容词后面通常接名词等搭配**），LSTM**无法对这些约束进行建模**，其性能会受到一些限制。

具体到本次实验，对比CRF和BiLSTM：

1. 从性能的角度，两者**十分接近，差别不大**；
2. 从训练速度的角度，CRF的训练速度**明显优于**BiLSTM的训练速度；
3. 从任务场景的角度，标注认识实际上**不需要特别依赖长久的上下文信息**，所以BiLSTM模型只会**增加额外的复杂度**；
4. 从数据规模的角度，在**实验数据规模较小**的情况下，整体来看，CRF的实验效果要略优于BiLSTM。

CRF和BiLSTM的性能对比如何呢？我们距离来看——

对于如下输入：

```
sentence = ['巴塞罗那', '布宜诺斯艾利斯', '顿涅茨克矿工', '名古屋鲸八', '皇家马德里']
gold = ['BIIE', 'BIIIIIE', 'BIIIIE', 'BIIIE', 'BIIIE']
```

```
检查CRF模型 -->
['BIIE', 'BIIIIIE', 'BIIEBE', 'BIIIE', 'BEBIE']
准确率：0.8533333333333333

检查DNN模型 -->
['BIES', 'BIIIIIE', 'BIIIIE', 'SBESS', 'BEBIE']
准确率：0.62
```

两者性能**各有千秋**，在这些专有名词问题上，CRF和BiLSTM都**不能保证完全做对**，分析原因可能如下：

1. 训练数据不够多，加上**足够多的训练数据**，或者**使用Bert预训练**的话，两者都有可能标注正确；
2. 与标注有关，比如“顿涅茨克矿工”一次，可以将它理解为一个完整的球队名字，标注为“BIIIIE”；也可以将它理解为“顿涅茨克”+“矿工”，标注为“BIIE”+“BE”。