

# 中山大学计算机学院

## 人工智能

### 本科生实验报告

(2022 学年春季学期)

课程名称: Artificial Intelligence

教学班级	计科 1 班	专业 (方向)	计算机科学与技术
学号	21307077	姓名	凌国明

## 一、实验题目

### 神经网络文本分类

使用 *CNN* 或 *RNN* 进行文本分类任务

需要进行文本清洗，清洗掉低频词，标点，数字等等

给定预训练的词嵌入数据（50 维）

在训练集上训练，在验证集上调参，在测试集上评估效果

```
data > 20ns > raw > ≡ test.txt
1 alt.atheism yeah , do you expect people to read the -
steam ! jim , sorry i ca n't pity you , jim and i 'm
will all end happily ever after anyway maybe if you :
flintstone 's chewables ! \) bake timmons , iii
2 talk.politics.mideast although i realize that prin-
sort about the arab countries if you want to continue
sort of questions of arab countries as well you real-
your fixation on israel would begin to look like the
nothing more than a fancy name for some bigot who hat-
3 sci.crypt notwithstanding all the legitimate fuss a
1000 , as i suspect 'clipper' phones will be \( b \)
```

#### 数据说明

每一行是一个数据项，第一列是标签，第二列是文本，中间以 '\t' 分隔

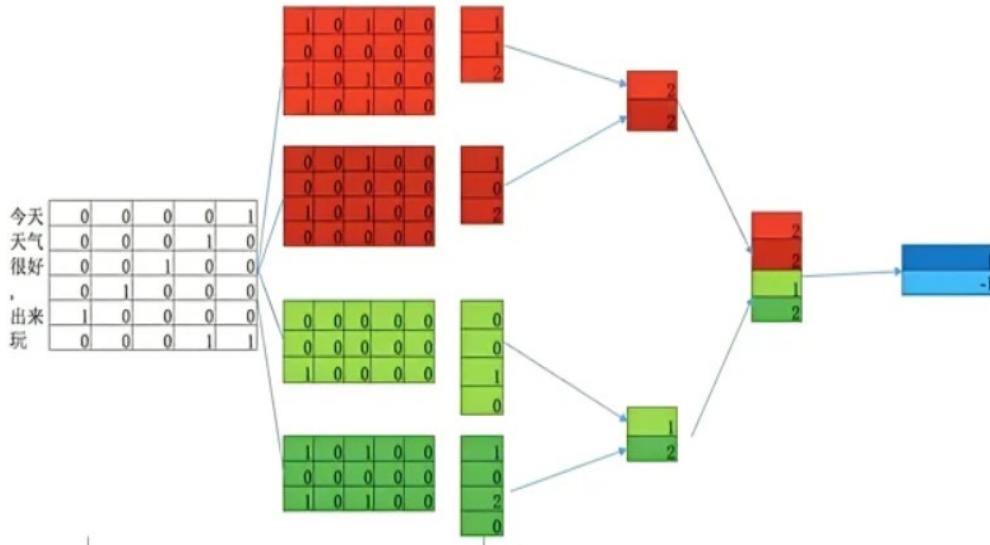
文本共有 20 个类别，文本中分词不一定准确，有许多噪声和干扰项

## 二、实验内容

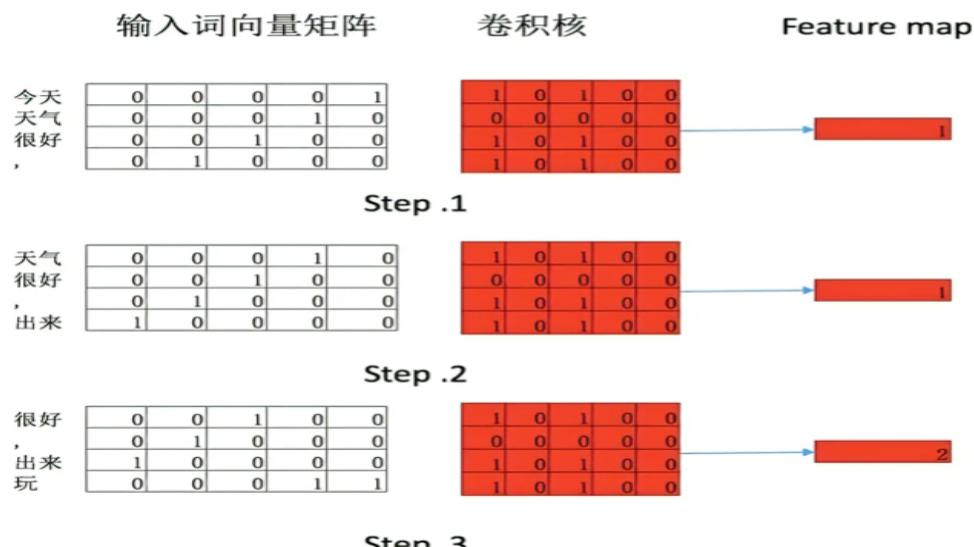
### 1、算法原理

#### TextCNN

将  $CNN$  用于  $NLP$ , 以下是网络结构



与CV中的CNN相比, TextCNN有如下的不同点: 自然语言是一维数据, 虽然经过word-embedding 生成了二维向量, 但是对词向量做从左到右滑动来进行卷积没有意义. 比如 "今天" 对应的向量[0, 0, 0, 0, 1], 按窗口大小为 1\* 2 从左到右滑动得到[0,0], [0,0], [0,0], [0, 1]这四个向量, 对应的都是"今天"这个词汇, 这种滑动没有帮助, 所以TextCNN都是 **从上到下滑动**



其中卷积核可以有多个 channel

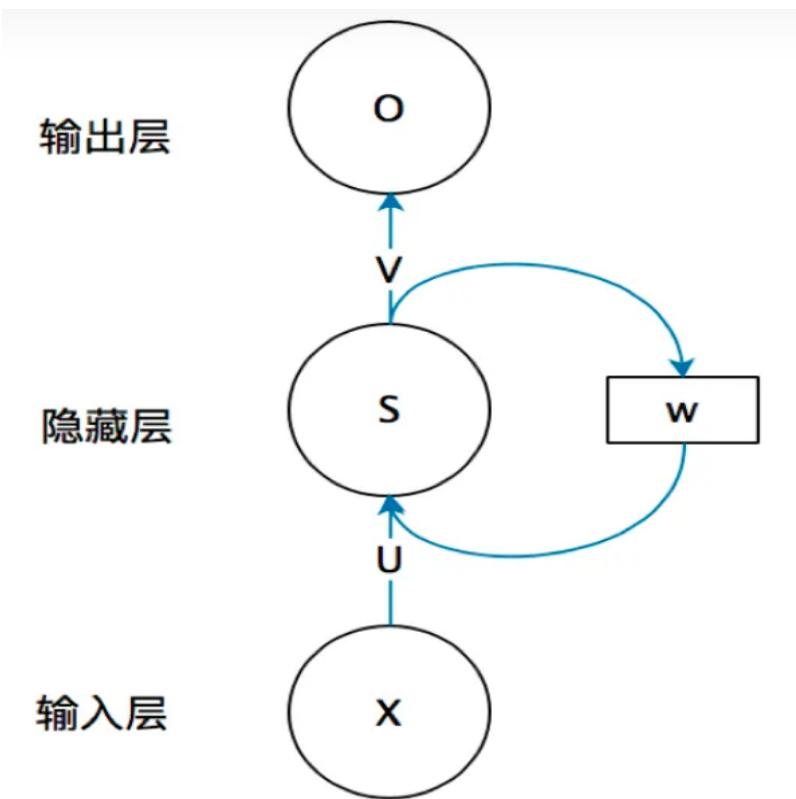
# RNN

## RNN 网络结构

普通的神经网络都只能单独地处理一个个的输入，前一个输入和后一个输入是完全没有关系的。但是，某些任务需要能够更好的处理序列的信息，即前面的输入和后面的输入是有关系的。

如在“预测词性”的任务中，输入为一个句子的各个分词，如果用普通的全连接神经网络来做，那么句子的各个分词的独立地输入神经网络，相互之间没有影响地得到输出。

但在一个句子中，前一个单词其实对于当前单词的词性预测是有很大影响的，如动词后面很有可能会接名词。这就导出了RNN。



1.  $t$  时刻输入向量  $x_t$
2.  $x_t$  经过矩阵  $U$
3.  $s_{t-1}$  经过矩阵  $W$
4. 得到  $s_t$  的计算公式为  $f(U * x_t + W * s_{t-1} + b_1)$
5. 输出层  $o_t$  公式为  $g(V * s_t + b_2)$

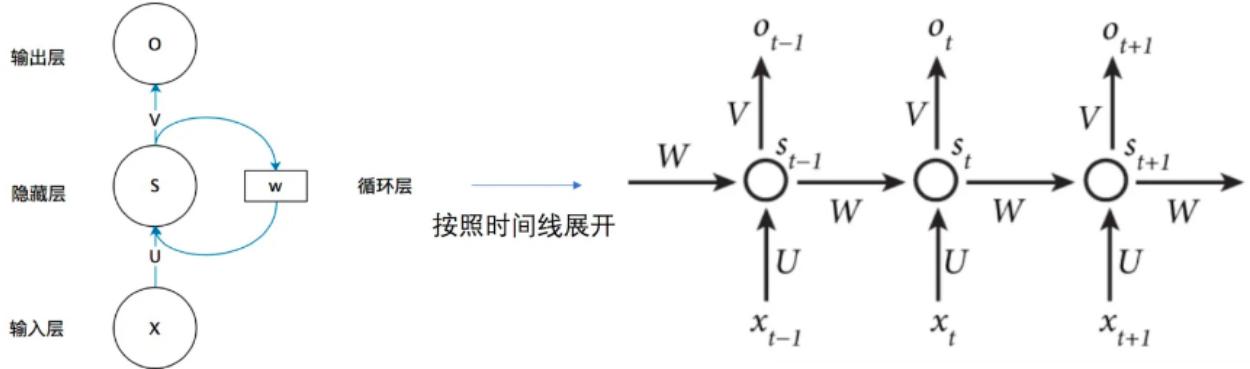
$x_t$  是  $t$  时刻的输入， $s_t$  是  $t$  时刻隐藏层的输出

其中  $s_0$  是计算第一个隐藏层需要的，初始化为全 0

$o_t$  是  $t$  时刻输出层的输出

$f$  和  $g$  是非线性激活函数， $b_1$  和  $b_2$  是偏置常数

按时间线展开：



公式：

$$o_t = g(V \cdot s_t + b_2)$$

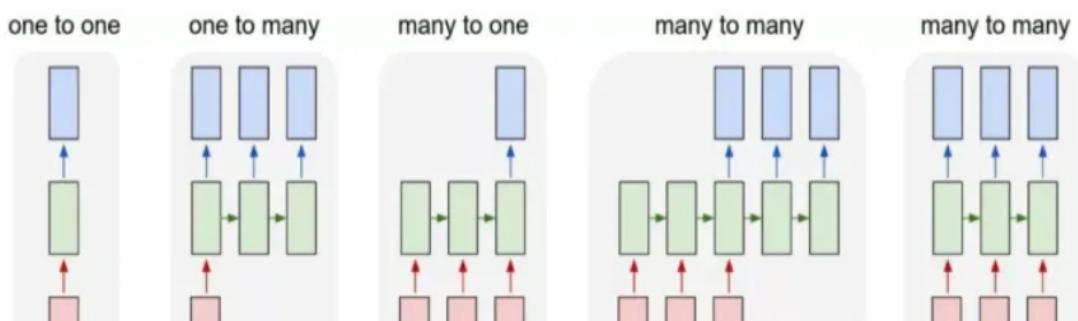
$$s_t = f(U \cdot x_t + W \cdot s_{t-1} + b_1)$$

输出公式推导：

$$\begin{aligned} o_t &= g(V \cdot s_t + b_2) \\ &= g(V \cdot f(U \cdot x_t + W \cdot s_{t-1} + b_1) + b_2) \\ &= g(V \cdot f(U \cdot x_t + W \cdot f(U \cdot x_{t-1} + W \cdot s_{t-2} + b_1) + b_1) + b_2) \\ &= g(V \cdot f(U \cdot x_t + W \cdot f(U \cdot x_{t-1} + W \cdot f(U \cdot x_{t-2} + \dots))) + b_2) \end{aligned}$$

注意：

1. 可以将隐藏的状态  $s_{t-1}$  看作网络的记忆，它捕获有关所有先前时间步骤中发生的事件的信息。当下输出  $o_t$  根据时间  $t$  的记忆计算。
2. 上图在每个时间步都有输出，但根据任务，这可能不是必需的。例如，在预测句子的情绪时，我们可能只关心最终的输出，而不是每个单词之后的情绪。同样，我们可能不需要在每个时间步骤输入。所以，RNN结构可以是下列不同的组合：



## RNN 特性

长期依赖问题：若输入序列长，则传输距离长，通过参数的稀释，最前面对最后面影响小，丢失上下文联系。研究发现传统 RNN 的长期依赖问题是这种网络结构本身的问题。

梯度消失问题：神经网络的权重/偏置梯度极小，导致神经网络参数调整速率急剧下降

梯度爆炸问题：神经网络的权重/偏置梯度极大，导致神经网络参数调整幅度过大，矫枉过正

## LSTM

### 概念

长短期记忆 (Long short-term memory, 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，相比普通的RNN，LSTM能够在更长的序列中有更好的表现。

LSTM从被设计之初就被用于解决一般递归神经网络中普遍存在的**长期依赖**问题，使用LSTM可以有效的传递和表达长时间序列中的信息并且不会导致长时间前的有用信息被忽略（遗忘）。与此同时，LSTM还可以解决RNN中的**梯度消失/爆炸问题**。

LSTM的设计借鉴了人类对于自然语言处理的直觉性经验。一个直观例子如下：

“这个笔记本非常棒，纸很厚，料很足，用笔写起来手感非常舒服，而且没有一股刺鼻的油墨味；  
更加好的是这个笔记本不但便宜还做工优良，我上次在别家买的笔记本裁纸都裁不好，还会割伤手.....”

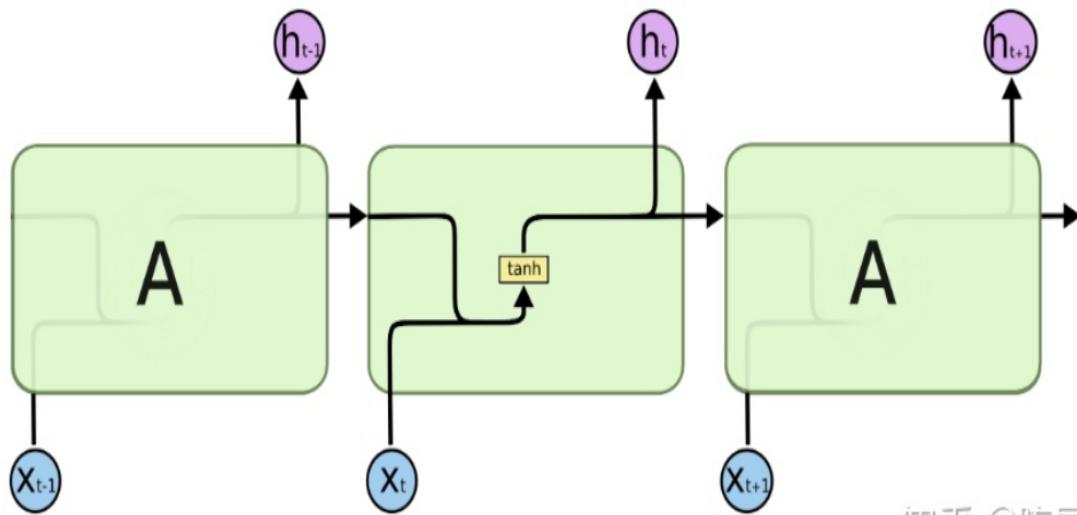
这段话中有几个重要的关键词“纸好”，“没味道”，“便宜”和“做工好”。从这个例子得出：

1. 在一个时间序列中，不是所有信息都是同等有效的，大多数情况存在“**关键词**”或者“关键帧”
2. 我们会在从头到尾阅读的时候“自动”**概括**已阅部分的内容并且用之前的内容帮助理解后文

基于以上这两点，LSTM的设计者提出了“长短期记忆”的概念——只有一部分的信息需要长期的记忆，而有的信息可以不记下来。同时，我们还需要一套机制可以动态的处理神经网络的“记忆”，因为有的信息可能一开始价值很高，后面价值逐渐衰减，这时候我们也需要让神经网络学会“遗忘”特定的信息

### 架构

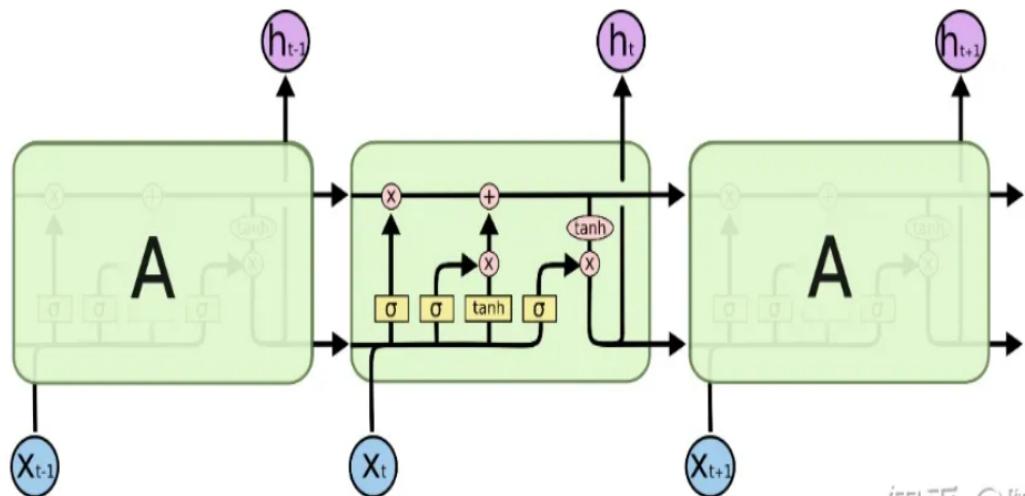
传统RNN的架构如下：



标准 RNN 中的重复模块包含单个层。

知乎 @陈昱

LSTM架构图如下：

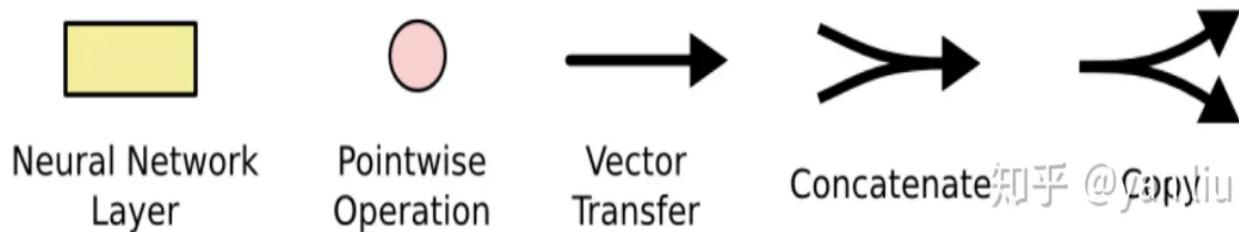


LSTM 中的重复模块包含四个交互层。

知乎 @陈昱

## 数据流向符号及含义

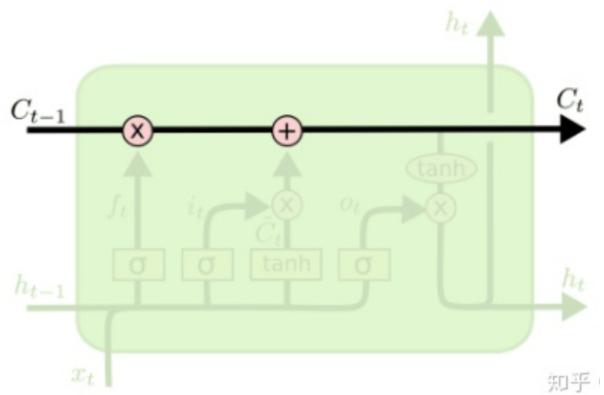
每个黄色方框表示一个神经网络层，由权值，偏置以及激活函数组成；每个粉色圆圈表示元素级别操作；箭头表示向量流向；相交的箭头表示向量的拼接；分叉的箭头表示向量的复制。



知乎 @Copyliu

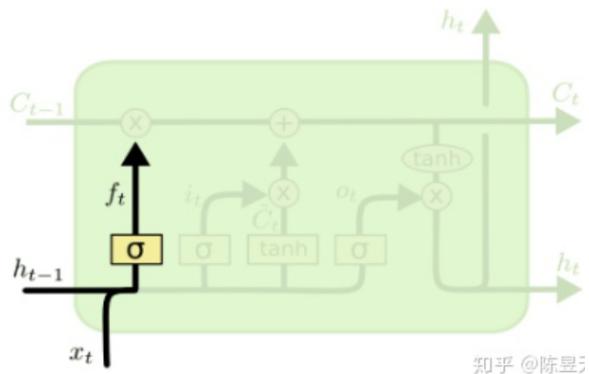
## 单元状态

LSTM能够从RNN中脱颖而出的关键就在于上图中从单元中贯穿而过的线——神经元的隐藏态（单元状态），我们可以将神经元的隐藏态简单的理解成递归神经网络对输入数据的“记忆”，用  $C_t$  表示神经元在  $t$  时刻过后的“记忆”，这个向量涵盖了在  $t + 1$  时刻前神经网络对于所有输入信息的“**概括总结**”



$$\text{公式: } C_t = f_t \cdot C_{t-1} + i_t \cdot C_t' \quad (\text{都是点乘})$$

## 遗忘门



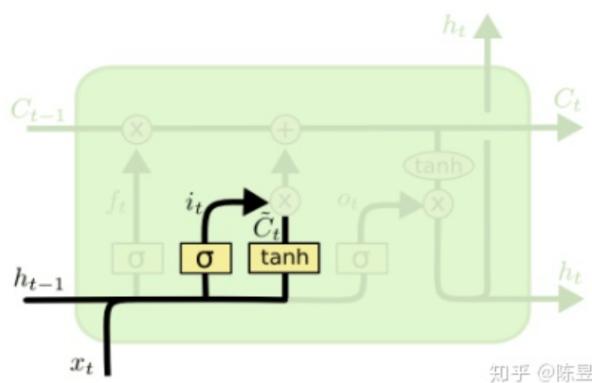
对于上一时刻LSTM中的单元状态来说，一些“信息”可能会随着时间的流逝而“过时”。为了不让过多记忆影响神经网络对现在输入的处理，我们应该选择性遗忘一些在之前单元状态中的分量——这个工作就交给了“遗忘门”

每一次输入一个新的输入，LSTM会先根据新的输入和上一刻的输出**决定遗忘掉之前的哪些记忆**——输入和上一步的输出会整合为一个单独的向量，然后通过 *sigmoid* 神经层，最后点对点的乘在单元状态上。因为 *sigmoid* 函数会将任意输入压缩到  $(0, 1)$  的区间上，我们可以非常直觉的得出这个门的工作原理——如果整合后的向量某个分量在通过 *sigmoid* 层后变为0，那么显然单元状态在对位相乘后对应的分量也会变成0，换句话说，“遗忘”了这个分量上的信息；如果某个分量通过 *sigmoid* 层后为1，单元状态会“保持完整记忆”。不同的 *sigmoid* 输出会带来不同信息的记忆与遗忘。通过这种方式，LSTM可以长期记忆重要信息，并且记忆可以随着输入进行**动态调整**

遗忘门的计算公式如下，其中  $f_t$  表示 *sigmoid* 神经层的输出向量

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (\text{得到的向量的每个元素在 } [0, 1] \text{ 之间})$$

## 记忆门



知乎 @陈昱

记忆门是用来控制是否将在  $t$  时刻（现在）的数据并入单元状态中的控制单位。首先，用  $tanh$  函数层将现在的向量中的**有效信息提取**出来，然后使用（图上  $tanh$  函数层左侧）的  $sigmoid$  函数来控制这些记忆要放“多少”进入单元状态。这两者结合起来就可以做到：

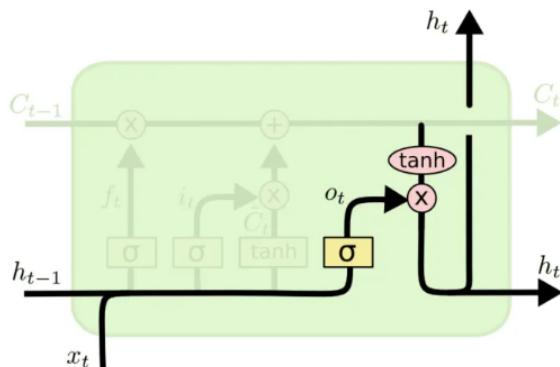
1. 从当前输入中**提取有效信息**
2. **对提取的有效信息做出筛选**，为每个分量做出评级 (0 ~ 1)，评级越高的最后会有越多的记忆进入单元状态

记忆门中的计算公式如下：

$$C'_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (\text{提取信息})$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (\text{筛选信息})$$

## 输出门



输出门，顾名思义，就是LSTM单元用于计算当前时刻的输出值的神经层。输出层会先将当前输入值与上一时刻输出值整合后的向量（也就是公式中的  $[h_{t-1}, x_t]$ ）用  $sigmoid$  函数提取其中的信息，接着，会将当前的单元状态通过  $tanh$  函数压缩映射到区间  $(-1, 1)$  中，最后两者点乘得到当前时刻的输出

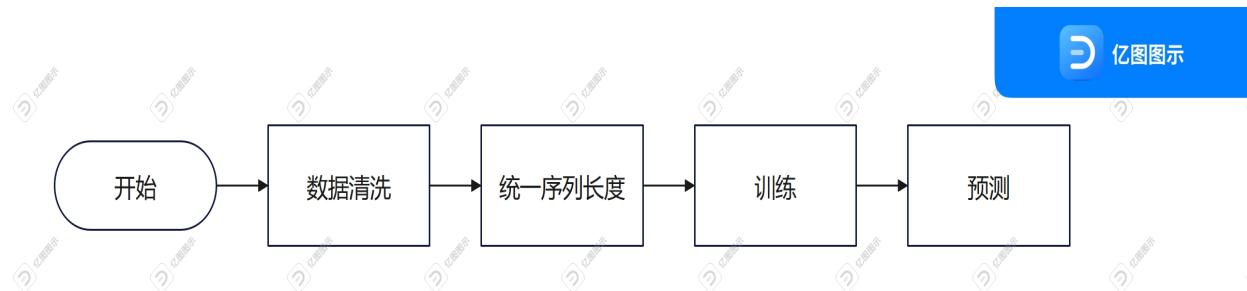
输出门中的计算公式如下：

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (\text{提取当前信息})$$

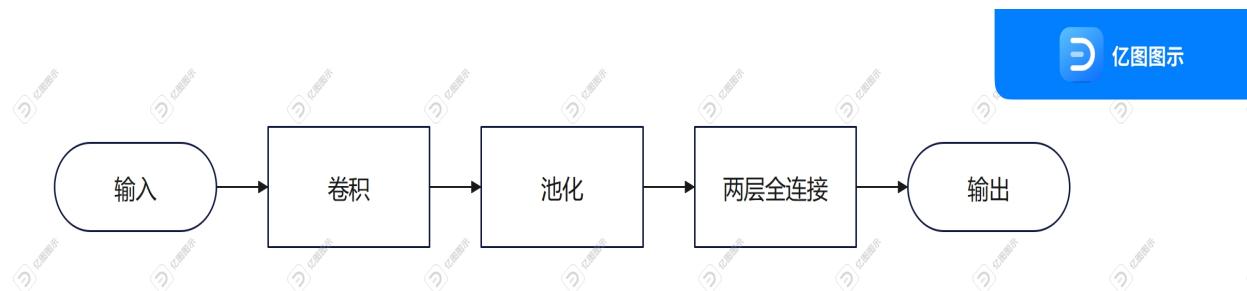
$$h_t = o_t \cdot \tanh(C_t) \quad (\text{点乘})$$

## 2、伪代码，流程图

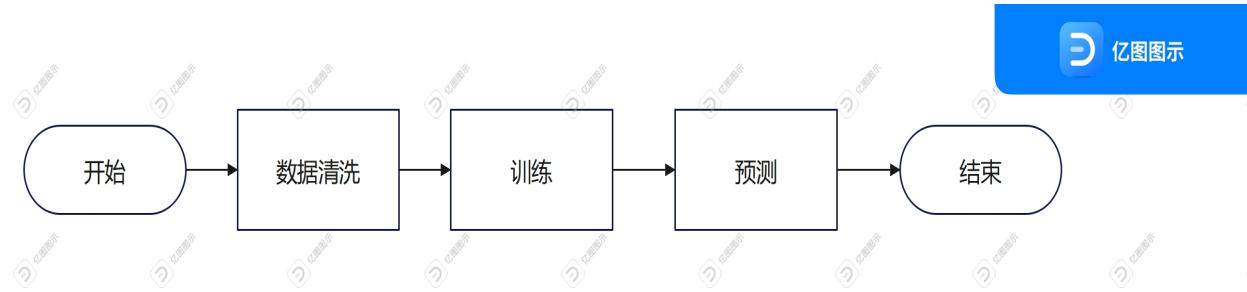
### CNN 处理流程



### CNN 网络结构

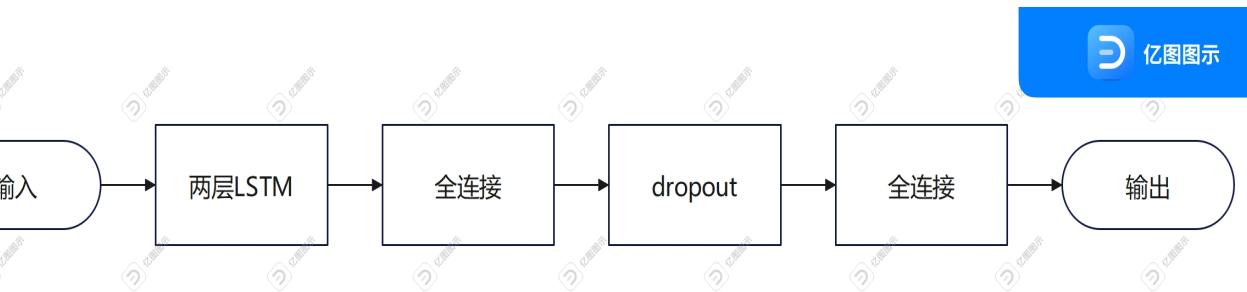


### RNN 处理流程



注意, \$CNN\$ 中需要统一序列长度, 而 \$RNN\$ 则不需要

### RNN 网络结构



### 3、关键代码展示（带注释）

#### 文本清洗

```
import re

def remove_punctuation(text_):
    # 删除标点符号
    text_ = re.sub(r'^\w\s]', '', text_)
    # 删除数字
    text_ = re.sub(r'\d+', '', text_)
    # 删除冠词
    text_ = re.sub(r'(?<=[^\w\s])(?=[\w\s])', '', text_)
    # 删除'\'和'\'后面的一个字符
    text_ = text_.replace('\\\\', '')
    # 去除两端的空格
    text_ = text_.lstrip('\t \n')
    # 将多个空格合并为一个空格后返回
    text_ = ' '.join(text_.split())
    return text_

if __name__ == '__main__':
    file_name = 'valid.txt'
    read_file = open(r'20ns/' + file_name, mode='r', encoding='ascii')
    write_file = open(r'processed_data/' + file_name, mode='w', encoding='ascii')
    for line in read_file:
        line = line.split('\t')
        label = line[0]
        text = line[1]
        write_file.write(label + '\t' + remove_punctuation(text) + '\n')
```

在这个分类任务中，大多是新闻？感觉标点（，。. ? !），数字，停用词（如冠词a, an, the等等）对分类的作用不大，因此认为可以去除。

经实验表明，通过re模块去除标点、冠词、数字后，同等条件下的准确率有较大的提高。

#### 处理结果

```
talk.politics.guns      put not your trust in princes is the biblical proverb the
of the us the idea that citizens had rights above those of the government was not
founders to a considerable extent englishmen also had those rights yes times chan
governments enslaving all of mankind was not possible until quite recently in the
serfs so having the people enslave themselves does not make anything better most f
those under them freedom of speech and freedom of religion are under real attack
soc.religion.christian the concept of god as a teacher is indeed interesting doe
not to mention thought provoking my own concept is that he is a father and we are
```

可见处理后的文本没有了标点、冠词、数字

# 数据集类

```
class MyDataSet(Dataset):
    def __init__(self, pwd, num=80):
        # 预训练词向量
        token_file = open("glove.6B.50d.txt", mode='r', encoding='utf8')
        # 数据集，格式为 label \t text
        file = open(pwd, mode='r', encoding='ascii')
        # 词典，分词：50维词向量
        self.word2vec = collections.defaultdict(default_value)
        # 利用预训练词向量文件，初始化词典
        for line in token_file:
            data = line.split(' ')
            if len(data) < 51:
                continue
            # 类型转化，从 str 转为 LongTensor
            for i in range(1, 51):
                data[i] = float(data[i])
            self.word2vec[data[0]] = data[1:]
        token_file.close()

        self.str_data = [] # 字符串形式的 data
        self.str_labels = [] # 字符串形式的 label
        self.vec_data = [] # 向量形式的 data
        self.vec_labels = [] # 向量形式的 label
        self.num_labels = [] # 数字形式的 label
        for line in file:
            label_, text_ = line.split('\t')
            text_ = text_.split(' ') # 以空格为分隔，分割 token
            self.str_data.append(text_) # 新增 字符串形式的 data
            self.str_labels.append(label_) # 新增 字符串形式的 label
            self.vec_data.append([]) # 新增 向量形式的 data
            self.vec_labels.append([0 for i in range(len(label_dict))]) # 新增 向量形式的 label
            self.vec_labels[-1][label_dict[label_]] = 1
            self.num_labels.append(label_dict[label_])
            for word_ in text_: # 对于 text 里的每个分词
                self.vec_data[-1].append(self.word2vec[word_]) # 每个 token 一个 []
        if len(self.vec_data[-1]) > num:
            self.vec_data[-1] = self.vec_data[-1][0:num]
        elif len(self.vec_data[-1]) < num:
            while len(self.vec_data[-1]) < num:
                self.vec_data[-1].append([0 for i in range(50)])
        if len(self.vec_data[-1]) != num:
            print('ERROR')
    file.close()
```

## CNN 类

```
class Model(nn.Module):
    def __init__(self, word_vec_size, out_size, num=80):
        super(Model, self).__init__()
        self.word_vec_size = word_vec_size
        self.out_size = out_size
        self.num = num
        self.cov = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=8, kernel_size=(3, 50), stride=1, padding=0, bias=False),
            nn.MaxPool2d(kernel_size=(2, 1), stride=2),
            nn.Tanh(),
        )
        self.linear = nn.Sequential(
            nn.Flatten(),
            nn.Linear(in_features=(num-2)*4, out_features=(num-2)*2),
            nn.Tanh(),
            nn.Linear(in_features=(num-2)*2, out_features=20),
        )
        self.softmax = nn.Softmax()

    def forward(self, X):
        X = self.cov(X)
        X = X.view(-1, (self.num-2)*4) # 展平
        X = self.linear(X)
        return X
```

## RNN 类

```
class Model(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, num_layers, num_classes, bidirectional=True):
        super(Model, self).__init__()
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, num_layers, batch_first=True, bidirectional=bidirectional)
        self.dropout = nn.Dropout(p=0.2)
        self.fc = nn.Linear(hidden_dim * (2 if bidirectional else 1), num_classes)

    def forward(self, x):
        # x 的形状: (batch_size, seq_length, embedding_dim)
        lstm_out, _ = self.lstm(x)
        # lstm_out 的形状: (batch_size, seq_length, hidden_dim)
        # 仅使用最后一个时间步的输出
        final_output = lstm_out[:, -1, :]
        final_output = self.dropout(final_output)
        # final_output 的形状: (batch_size, hidden_dim)
        out = self.fc(final_output)
        # out 的形状: (batch_size, num_classes)
        return out
```

# 各种实例化

```
batch_size = 1 # 暂时设置为 1
model = Model(50, 80, 2, 20) # 实例化模型
my_train_data = MyDataSet(r'processed_data/train.txt') # 训练集
my_test_data = MyDataSet(r'processed_data/valid.txt') # 验证集
train_dataloader = DataLoader(my_train_data, batch_size=batch_size, shuffle=True)
test_dataloader = DataLoader(my_test_data, batch_size=batch_size, shuffle=True)
optimizer = optim.Adam(model.parameters(), lr=0.001) # 优化器
scheduler = optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.99) # 调整学习率, 指数调整
loss_fun = nn.CrossEntropyLoss() # 交叉熵损失函数

if torch.cuda.is_available(): # 迁移到 GPU 上训练
    model = model.cuda()
    loss_fun = loss_fun.cuda()
```

## 训练代码（两个模型的训练测试代码都差不多）

```
for epoch in range(40):
    print('-----Epoch ', epoch+1, '-----', sep=' ')
    right = 0
    total = 0
    model.train() # 训练
    for sen, label in train_dataloader:
        if torch.cuda.is_available(): # 迁移到 GPU 上训练
            sen = sen.cuda()
            label = label.cuda()
        optimizer.zero_grad() # 清空梯度记录
        output = model(sen) # 输入送入模型得到输出
        loss = loss_fun(output, label) # 计算损失函数
        loss.backward() # 梯度反向传播
        optimizer.step() # 梯度下降
        total += 1

        if np.argmax(output.cpu().detach().numpy(), axis=1) == label.cpu().detach().numpy():
            right += 1
    print("训练集准确率: ", right / total)

    model.eval() # 测试 (计算在验证集上的准确率)
    with torch.no_grad(): # 不需要计算梯度
        right = 0
        total = 0
        # 同上

    scheduler.step() # 指数调整学习率
```

## 4、创新点&优化

### LSTM

RNN 存在以下问题

1. 长期依赖问题：若输入序列长，则传输距离长，通过参数的稀释，最前面对最后面影响小，丢失上下文联系。研究发现传统  $RNN$  的长期依赖问题是这种网络结构本身的问题。
2. 梯度消失问题：神经网络的权重/偏置梯度极小，导致神经网络参数调整速率急剧下降
3. 梯度爆炸问题：神经网络的权重/偏置梯度极大，导致神经网络参数调整幅度过大，矫枉过正

为了解决这个问题，用到了改进版的  $RNN$ ，也就是  $LSTM$ ，在  $LSTM$  中有记忆门和遗忘门等机制来解决长期依赖问题，这点在原理部分有具体的解释

### 学习率的探索

一开始设置学习率  $lr = 0.01$ ，认为也不高，但是就是发生了振荡，准确率一直上不去。

后来把学习率调低到  $0.001 - 0.00001$ ，感觉好了很多。

经思考，认为学习率太高会导致振荡，但学习率太低会导致算法收敛速度慢，因此想要**动态调整学习率**。

```
optimizer = optim.Adam(model.parameters(), lr=0.001) # 优化器
```

一开始设置一个高一点的学习率

```
scheduler = optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.99) # 调整学习率，指数调整
```

设置一个  $scheduler$  以动态调整学习率

```
scheduler.step() # 指数调整学习率
```

每个  $epoch$  完成后调用这个语句，使得学习率  $lr = lr \cdot \gamma$ ，这就是**学习率的指数衰减**

认为指数衰减可以使得学习率随着  $epoch$  的增加而减小，这意味着一开始可以更快地学习数据中的特征；同时训练后期可以更细地学习数据的特征，而不至于产生振荡。

# 三、实验结果与分析

## 1、实验结果展示示例

### TextCNN

在 *CNN* 中，输入需要统一序列长度。笔者是这样处理的：

1. 长度不足  $n$  的，在后面添加全零的 *padding*，补足长度  $n$ 。
2. 长度超过  $n$  的，只取前  $n$  个单词的词向量送进网络。

对  $n$  进行了讨论，训练 50 个 *epoch* 的结果如下

表1 不同序列长度的准确率（epoch 50）

序列长度 n	40	60	80	100	120
验证集准确率	0.3041	0.3109	0.3286	0.3027	0.2822

可见，序列长度  $n = 80$  时验证集准确率最高，为 32.86%

### 训练后期截图

```
-----Epoch 49-----
训练集准确率: 0.9965874189512001
验证集准确率: 0.3254437869822485

-----Epoch 50-----
训练集准确率: 0.9967011716528268
验证集准确率: 0.32862994993172506
```

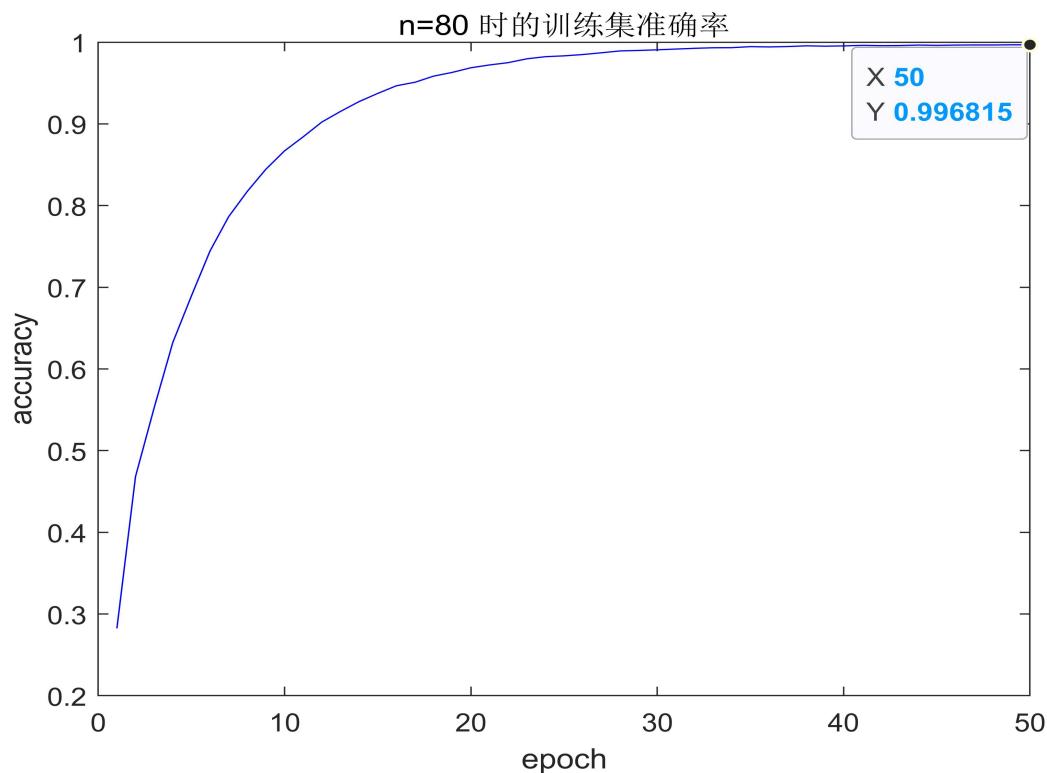
### 测试效果

```
测试集准确率: 0.31476931476931475

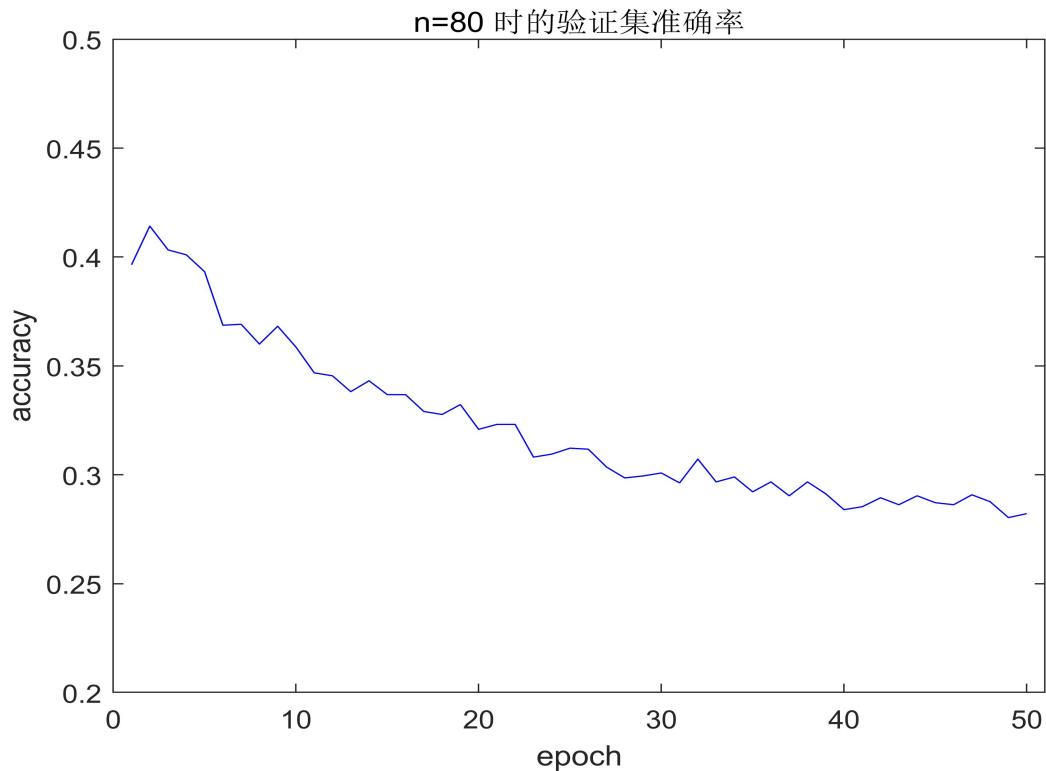
进程已结束，退出代码0
```

# 训练过程

## 训练集准确率



## 验证集准确率



过拟合啦！这是服务器上跑的结果，服务器上的代码忘记更新了，没有加上池化层。如果加上池化层，应该可以缓解这个过拟合的问题，从而达到更高的准确率。

## RNN (LSTM)

```
----- Epoch 39 -----
训练集准确率:  0.9796382664088272
验证集准确率:  0.6299499317250796

----- Epoch 40 -----
训练集准确率:  0.9790695029006939
验证集准确率:  0.6404187528447883
```

可见训练集准确率还没到最高，但程序已经停止了（只设置了 40 个epoch），实际上准确率还可以进一步提高

## 测试效果

```
测试集准确率:  0.6493311493311493

进程已结束, 退出代码0
```

测试集上准确率达到 64.93%

## 结果

1	alt.atheisrhyeah do you expect people to read the faq etc and actu
2	talk.politics.mideastalthough i realize that principle is not one of yo
3	comp.sys.ibm.pc.hardwarenotwithstanding all the legitimate fuss a
4	rec.sport.hockeywell i will have to change the scoring on my playo
5	comp.os.ms-windows.miscok i have a record that shows a iisi with
6	comp.graphicsarchive name graphics resources list part last modif
7	erates a bbs see above the entry for the graphics bbs you can call
8	rec.sport.baseballi have a roberto clemente topps baseball card fo
9	talk.politics.mideasti damon no matter what system or explanatic
10	rec.sport.hockeyand of course mike ramsey was at one time the ca
11	talk.religion.misci m sorry i thought we were discussing heresy i as
12	rec.sport.hockeyas i promised i would give you the name of the pa

结果放在 Result 文件夹中

(本来想放训练过程的准确率折线图的，数据丢了

## 2、参数说明

*TextCNN* 中讨论了序列长度  $n$  这个超参数

但是由于时间关系，卷积层中卷积核大小，池化层中卷积核大小，全连接层的特征数目等皆未作讨论。

同理，*RNN* 中，隐藏层的数量只简单调了一下，没有都跑出最终结果来比较，而且隐藏层的大小也未作讨论。

所以还有很多可以优化的地方，准确率也可以进一步提高。

## 四、参考资料

<https://pytorch.org/docs/stable/nn.html> PyTorch 官方文档

<https://zhuanlan.zhihu.com/p/123857569> LSTM 详解

<https://zhuanlan.zhihu.com/p/129808195> TextCNN 详解

原理部分很多图片都是从以上两个链接中截取的，但解释环节并没有照搬照抄，是之前自学的时候自己参考资料概况总结的，这个报告刚好用得上就把之前的自学报告截取过来了