本文主要内容

- 简略介绍循环神经网络（RNN, Recurrent Neural Network），其中涉及单层RNN结构、多层RNN结构、双向RNN结构、双向RNN+Attention结构
- 使用RNN进行文本分类任务，并给出模型的定义代码
- 本文代码【 https://github.com/540117253/Chinese-Text-Classification 】

一、RNN概述

循环神经网络RNN是特指一类专用于处理序列数据的模型，目前主流的RNN单元有LSTM(Long Short-Term Memory)和GRU(Gated Recurrent Unit)两种。相比最原始的RNN单元，LSTM通过增加记忆单元来缓解长序列数据训练时所产生的梯度消失问题，而GRU则是一种基于LSTM进行改进以进一步提升训练速度的变体单元。下文依次介绍单层RNN结构、多层RNN结构、双向RNN结构，这三种网络结构都可以根据实际需求来任意选择RNN单元（LSTM或GRU）。

1.1 单层RNN结构

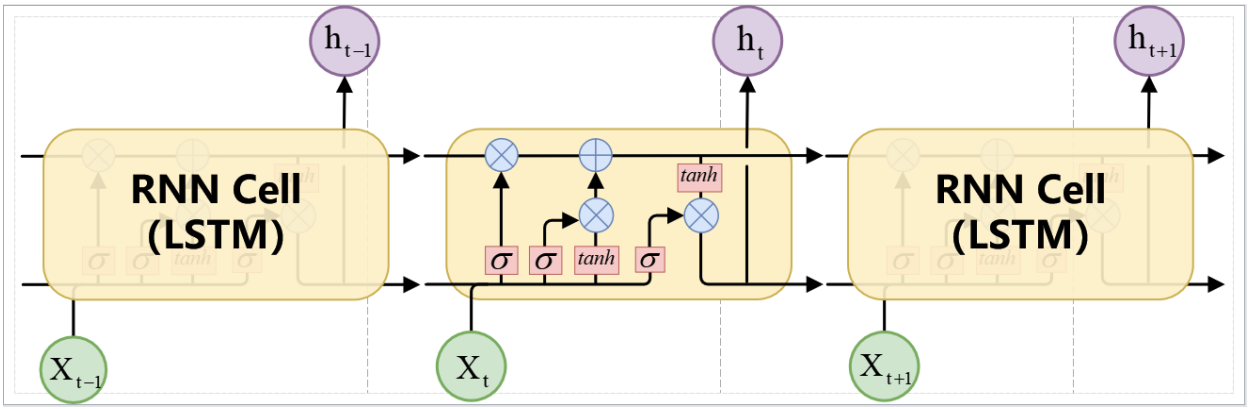


图1 RNN（以LSTM为例）处理过程示意图

假设给定一个句子 $S=\{w_t\}_{t=1}^{N}$，其中句子的长度为 N 个单词，句子中第 t 个单词为 w_t。在RNN处理前，需要先将每个单词 w_t 映射为一个向量 x_t 进行表达，即得到 $S=\{x_t\}_{t=1}^{N}$。在RNN的处理过程中，是依照从前往后的次序进行处理，即从第一个单词 x_1 到第 t 个单词 x_t 的次序进行运算。图1以LSTM作为基本单元为例（GRU单元同理），展示了RNN处理一个句子 $S=\{x_t\}_{t=1}^{N}$ 的整体示意图。处理过程的公式描述如下：

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

$$i_t = \sigma(w_i[h_{t-1}, x_t]) + b_i$$

$$\widetilde{C} = tanh(W_c[h_{t-1}, x_t] + b_c)$$

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

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh(C_t)$$

上述公式描述了句子 $S=\{x_t\}_{t=1}^{N}$ 中第 t 个单词 x_t （第 t 个时间步的输入）得到运算结果 h_t （第 t 个时间步的输出）的完成过程。 $[\cdot]$ 表示拼接操作， σ 表示sigmoid函数， $*$ 表示哈达玛积（Hadamard Product）。

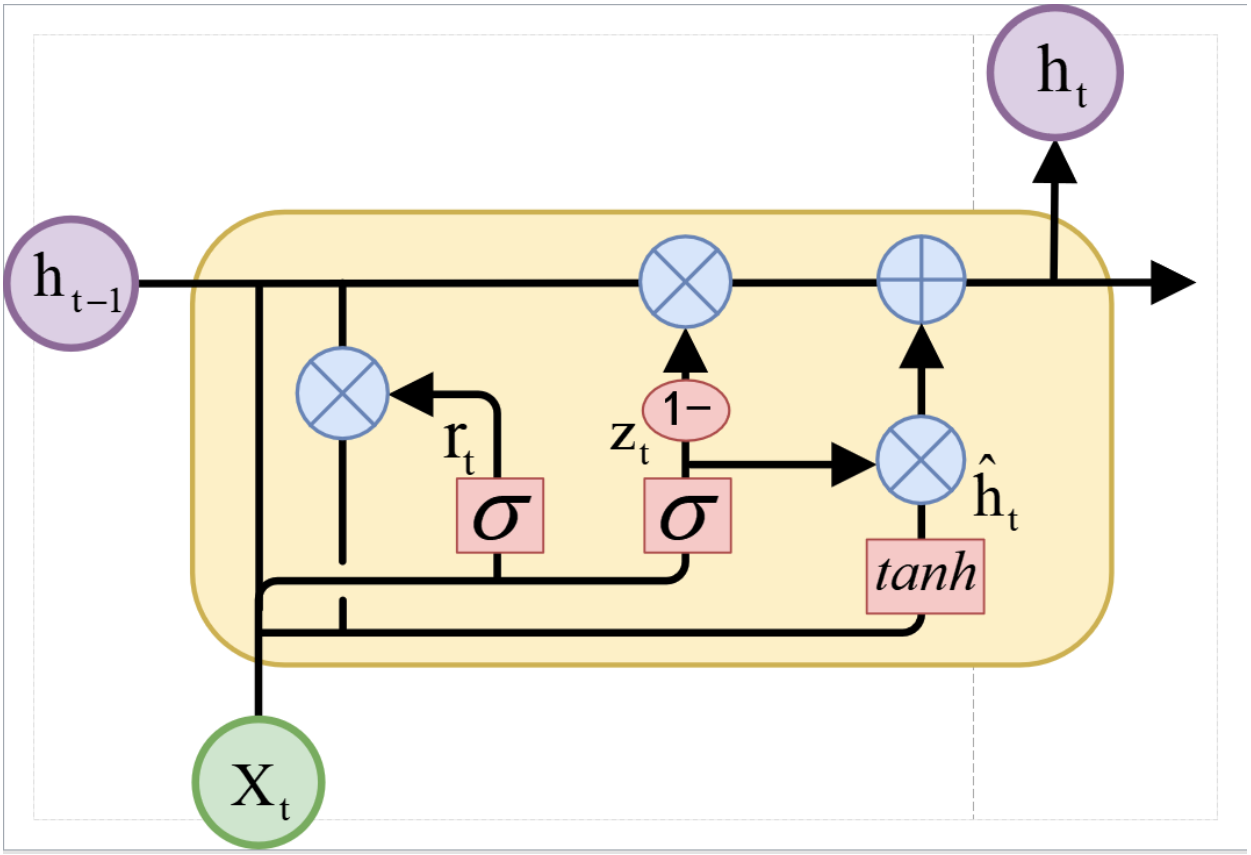


图2 GRU结构示意图

同样采用图1的结构，可以将LSTM单元替换为GRU单元，具体的GRU结构如图2所示。针对句子 $S=\{x_t\}_{t=1}^{N}$ 中第 t 个单词 x_t ，使用GRU计算出第 t 个时间步的输出 h_t 的过程可以描述为：

$$z_t = \sigma(W_z[h_{t-1}, x_t])$$

$$r_t = \sigma(w_r[h_{t-1}, x_t])$$

$$\widehat{h_t} = tanh(W[r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \widehat{h_t}$$

1.2 多层RNN结构

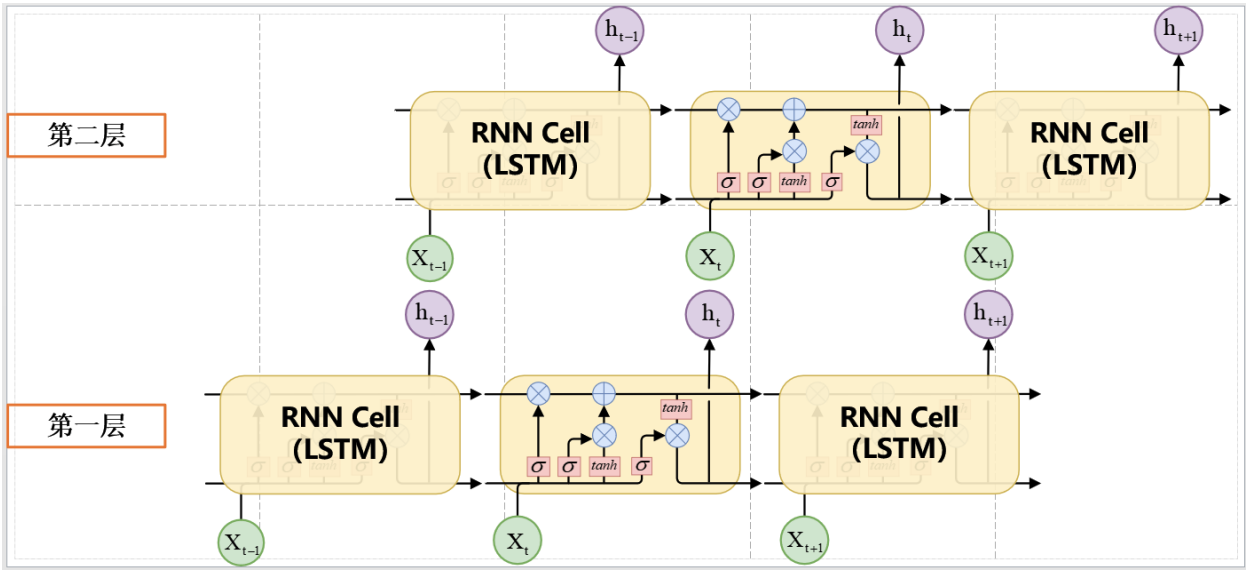


图3 多层RNN结构（以LSTM为例）

为了更全面细致地提取每个单词的信息，同样可以采取加深网络结构的方式堆叠多个RNN单元构建深层网络。多层RNN与单层RNN相似，同样是采取将一个句子 $S=\{xt\}_{t=1}^{N}$ 从前往后的次序进行处理。与单层RNN的区别在于，多层RNN的第 n 层的第 t 个时间步的输入 xt 就是第 $n-1$ 层的第 t 个时间步的输出 ht 。以LSTM为基本单元来搭建2层RNN为例，具体如图3所示（采用GRU为基本单元同理）。

1.3 双向RNN结构

图4展示了双向RNN结构（以LSTM为例）。给定一个句子 $S=\{wt\}_{t=1}^{N}$ ，其中句子的长度为 N 个单词，句子中第 t 个单词的向量为 wt 。双向RNN是一种RNN网络结构，其将读入的序列（这里是一个句子）从前往后和从后往前同时处理，最终将前向和后向的隐状态拼接作为最终的隐状态。

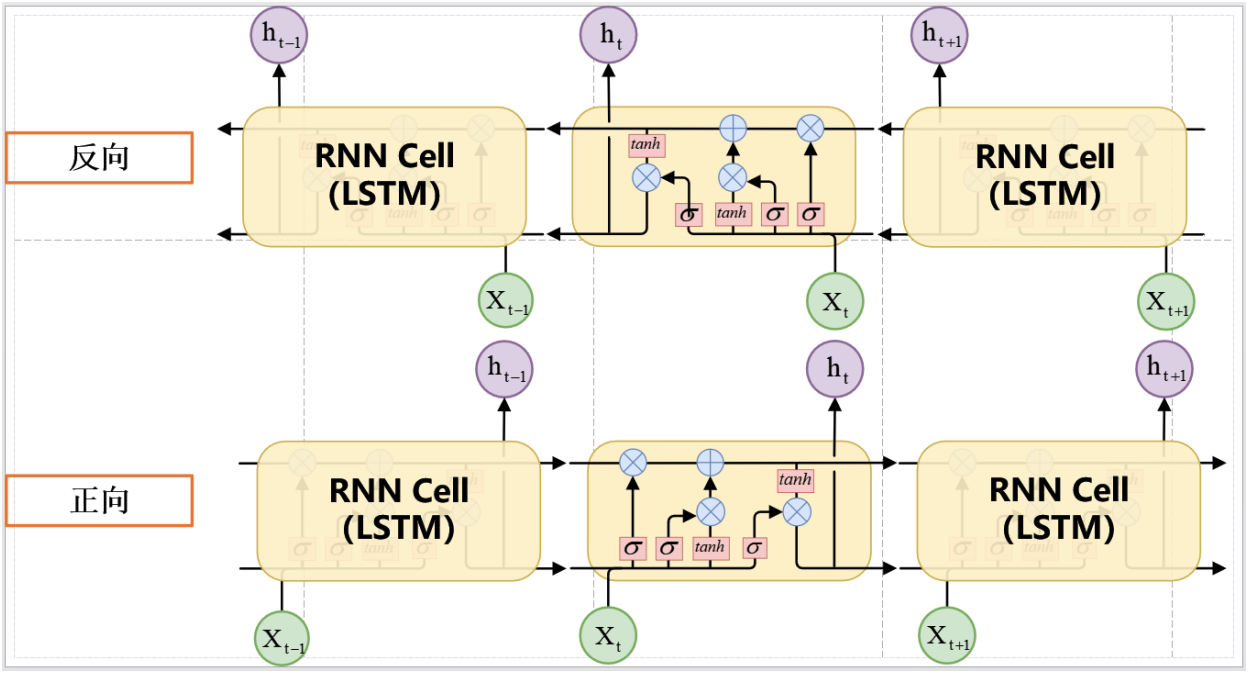


图4 双向RNN结构（以LSTM为例）

$$\overrightarrow{h_t}=RNN(\overrightarrow{h}_{t-1},w_t)$$

$$\overleftarrow{h_t} = RNN(\overleftarrow{h}_{t+1}, w_t)$$

$$h_t = [\overrightarrow{h_t}, \overleftarrow{h_t}]$$

其中 $RNN()$ 可以是LSTM单元或者是GRU单元，$[\cdot]$ 表示拼接操作。

1.4 双向RNN+Attention 结构

Attention机制的核心思想是赋予模型更关注与任务更相关的部分，而降低对任务次相关部分的关注程度。其原理可以看作为键值查询，通过用户给定的Query，来得到序列中与任务最相关位置的权值。

在文本分类任务中，基于RNN结构的模型都是将序列处理后的最后一个隐状态作为分类依据。由于RNN能捕获序列中的序列信息，因此最后一个隐状态包含了整个句子的信息，能够较好地应对文本分类任务。

但是，该句子为何被模型划分为该类别，我们不能直观地进行解释。为了提高分类结果的可解释性，这里采用Attention机制对句子序列中的每个位置都计算出一个注意力得分（权值），最终权值越高的位置表明对分类结果产生越重要的影响。

假设一个句子序列 $S = \{wt\}_{t=1}^{N}$ 进行RNN处理后，得到的序列为 $H = \{ht\}_{t=1}^{N}$，其中作为最后一个位置隐状态 hN 包含了整个序列的信息。这里 hN 将作为Query，对序列中的各个位置进行计算，得到各个位置与 hN 的关联度（与分类任务的关联度）。

$$query = W_q h_N + b_q$$

$$key = W_k H + b_k$$

$$attention = softmax(sum(W_a) tanh(keys + query))$$

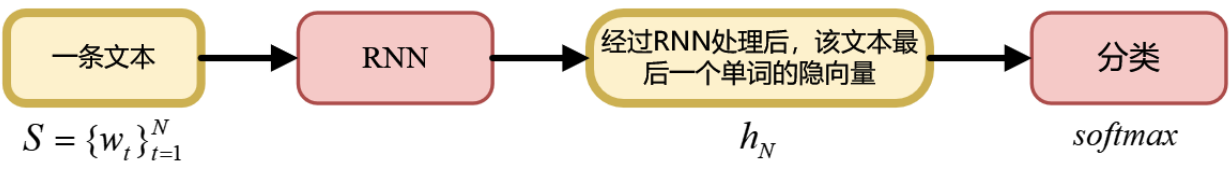
$$outputs = attention \times H$$

其中 $\times$ 表示 H 按照attention得分加权求和，得到的 $outputs$ 继续送入全连接层进行分类结果。

1.5 基于RNN的文本分类通用结构

在文本分类任务中，基于RNN结构的模型都是将序列处理后的最后一个隐状态 HN 作为分类依据。由于RNN能捕获序列中的序列信息，因此最后一个隐状态包含了整个句子的信息，能够较好地应对文本分类任务。

因此，基于RNN的文本分类通用结构为：



二、RNN文本分类实例

2.1 数据集介绍

1. 下载地址:

【https://github.com/skdjfla/toutiao-text-classfication-dataset 】

2. 格式:

```
6552431613437805063_!_102_!_news_entertainment_!_谢娜为李浩菲澄清网络谣言，之后她的两个行为给自己加分_!_佟丽娅,网络谣言,快乐大本营,李浩菲,谢娜,观众们
```

每行为一条数据，以_!_分割的个字段，从前往后分别是 新闻ID，分类code（见下文），分类名称（见下文），新闻字符串（仅含标题），新闻关键词

分类code与名称：

```
100 民生 故事 news_story
101 文化 文化 news_culture
102 娱乐 娱乐 news_entertainment
103 体育 体育 news_sports
104 财经 财经 news_finance
106 房产 房产 news_house
107 汽车 汽车 news_car
108 教育 教育 news_edu
109 科技 科技 news_tech
110 军事 军事 news_military
112 旅游 旅游 news_travel
113 国际 国际 news_world
114 证券 股票 stock
115 农业 三农 news_agriculture
116 电竞 游戏 news_game
```

2.2 预训练词向量

预训练词向量使用的是，基于ACL-2018模型在百度百科训练的词向量。

下载地址：【 https://github.com/Embedding/Chinese-Word-Vectors 】

2.3 数据预处理

1. 清除无用字符，并且进行分词处理
2. 建立整个数据集的字典,key=word, value=词语的编号
3. 对进行截断或补0处理，确保每条样本的长度为maxlen
4. 序列化样本的标签，例如“体育类新闻”的类别编号为1，“娱乐类新闻”的类别编号为2
5. 将处理好的数据转化为DataFrame格式，并保存到硬盘

2.4 模型的定义

该小节一共分别给出多层LSTM（Multi_LSTM），双向RNN（Bi_RNN），双向RNN+Attention（Bi_RNN_Attention）的代码定义。

2.4.1 Multi_LSTM

```
'''
    Text => Multi_Layer_LSTM => Fully_Connected => Softmax
'''
class Multi_LSTM:
    def __init__(self, rnn_output_dim, num_layers, embedded_size,
                 dict_size, maxlen, label_num, learning_rate):

        self.droput_rate = 0.5

        # print('model_Name:', 'Multi_LSTM')

        self.X = tf.placeholder(tf.int32, [None, maxlen], name='input_x')
        self.Y = tf.placeholder(tf.int64, [None])

        self.encoder_embeddings = tf.Variable(tf.random_uniform([dict_size,
embedded_size], -1, 1), trainable=False)
        encoder_embedded = tf.nn.embedding_lookup(self.encoder_embeddings, self.X)

        rnn_cells = [keras.layers.LSTMCell(units=rnn_output_dim) for _ in
range(num_layers)]
        outputs =keras.layers.RNN(rnn_cells,return_sequences=True, return_state=False)
(encoder_embedded)
        outputs = tf.nn.dropout(outputs, keep_prob = self.droput_rate)

        self.logits = keras.layers.Dense(label_num, use_bias=True)(outputs[:,-1]) # 取出
每条文本最后一个单词的隐藏层输出
        self.probability = tf.nn.softmax(self.logits, name='probability')

#         self.cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits =
self.logits, labels = self.Y))
        self.cost = tf.nn.sparse_softmax_cross_entropy_with_logits(
                                                                   labels = self.Y,
                                                                   logits = self.logits)
        self.cost = tf.reduce_mean(self.cost)
        self.optimizer = tf.train.AdamOptimizer(learning_rate =
learning_rate).minimize(self.cost)
        self.pre_y = tf.argmax(self.logits, 1, name='pre_y')
        correct_pred = tf.equal(self.pre_y, self.Y)
        self.accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

2.4.2 Bi_RNN

```
'''
    Text => Bidirectional GRU or LSTM => Fully_Connected => Softmax
'''
class Bi_RNN:
```

```
    def __init__(self, rnn_output_dim, embedded_size,
                 dict_size, maxlen, label_num, learning_rate, rnn_type):

        self.droput_rate = 0.5

        # print('model_Name:', 'Bi_RNN')

        '''
            Process the Reviews with Bi-RNN
        '''
        def bi_rnn(rnn_type, inputs, rnn_output_dim):
            if rnn_type == 'gru':
                h = keras.layers.Bidirectional(

keras.layers.GRU(rnn_output_dim,return_sequences=True,unroll=True),
                                merge_mode='concat'
                        )(inputs)

            elif rnn_type == 'lstm' :
                h = keras.layers.Bidirectional(

keras.layers.LSTM(rnn_output_dim,return_sequences=True,unroll=True),
                                merge_mode='concat'
                        )(inputs)
            return h # shape= (None, u_n_words, 2*rnn_output_dim) or # shape(H_d) =
(None, i_n_words, 2*rnn_output_dim)

        self.X = tf.placeholder(tf.int32, [None, maxlen], name='input_x')
        self.Y = tf.placeholder(tf.int64, [None])

        self.encoder_embeddings = tf.Variable(tf.random_uniform([dict_size,
embedded_size], -1, 1), trainable=False)
        encoder_embedded = tf.nn.embedding_lookup(self.encoder_embeddings, self.X)

        outputs = bi_rnn(rnn_type = rnn_type, inputs = encoder_embedded, rnn_output_dim =
rnn_output_dim)

        outputs = tf.nn.dropout(outputs, keep_prob = self.droput_rate)

        self.logits = keras.layers.Dense(label_num, use_bias=True)(outputs[:,-1]) # 取出
每条文本最后一个单词的隐藏层输出
        self.probability = tf.nn.softmax(self.logits, name='probability')

        self.cost = tf.nn.sparse_softmax_cross_entropy_with_logits(
                                                                   labels = self.Y,
                                                                   logits = self.logits)
        self.cost = tf.reduce_mean(self.cost)
        self.optimizer = tf.train.AdamOptimizer(learning_rate =
learning_rate).minimize(self.cost)
        self.pre_y = tf.argmax(self.logits, 1, name='pre_y')
        correct_pred = tf.equal(self.pre_y, self.Y)
        self.accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

2.4.3 Bi_RNN_Attention

```
'''
    Text => Bidirectional GRU or LSTM => Word_Attention => Fully_Connected => Softmax
'''
class Bi_RNN_Attention:
    def __init__(self, rnn_output_dim, embedded_size,
                 dict_size, maxlen, label_num, learning_rate, attention_size, rnn_type):

        self.droput_rate = 0.5

        # print('model_Name:', 'Bi_RNN_Attention')

        '''
            Process the Reviews with Bi-RNN
        '''
        def bi_rnn(rnn_type, inputs, rnn_output_dim):
            if rnn_type == 'gru':
                h = keras.layers.Bidirectional(

keras.layers.GRU(rnn_output_dim,return_sequences=True,unroll=True),
                               merge_mode='concat'
                        )(inputs)

            elif rnn_type == 'lstm' :
                h = keras.layers.Bidirectional(

keras.layers.LSTM(rnn_output_dim,return_sequences=True,unroll=True),
                               merge_mode='concat'
                        )(inputs)
            return h # shape= (None, u_n_words, 2*rnn_output_dim) or # shape(H_d) =
(None, i_n_words, 2*rnn_output_dim)

        self.X = tf.placeholder(tf.int32, [None, maxlen], name='input_x')
        self.Y = tf.placeholder(tf.int64, [None])

        self.encoder_embeddings = tf.Variable(tf.random_uniform([dict_size,
embedded_size], -1, 1), trainable=False)
        encoder_embedded = tf.nn.embedding_lookup(self.encoder_embeddings, self.X)

        outputs = bi_rnn(rnn_type = rnn_type, inputs = encoder_embedded, rnn_output_dim =
rnn_output_dim) # shape = [None, maxlen, rnn_output_dim]

        outputs = tf.nn.dropout(outputs, keep_prob = self.droput_rate)

        '''
            Word Attention Layer
        '''
        attention_w = tf.get_variable("attention_v", [attention_size], tf.float32)
        query = keras.layers.Dense(attention_size)(tf.expand_dims(outputs[:,-1], 1)) #
shape =[None, 1, attention_size]
        keys = keras.layers.Dense(attention_size)(outputs) # shape = [None, maxlen,
attention_size]
        self.attention = tf.reduce_sum(attention_w * tf.tanh(keys + query), 2) # shape =
[None, maxlen]
        self.attention = tf.nn.softmax(self.attention, name='attention')
        outputs = tf.squeeze(
                               tf.matmul(
                                   tf.transpose(outputs, [0, 2,
```

```
1]),tf.expand_dims(self.attention, 2)
                                ), # shape = [None, rnn_output_dim, 1]
                            2) # shape = [None, rnn_output_dim]

        self.logits = keras.layers.Dense(label_num, use_bias=True)(outputs)
        self.probability = tf.nn.softmax(self.logits, name='probability')

        self.cost = tf.nn.sparse_softmax_cross_entropy_with_logits(
                                                                labels = self.Y,
                                                                logits = self.logits)
        self.cost = tf.reduce_mean(self.cost)
        self.optimizer = tf.train.AdamOptimizer(learning_rate =
learning_rate).minimize(self.cost)
        self.pre_y = tf.argmax(self.logits, 1, name='pre_y')
        correct_pred = tf.equal(self.pre_y, self.Y)
        self.accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

2.5 训练模型

1. 将预处理好的数据集切分为80%的训练集，10%作为验证集，10%作为测试集
2. 选定Multi_LSTM, Bi_RNN, Bi_RNN_Attention其中一个模型
3. 每使用一次训练集进行训练后，就使用验证集进行测试。
4. 当验证集的准确率连续下降5次，就停止步骤3，然后使用测试集的结果作为模型的最终性能。