【程序喵笔记】RNN实现文本分类

原创 毛毛喵 毛毛喵 2020-04-21

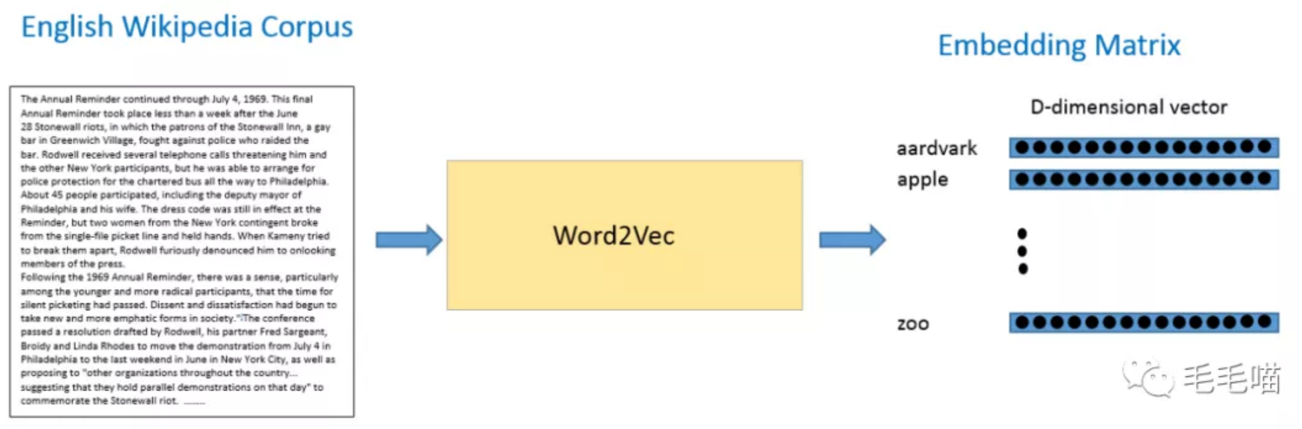
RNN实现文本分类

目标：运用RNN网络对影评数据集进行分类

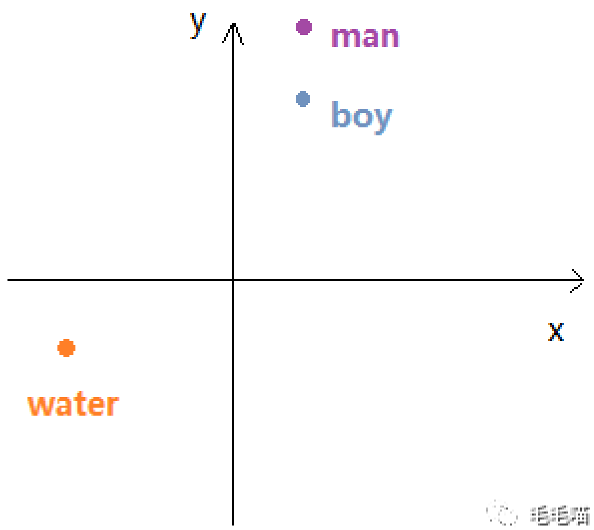
词向量模型 word2vec

词向量概述

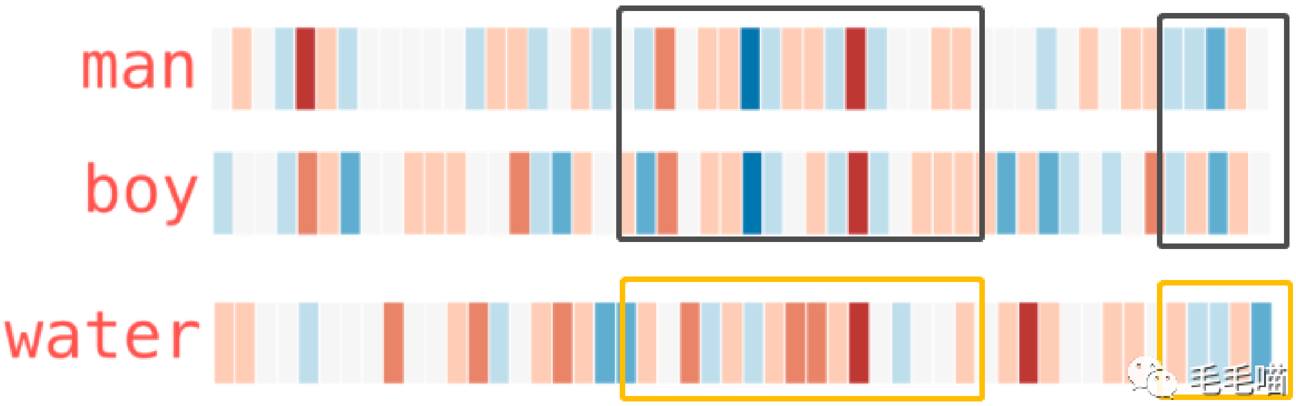
一句话很长，由千千万万个词构成，计算机想要明白一句话，就要先认识一个个词，计算机擅长处理什么？是数字啊，所以我们就把“词”转化成一个个数字组成“向量”。我们希望得到一个 大表 `embedding maxtrix` ，每个词对应的向量都能找到。



这个词向量不能随便设定的吧，首先需要相近的词汇离得近一点，举个栗子，一个二维向量（a，b）代表一个词。比如“man”这个词在空间中（1，5），boy（1，4），而water（-1，-4），我们看看在坐标系下这些点。



很显然，man和boy离得很近，离water很远。但是二维远远不能满足对一个词的描述，通常情况下，我们用上百维的向量来描述一个词，两个词之间的关联程度用余弦距离或者欧式距离表示。还是这三个词，用更高维度标签，不同颜色代表不同数字范围，对应数字越大表示越深红。



圈出的部分明显可以看出，相似的词是有很大的相似程度的。但是词向量这么多维的数字人为的编造也太麻烦了，我们可以用一个神经网络来构造每个词对应的向量。

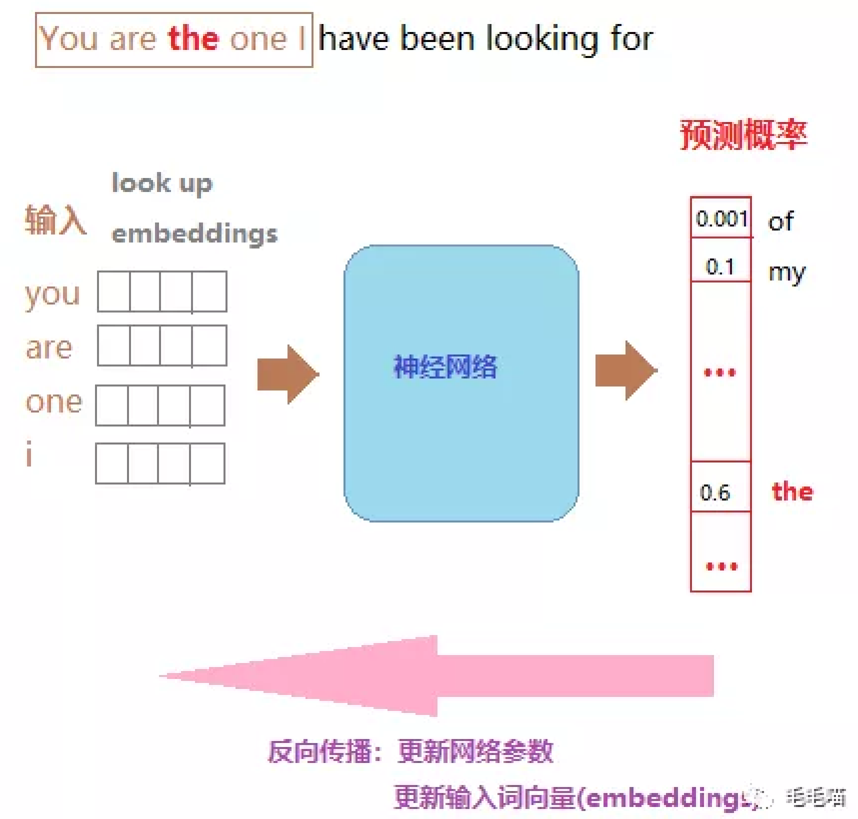
词向量模型

既然是神经网络，那肯定就要有输入和输出，词不可能单个的蹦出来，一定是有上下文关系的，我们就利用这一点，就可以构建词向量模型的网络啦，假设我们的词向量是一个4维向量。

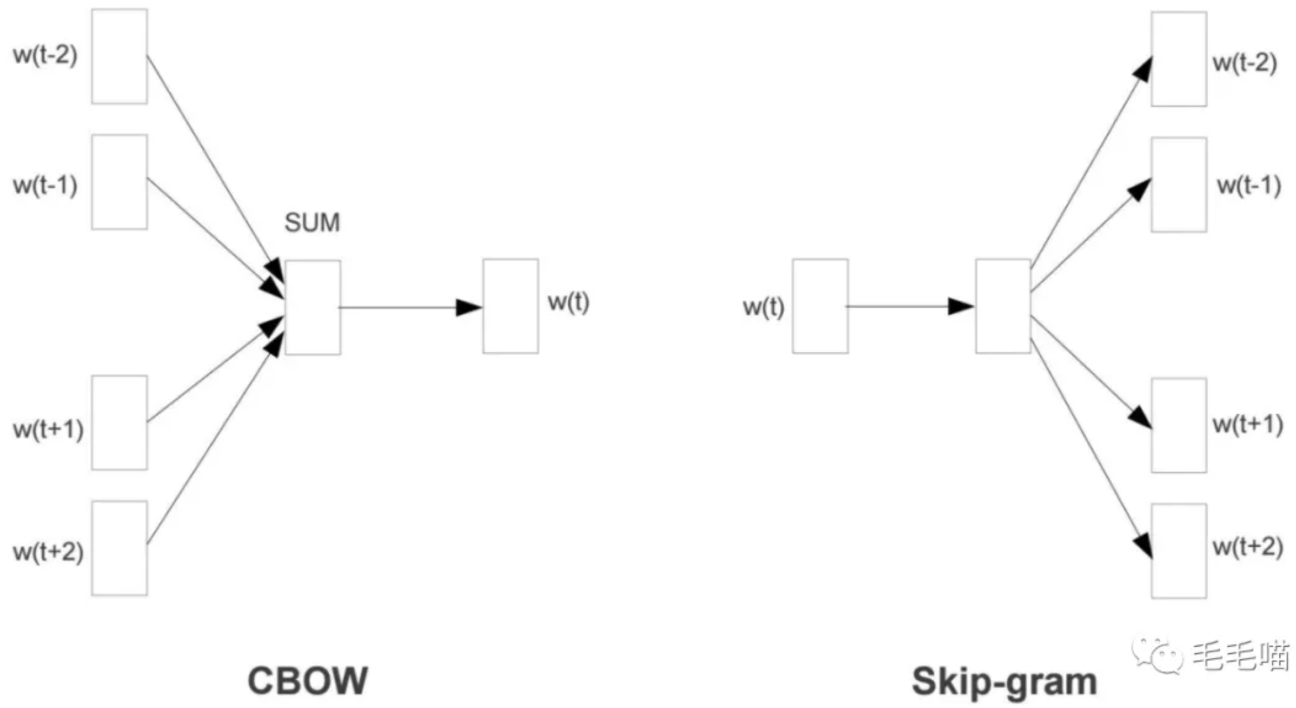
举个甜甜的表白词：You are the one I have been looking for.（你就是我一直在追寻的幸福）,依次先圈出5个词(滑动窗口)，找到中间的词 `w(t)` 和这个词的上下文 `w(t-2), w(t-1), w(t+1), w(t+2)` 。

可以将上下文作为输入，中间词作为输出，来判断判断输出这个词是不是上下文所对应的中间词。先建立一个词向量大表 `embedding` ，里面有所有词和对应的向量（随机初始化）.

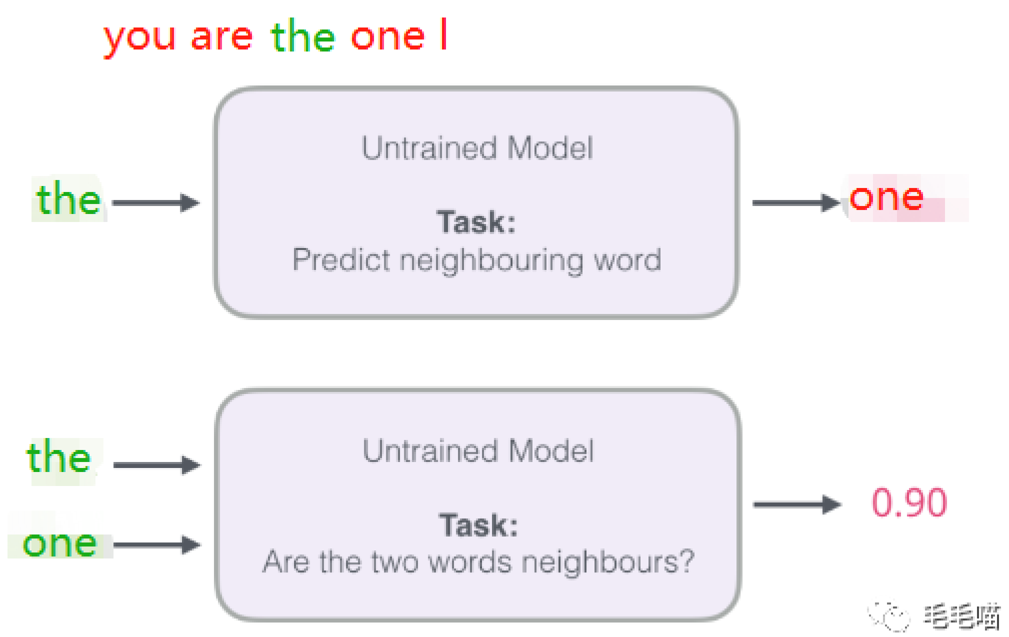
前向传播：根据输入词,找到大表中对应的向量，经过神经网络得到各种词的概率，判断输出是否为中间词。反向传播：不仅更新网络模型，也会更新词向量的大表.



常见有两种模型，CBOW是上下文为输入，中间词的输出。而skip-gram正好相反，但是原理相近。



这样虽然可以找到词向量，但是最后的预测概率是一个超多的分类问题，需要的参数可是太多了，那我们能不能简化一点，比较输入和输出是否相邻，那就变成一个二分类问题，是相邻的为1，不是为0，以skip-gram为例。



原先的输入和输出左边，右边是改进后增加 `target` 。

You are the one I have been looking for.

You are the one I have been looking for.

input	output
the	you
the	are
the	one
the	I
one	are
one	the
one	I
one	have

input1	input2	target
the	you	1
the	are	1
the	one	1
the	I	1
one	are	1
one	the	1
one	I	1
one	have	

这样看上去没啥大问题，但是，所有标签全为1啊，并没有0，这样神经网络还学啥，任意两个词都是相邻的了。所以，在这里，我们增加一个**负采样**，就是人为的创造一些与不相邻的数据，让target为0.

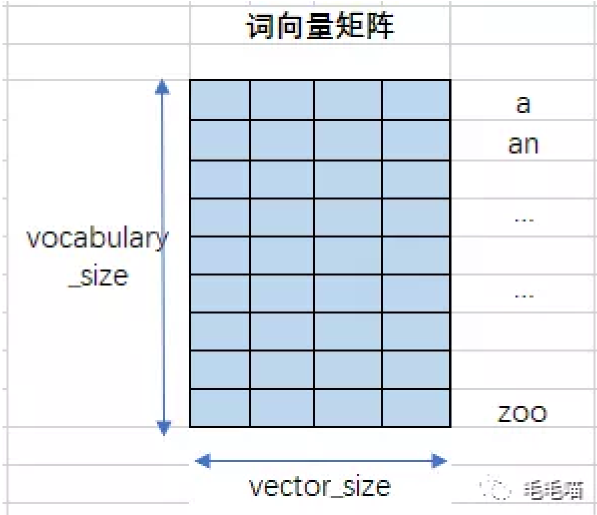
input1	input2	target
the	you	1
the	are	1
the	one	1
the	I	1
one	are	1
one	the	1
one	I	1
one	have	1

input1	input2	target
the	you	1
the	are	1
the	one	1
the	I	1
the	his	0
the	sick	0
one	are	1
one	the	1
one	I	1
one	have	1
one	what	0
one	hello	

负采样的个数，根据模型而定，滑动窗口的大小和每个窗口取几个值也是根据模型而定，大体的思路就是这样的。总结一下，三大步：

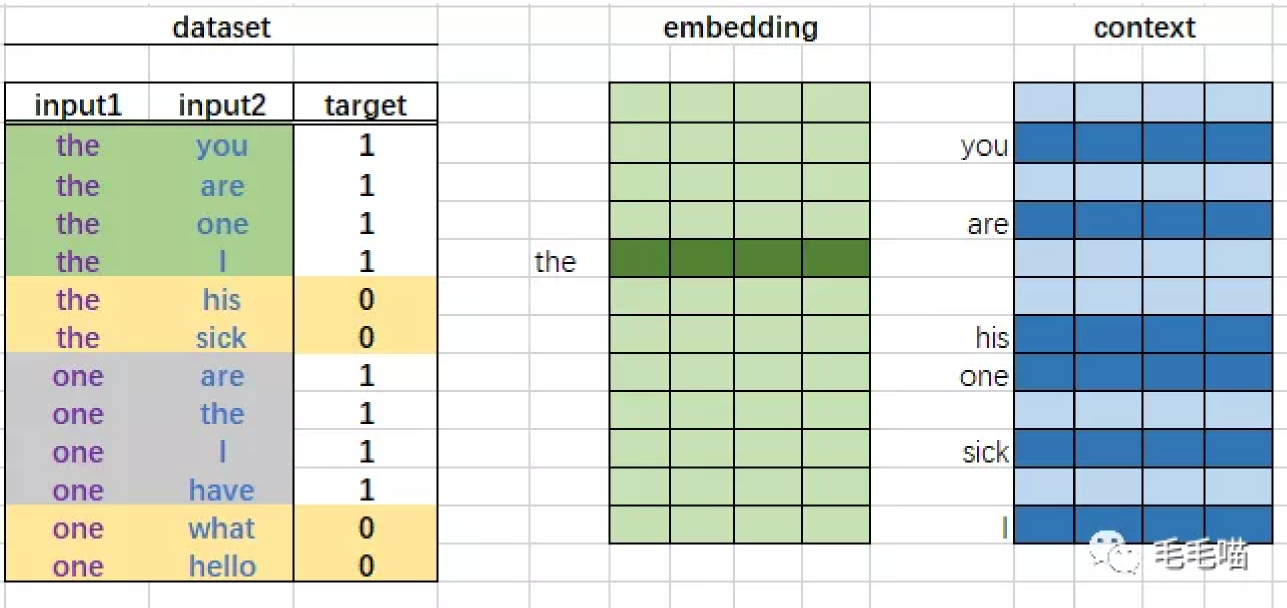
1.初始化词向量矩阵

词汇个数： `vocabulary_size` ，每个向量维度 `vector_size` ，相当于一个大表，每个词对应向量，这也是最终所需要的



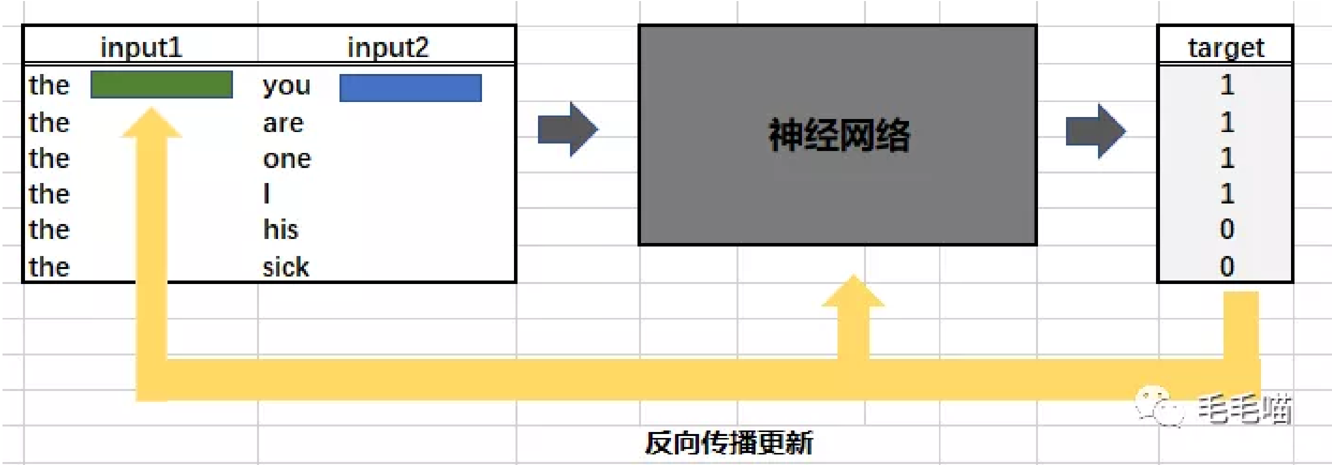
2.构建数据

将文本做滑动窗口处理，找到中间词和上下文对应词向量，比如the用一个4维向量表示，you、are、his等一些词也在词向量表格找到对应4维向量。



3.构建网络更新参数

通过神经网络返向传播来计算更新，此时不仅更新网络的权重参数矩阵，也会更新词向量矩阵（绿色的4维向量）。



模型实现

1.数据预处理

构建语料库，根据语料库数目限制和最小词频限制，构建出 `47135` 个词作为语料库。

```
max_vocabulary_size = 50000# 语料库最大词语数
min_occurrence = 10# 最小词频

# 读取数据
data_path = 'text8.zip'
with zipfile.ZipFile(data_path) as f:
    text_words = f.read(f.namelist()[0]).lower().split()#总共17005207词，我们只选取最常见的

# 创建计数器，从多到少计算词频
# 第一个词为‘UNK’，文本中不在语料库中的词汇用这个代替
count = [('UNK', -1)]
# 基于词频返回max_vocabulary_size个常用词
count.extend(collections.Counter(text_words).most_common(max_vocabulary_size - 1))

# 剔除掉出现次数少于'min_occurrence'的词
for i in range(len(count) - 1, -1, -1):
    if count[i][1] < min_occurrence:
        count.pop(i)
    else:
        break
```

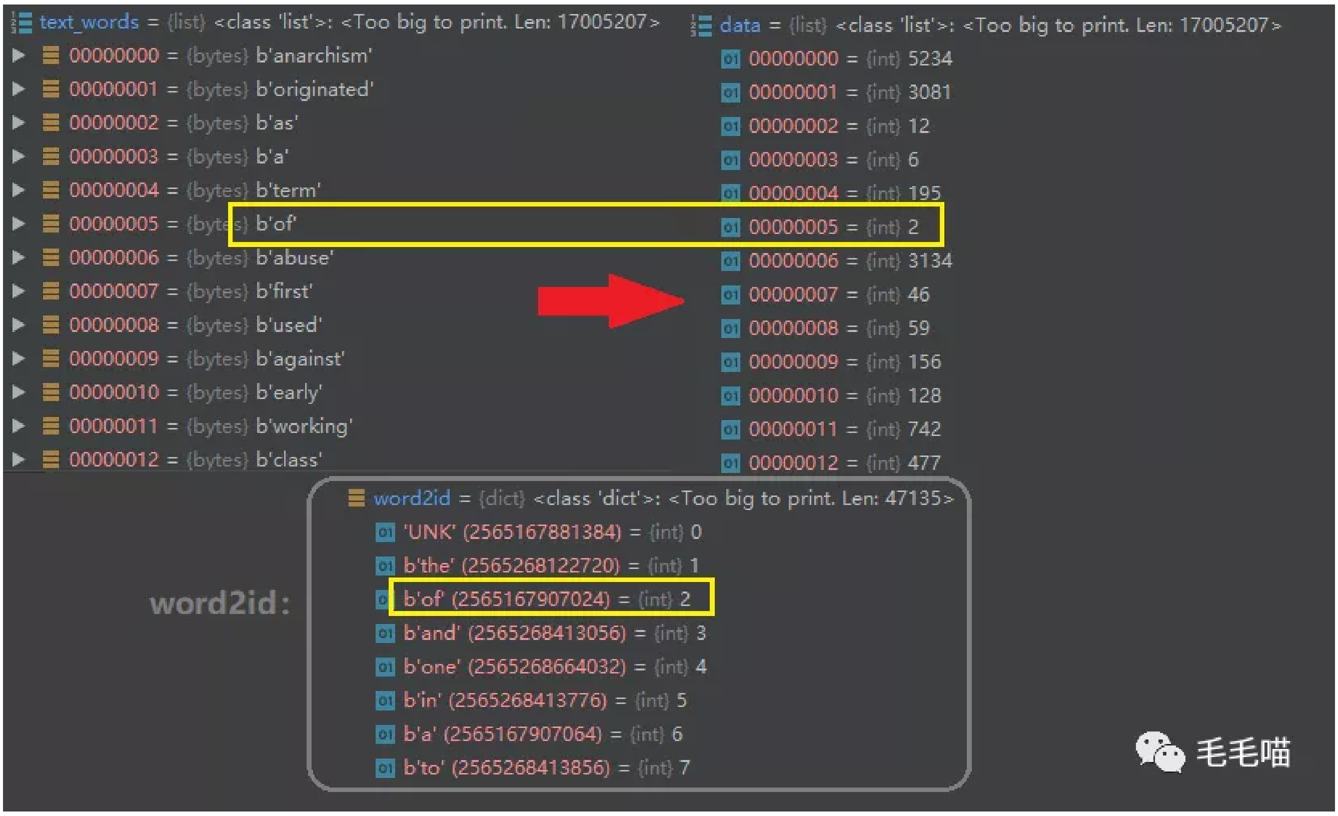
词-ID映射，语料库中每个词给出对应ID，并将文本转化为ID

```
# 每个词都分配一个ID
word2id = dict()
```

```
for i, (word, _)in enumerate(count):
    word2id[word] = i

# 所有词转换成ID
data = list()
unk_count = 0
for word in text_words:
    index = word2id.get(word, 0)
    if index == 0:
        unk_count += 1
    data.append(index)
count[0] = ('UNK', unk_count)
# id2word是word2id反映射
id2word = dict(zip(word2id.values(), word2id.keys()))
```

文本 `text_words` 根据 `word2id` 变成了一个个数字索引 `data`



构建词向量矩阵，将索引转化维词向量

```
embedding_size = 200# 词向量由200维向量构成
embedding = tf.Variable(tf.random.normal([vocabulary_size, embedding_size])) #维度: 47135, 200

#通过tf.nn.embedding_lookup函数将索引转换成词向量
def get_embedding(x):
    x_embed = tf.nn.embedding_lookup(embedding, x)
    return x_embed
```

2.滑动窗口提取数据对

滑动窗口大小为 `2*skip_window + 1` ，即为7，中间的作为输入，左右两边随机选择 `num_skips` 个词作为标签，构成一对数据。

```
skip_window = 3# 左右窗口大小
num_skips = 2# 一次制作多少个输入输出对
batch_size = 128#一个batch下有128组数据
```

在一个 `batch` 下取数据，即一次取128组数据对，每个窗口取2组，在 `128/2` 个窗口下取即可

```
def next_batch(batch_size, num_skips, skip_window):
    global data_index
    assert batch_size % num_skips == 0
    assert num_skips <= 2 * skip_window
    batch = np.ndarray(shape=(batch_size), dtype=np.int32)
    labels = np.ndarray(shape=(batch_size, 1), dtype=np.int32)
    # 数据窗口为7
    span = 2 * skip_window + 1
    # 创建队列，长度为7
    buffer = collections.deque(maxlen=span)#创建一个长度为7的队列
    if data_index + span > len(data):#如果文本被滑完，从头再来
        data_index = 0
    # 比如一个队列为deque([5234, 3081, 12, 6, 195, 2, 3134], maxlen=7)，数字代表词ID
    buffer.extend(data[data_index:data_index + span])
    data_index += span
    for i in range(batch_size // num_skips):
        context_words = [w for w in range(span) if w != skip_window]#上下文为[0, 1, 2, 4, 5, 6]
        words_to_use = random.sample(context_words, num_skips)#在上下文里随机选2个候选词
        for j, context_word in enumerate(words_to_use):
            batch[i * num_skips + j] = buffer[skip_window]#输入都为当前窗口的中间词，即3
            labels[i * num_skips + j, 0] = buffer[context_word]#标签为当前候选词
        # 窗口右移，如果文本读完，从头再来
        if data_index == len(data):
            buffer.extend(data[0:span])
            data_index = span
        else:
            buffer.append(data[data_index])
            data_index += 1
```

```
        data_index = (data_index + len(data) - span) % len(data)
        return batch, labels
```

3.迭代优化

计算损失，损失为 `nce` 损失，同时加入负采样。先分别计算出正样本和采样出的负样本对应的输出（0到1之间数字），再通过 sigmoid交叉熵来计算损失。

```
num_sampled = 64# 负采样个数
# nce权重和偏差
nce_weights = tf.Variable(tf.random.normal([vocabulary_size, embedding_size]))
nce_biases = tf.Variable(tf.zeros([vocabulary_size]))
# 定义nce损失，x_emded为转化为词向量的中间词，y为上下文词
def nce_loss(x_embed, y):
    with tf.device('/cpu:0'):
        y = tf.cast(y, tf.int64)
        loss = tf.reduce_mean(
            tf.nn.nce_loss(weights=nce_weights,
                           biases=nce_biases,
                           labels=y,
                           inputs=x_embed,
                           num_sampled=num_sampled,
                           num_classes=vocabulary_size))
        return loss
```

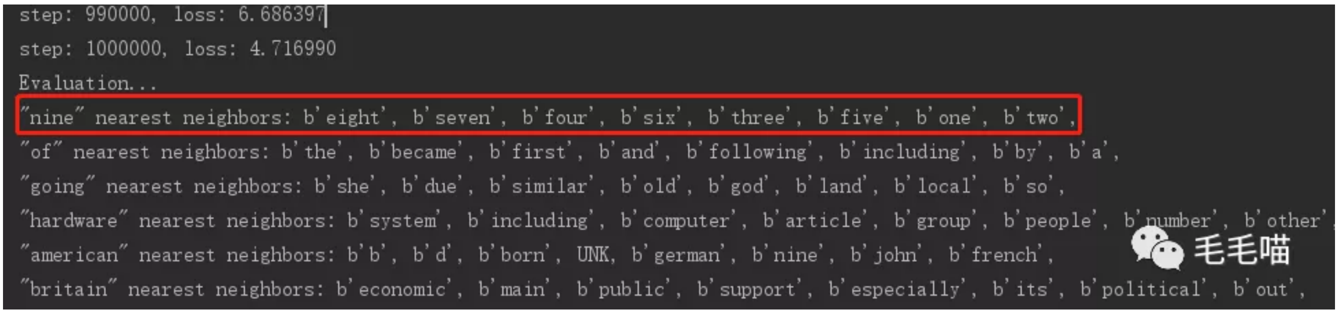
计算梯度，更新 `embedding` 和 `nce` 参数

```
with tf.GradientTape() as g:
    x, y = next_batch(batch_size, num_skips, skip_window)
    emb = get_embedding(x)
    loss = nce_loss(emb, y)

# 计算梯度
gradients = g.gradient(loss, [embedding, nce_weights, nce_biases])
# 更新
optimizer = tf.optimizers.SGD(learning_rate)
optimizer.apply_gradients(zip(gradients, [embedding, nce_weights, nce_biases]))
```

4.验证

找一些词，看看这个词邻近的词的8个词（计算向量余弦相似度），"nine"很明显，邻近单词都是数字。



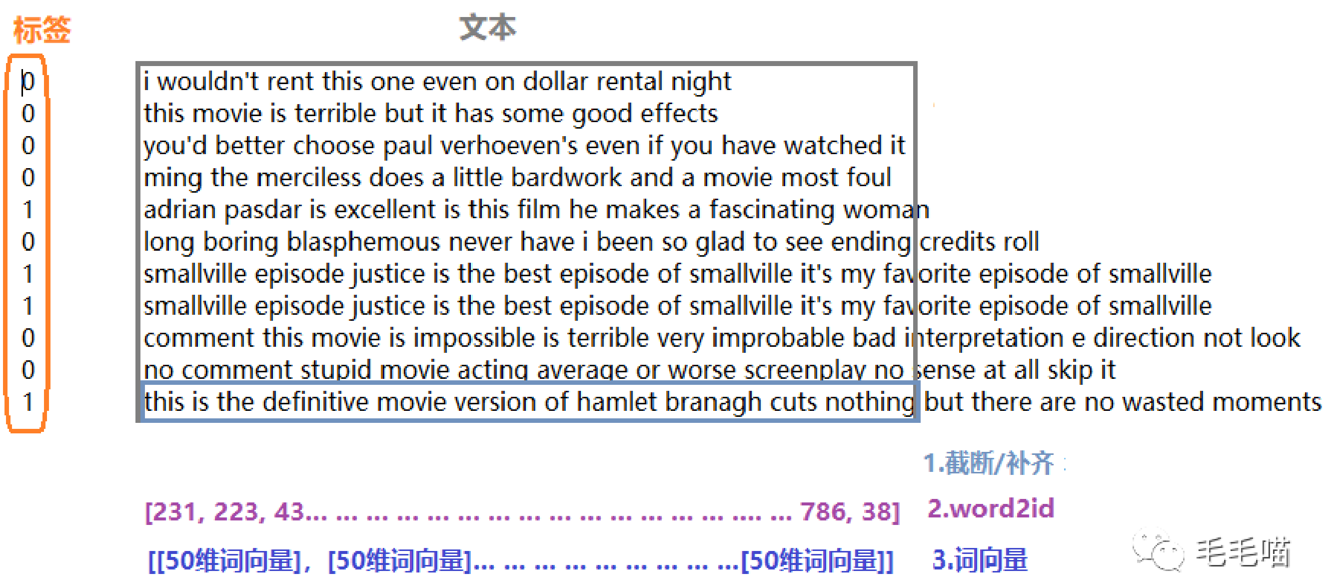
5.补充

大多数情况下，词向量模型不需要我们自己训练，有网上训练好的，直接down下来，但是一些特殊任务有大量特殊词汇，需要自己训练。

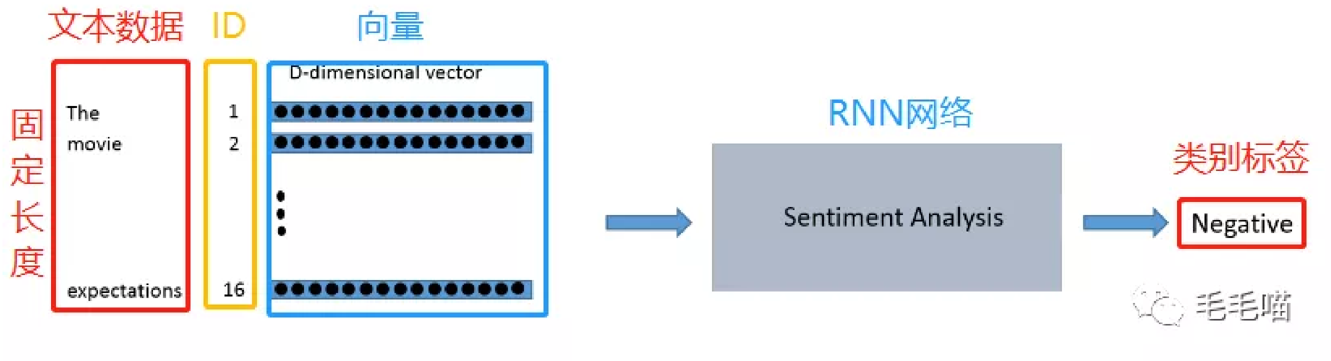
文本分类任务

整体思路

- 数据集：一条影评对应一个标签，标签为0/1，消极评价/积极评价
- 词向量模型：加载训练好的词向量模型。https://nlp.stanford.edu/projects/glove/ 50维词向量
- 序列网络模型：双向RNN模型进行识别



将文本以固定长度 `max_len` 截取，每个词转化为id，再转化为词向量，输入RNN网络进行二分类。



数据准备

word2id语料表

根据词频构建语料表，即训练集出现的词

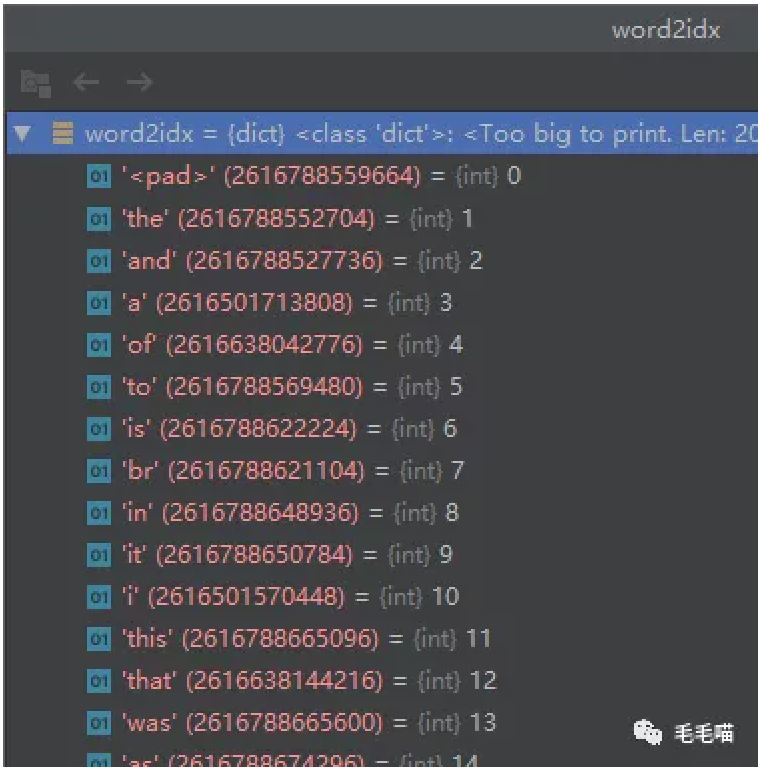
```
# 构建语料表，基于词频来进行统计
counter = Counter()
with open('./data/train.txt',encoding='utf-8') as f:
    for line in f:
        line = line.rstrip()
        label, words = line.split('\t')
        words = words.split(' ')
        counter.update(words)

# <pad>用作文本补齐，长度不够填充
words = ['<pad>'] + [w for w, freq in counter.most_common() if freq >= 10]
print('Vocab Size:', len(words))

# 保存文件
Path('./vocab').mkdir(exist_ok=True)
with open('./vocab/word.txt', 'w',encoding='utf-8') as f:
    for w in words:
        f.write(w+'\n')
```

读取文件并标号得到word2id映射字典

```
word2idx = {}
with open('./vocab/word.txt',encoding='utf-8') as f:
    for i, line in enumerate(f):
        line = line.rstrip()
        word2idx[line] = i
```

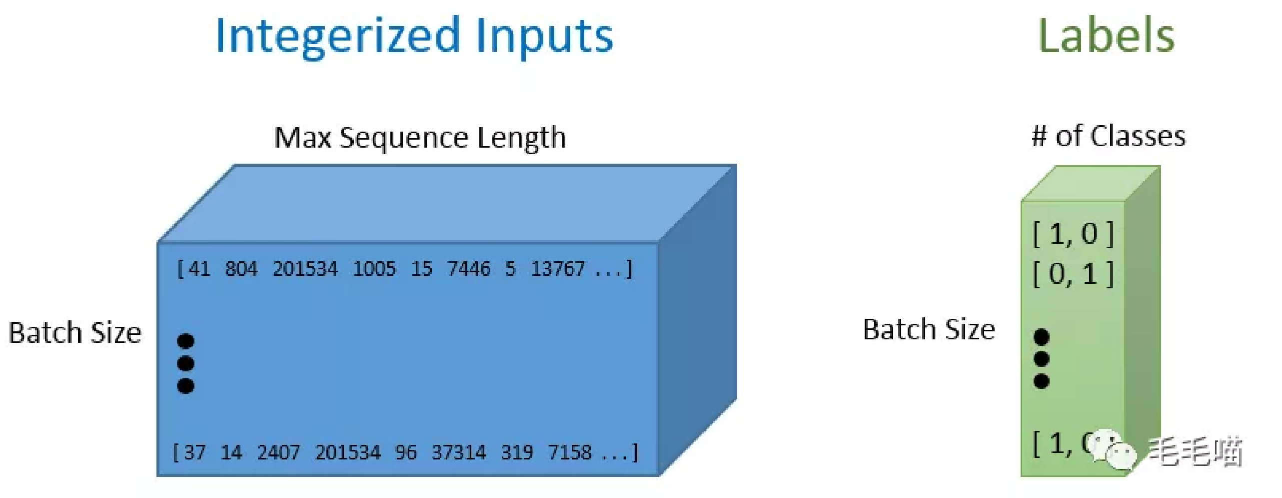


构建训练数据

运用数据生成器 `data_generator` 生成数据和标签，并运用 `tf.data.Dataset.from_generator(data_generator,output_data_type,output_data_shape)` 不断读取

```
# 定义生成器
def data_generator(f_path, params):
    with open(f_path,encoding='utf-8') as f:
        print('Reading', f_path)
        for line in f:
            line = line.rstrip()
            # 标签和文本分别保存
            label, text = line.split('\t')
            text = text.split(' ')
            # 每个词对应id，利用word2id字典查找
            x = [params['word2idx'].get(w, len(word2idx)) for w in text]
            # 文本长度保持一致
            if len(x) >= params['max_len']:#截断操作，max_len = 1000
                x = x[:params['max_len']]
            else:
                x += [0] * (params['max_len'] - len(x))#补齐操作
            y = int(label)
            yield x, y
```

把数据放入batch中，训练数据加入 `shuffle` 操作



词向量模型

embedding矩阵

下载词向量表是一个词对应50维向量，但是我们的语料表可能有一部分单词出现在下载向量的表中，没有出现记为 `unknown` ，找到到训练集语料表每个词索引所对应的向量，即 `embedding`

```
#一个大表，里面有20598个词和一个‘unknown’，每个词有50维向量【20599*50】
embedding = np.zeros((len(word2idx)+1, 50))

with open('./data/glove.6B.50d.txt',encoding='utf-8') as f: #下载的词向量映射
    count = 0
    for i, line in enumerate(f):
        # 提取词和词向量
        line = line.rstrip()
        sp = line.split(' ')
        word, vec = sp[0], sp[1:]
        # 判断词是否出现在我们的语料表中
        if word in word2idx:
            count += 1
            embedding[word2idx[word]] = np.asarray(vec, dtype='float32') #将词转换成对应的向量
np.save('./vocab/word.npy', embedding)
```

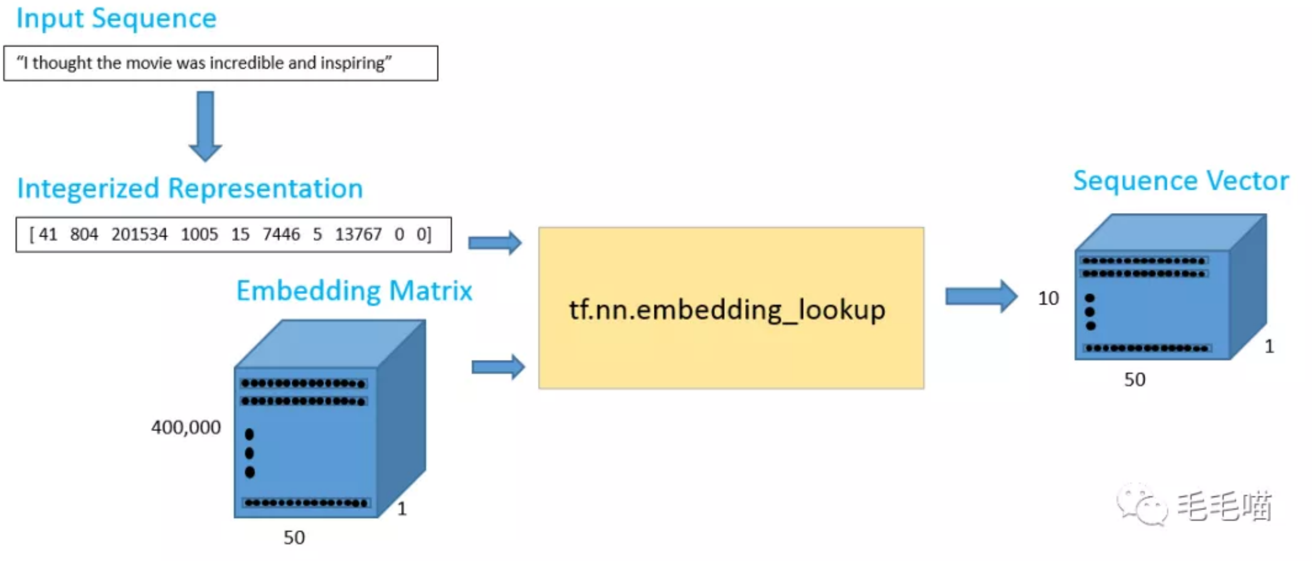
`embedding矩阵` 为一个 `20599*50` 维度的数组，每行为语料表单词所对应的向量。

embedding

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.41...	0.24...	-0.41...	0.121...	0.34...	-0.0...	-0.4...	-0.1...	-0.00...	-0.6...	0.2...	-0.1...	-0.55...	0.14...
2	0.26...	0.14...	-0.27...	0.016...	0.11...	0.69...	-0.5...	-0.4...	-0.33...	-0.1...	0.2...	0.30...	-0.45...	-0.41...
3	0.21...	0.46...	-0.46...	0.100...	1.01...	0.74...	-0.5...	-0.2...	0.16...	0.13...	-0.2...	-0.4...	-0.21...	0.51...
4	0.70...	0.57...	-0.47...	0.180...	0.54...	0.72...	0.18...	-0.5...	0.10...	-0.1...	0.0...	-0.3...	-0.11...	-0.83...
5	0.68...	-0.0...	0.301...	-0.17...	0.42...	0.03...	-0.4...	0.13...	-0.29...	-0.0...	0.1...	0.22...	-0.10...	-0.43...
6	0.61...	0.64...	-0.46...	0.375...	0.74...	0.53...	0.00...	-0.6...	0.26...	0.11...	0.4...	0.20...	-0.05...	-0.34...
7	-0.1...	0.38...	0.501...	-0.58...	-1.0...	0.31...	0.12...	-1.3...	-1.25...	-0.9...	0.2...	1.25...	-0.54...	-0.25...
8	0.33...	0.24...	-0.60...	0.109...	0.03...	0.15...	-0.5...	-0.0...	-0.09...	-0.3...	0.0...	-0.8...	-0.36...	-0.67...
9	0.61...	-0.2...	-0.10...	-0.05...	0.50...	0.34...	-0.3...	-0.1...	-0.03...	0.10...	0.0...	0.24...	-0.43...	-0.33...
10	0.11...	0.15...	-0.08...	-0.74...	0.75...	-0.4...	-0.3...	0.51...	-0.98...	0.00...	-0.1...	0.83...	-1.07...	-0.51...
11	0.53...	0.40...	-0.40...	0.154...	0.47...	0.20...	-0.2...	-0.3...	-0.10...	0.10...	-0.1...	0.10...	-0.49...	-0.25...
12	0.88...	-0.1...	0.135...	0.098...	0.51...	0.49...	-0.4...	-0.3...	0.01...	0.12...	0.0...	0.35...	-0.60...	-0.18...
13	0.08...	-0.1...	-0.24...	-0.33...	0.56...	0.39...	-0.9...	0.03...	-0.61...	-0.3...	0.5...	0.12...	[illegible]	[illegible]
14	0.20...	0.12...	-0.30...	-0.23...	0.30...	0.33...	-0.5...	-0.4...	-0.48...	0.03...	0.3...	0.54...	-0.20...	-0.08...

embedding_lookup

有了 `embedding矩阵` 和文本序列的词id，利用 `tf.nn.embedding_lookup` 就可以将输入的序列转化为序列向量

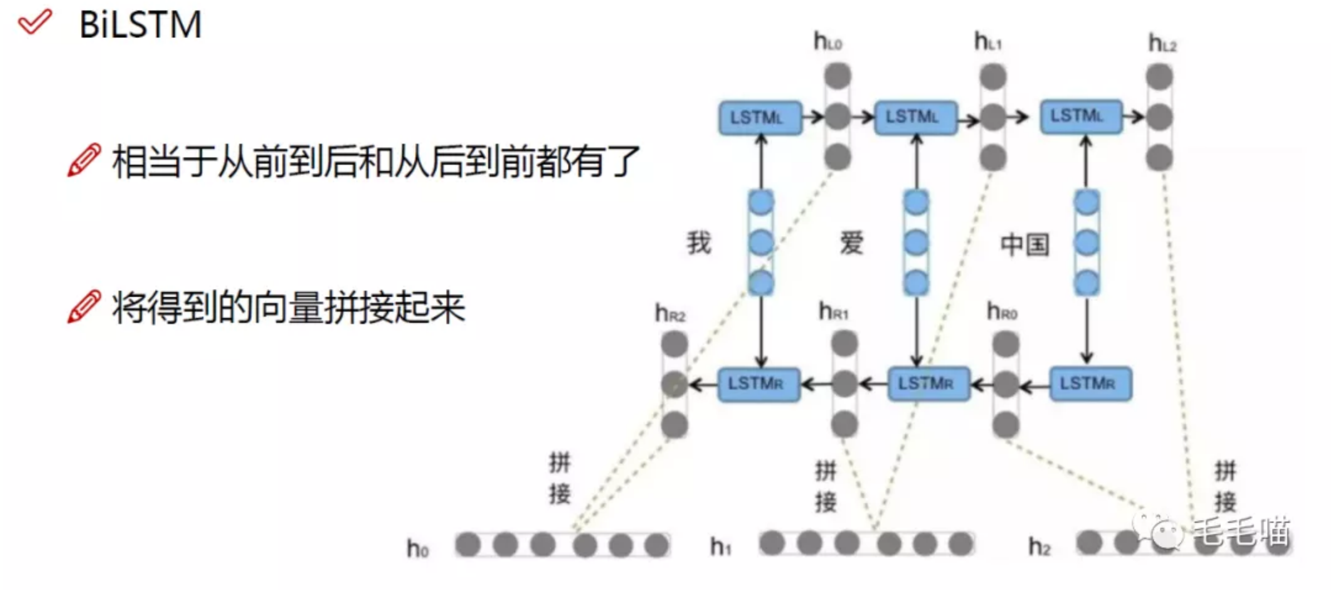


词向量模型是整体网络结构中的一层，但是词向量模型不会更新参数。

序列网络模型

双向RNN

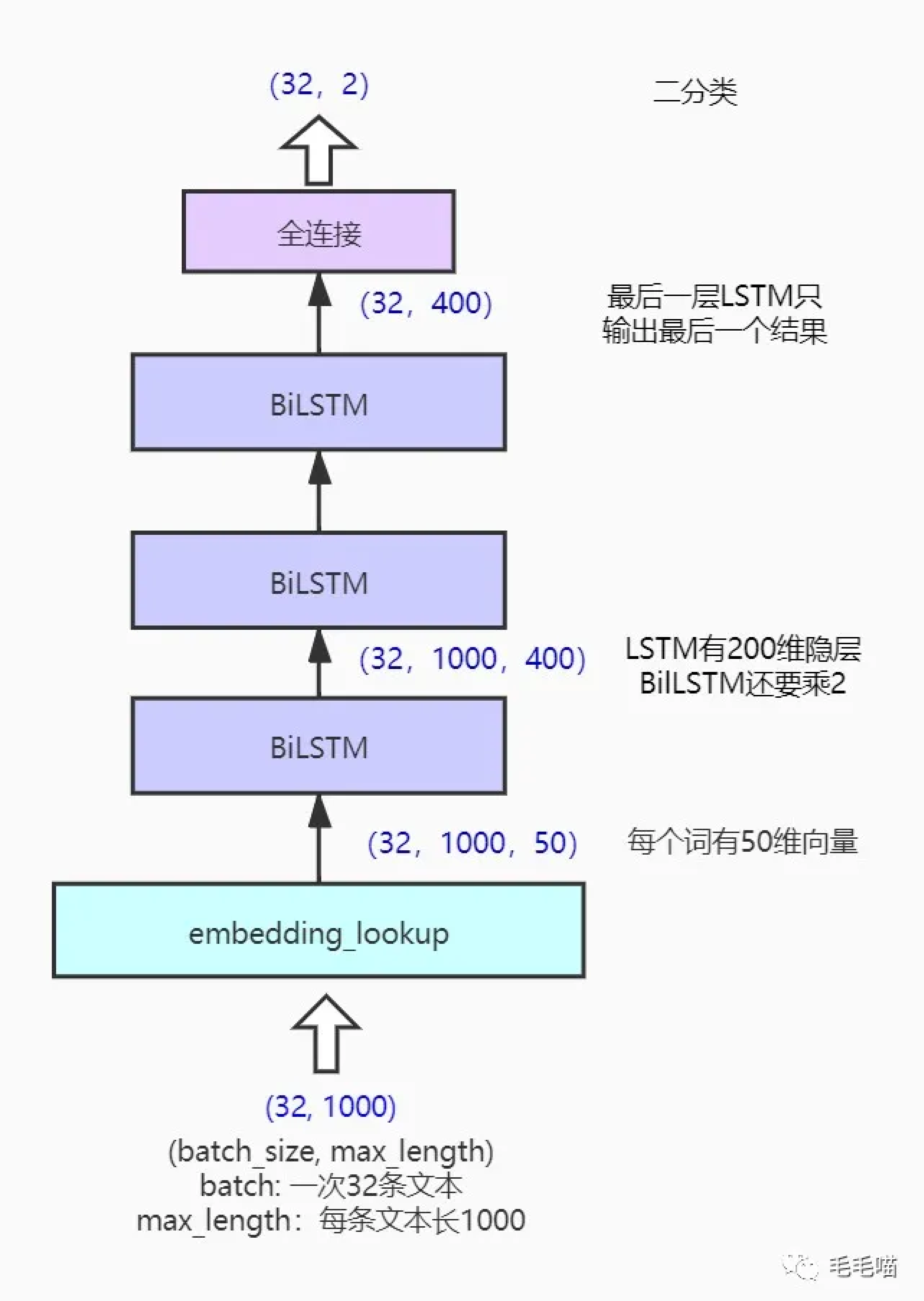
RNN从前到后走一遍，得到隐层特征h，双向RNN在实现的时候，再从后到前走一遍，又得到另一组h，两组h拼接一起就是双向RNN的隐层生成



实现很简单，只需要封装一个 `Bidirectional` ，比如 `self.rnn1 = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(params['rnn_units'], return_sequences=True))` 。这样隐层输出会有 `2*rnn_units` 个隐层特征。

网络构架

增加了三层双向RNN网络，每舱网络有 `200*2` 维隐层特征，并添加 `dropout` ，最后全连接层。



改进：

RNN训练的时候速度很慢，文本长度1000，要1000长度的时间序列以词输入RNN。为了提高速度，利用 `reshape` ，保证一次输入10长度的时间序列输入RNN，再利用 `reduce_max` 将

RNN隐层10长度的时间序列压缩维1个长度的序列，再进入下一层RNN，先 `reshape` 再 `reduce_max` ，这样可以提高运算速度。

```
x = tf.reshape(x, (batch_sz*10*10, 10, 50)) #改变尺寸，只有10长度的时间序列输入RNN
x = self.drop1(x, training=training)
x = self.rnn1(x)
x = tf.reduce_max(x, 1)
```

训练

损失：softmax交叉熵

学习率：学习率随着步长指数下降。 `decay_lr = tf.optimizers.schedules.ExponentialDecay(params['lr'], 1000, 0.95)`

评估：验证集准确率三次不下降就停止训练。

```
INFO:tensorflow:Step 10450 | Loss: 0.2917 | Spent: 77.7 secs | LR: 0.000176
INFO:tensorflow:Step 10500 | Loss: 0.4189 | Spent: 77.6 secs | LR: 0.000175
INFO:tensorflow:Step 10550 | Loss: 0.3117 | Spent: 77.7 secs | LR: 0.000175
INFO:tensorflow:Step 10600 | Loss: 0.2229 | Spent: 78.0 secs | LR: 0.000174
INFO:tensorflow:Step 10650 | Loss: 0.3178 | Spent: 77.8 secs | LR: 0.000174
INFO:tensorflow:Step 10700 | Loss: 0.1415 | Spent: 77.8 secs | LR: 0.000173
INFO:tensorflow:Step 10750 | Loss: 0.2664 | Spent: 77.8 secs | LR: 0.000173
INFO:tensorflow:Step 10800 | Loss: 0.2654 | Spent: 77.7 secs | LR: 0.000172
INFO:tensorflow:Step 10850 | Loss: 0.1829 | Spent: 77.7 secs | LR: 0.000172
INFO:tensorflow:Step 10900 | Loss: 0.2204 | Spent: 77.7 secs | LR: 0.000172
Reading ./data/test.txt
INFO:tensorflow:Evaluation: Testing Accuracy: 0.863
INFO:tensorflow:Best Accuracy: 0.879
INFO:tensorflow:Testing Accuracy not improved over 3 epochs, Early Stop
```


相关推荐

【程序喵笔记】递归神经网络实现预测
【程序喵笔记】ResNet网络实现分类
【程序喵笔记】从神经网络到卷积神经网络