


基于Seq2Seq+Attention模型的Textsum文本自动摘要

1,December 27th

 Github下载完整代码

简介

这篇文章中我们将基于Tensorflow的Seq2Seq+Attention模型，介绍如何训练一个中文的自动生成新闻标题的模型。自动总结(Automatic Summarization)类型的模型一直是研究热点。直接抽出重要的句子的抽取式方法较为简单，有如textrank之类的算法，而生成式(重新生成新句子)较为复杂，效果也不尽如人意。目前比较流行的Seq2Seq模型，由 Sutskever等人提出，基于一个Encoder-Decoder的结构将source句子先Encode成一个固定维度d的向量，然后通过Decoder部分一个字符一个字符生成Target句子。添加入了Attention注意力分配机制后，使得Decoder在生成新的Target Sequence时，能得到之前Encoder编码阶段每个字符的隐藏层的信息向量Hidden State，使得生成新序列的准确度提高。

数据准备和预处理

我们选择公开的“搜狐新闻数据(SogouCS)”的语料，包含2012年6月—7月期间的新闻数据，超过1M的语料数据，包含新闻标题和正文的信息。数据集可以从搜狗lab下载。http://www.sogou.com/labs/resource/cs.php

数据的预处理阶段极为重要，因为在Encoder编码阶段处理那些信息，直接影响到整个模型的效果。我们主要对下列信息进行替换和处理：

- 特殊字符：去除特殊字符，如：“「，」，¥,...”；
- 括号内的内容：如表情符，【嘻嘻】，【哈哈】
- 日期：替换日期标签为TAG_DATE，如：***年*月*日，****年*月，等等
- 超链接URL：替换为标签TAG_URL；
- 删除全角的英文：替换为标签TAG_NAME_EN；
- 替换数字：TAG_NUMBER；

在对文本进行了预处理后，准备训练语料：我们的Source序列，是新闻的正文，待预测的Target序列是新闻的标题。我们截取正文的分词个数到MAX_LENGTH_ENC=120个词，是为了训练的效果正文部分不宜过长。标题部分截取到MIN_LENGTH_ENC = 30，即生成标题不超过30个词。

- 在data_util.py类中，生成训练数据时做了下列事情：
- create_vocabulary()方法创建词典；
- data_to_token_ids()方法把训练数据(content-train.txt)转化为对应的词ID的表示；

两个文件的格式：

```
# 数据1 正文 content-train.txt
世 间 本 没 有 歧 视 TAG_NAME_EN 歧 视 源 自 于 人 的 内 心 活 动 TAG_NAME_EN " 以 爱 之 名 " TAG_DATE 中 国 艾
济 慈 之 家 小 朋 友 感 受 爱 心 椅 子 TAG_DATE TAG_NAME_EN 思 源 焦 点 公 益 基 金 向 盲 童 孤 儿 院 " 济 慈 之 家
...

# 数据2 标题 title-train.txt
艾 滋 病 反 歧 视 创 意 大 赛
思 源 焦 点 公 益 基 金 联 手 曲 美 家 具 共 献 爱 心
...
```

训练模型

```
#代码1
python headline.py
```

预测

运行predict.py, 交互地输入分好词的文本, 得到textsum的结果

```
#代码2-1
python predict.py

# 输入和输出
#> 中 央 气 象 台 TAG_DATE TAG_NUMBER 时 继 续 发 布 暴 雨 蓝 色 预 警 TAG_NAME_EN 预 计 TAG_DATE TAG_NUMBE
#current bucket id0
#中 央 气 象 台 发 布 暴 雨 蓝 色 预 警
#>
```

我们尝试输入下列分好词的新闻正文，一些挑选过的自动生成的中文标题如下：

ID	新闻正文	新闻标题	textsum自动生
----	------	------	------------

推荐文章

API 调 用

www.deepnlp.org

Tensorflow 下 构 建 L

型 进 行 序 列 化 标 注

Tensorflow 并 行

(multicore)，多 线 程 (thread)

基 于 Seq2Seq+Atte

型 的 Textsum 文 本 自

Tensorflow C++ AP

训 练 模 型 和 生 产 环 境

			成标题
469	中央 气象台 TAG_DATE TAG_NUMBER 时 继续 发布 暴雨 蓝色 预警 TAG_NAME_EN 预计 TAG_DATE TAG_NUMBER 时至 TAG_DATE TAG_NUMBER 时 TAG_NAME_EN 内蒙古 东北部、 山西 中 北部、 河北 中部 和 东北部、 京津 地区、 辽宁 西南部、 吉林 中部、 黑龙江 中部 偏南 等 地 的 部分 地区 有 大雨 或 暴雨。	中央 气象台 继续 发布 暴雨 预警 北 京 等 地 有 大雨	中央 气象台 发 布 暴雨 蓝色 预 警
552	美国 科罗拉多州 山林 大火 持续 肆虐 TAG_NAME_EN 当地 时间 TAG_DATE 横扫 州 内 第二 大 城市 科罗拉多斯 普林斯 一 处 居民区 TAG_NAME_EN 迫使 超过 TAG_NUMBER TAG_NAME_EN TAG_NUMBER 万 人 紧急 撤离。 美国 正 值 山火 多发 季 TAG_NAME_EN 现有 TAG_NUMBER 场 山火 处于 活跃 状态。	山火 横扫 美 西部 TAG_NUMBER 州 奥 巴马 将 赴 灾 区 视察 联邦 调 查 局 介 入 查 原因	美国 科罗拉多 州 山火 致 TAG_NUMBER 人 死亡
917	埃及 选举 委员会 昨天 宣布 TAG_NAME_EN 穆斯林 兄弟会 下 属 自由 与 正义党 主席 穆尔西 获得 TAG_NUMBER TAG_NAME_EN TAG_NUMBER TAG_NAME_EN 的 选票 TAG_NAME_EN 以 微弱 优 势 击败 前 总理 沙 菲克 赢得 选举 TAG_NAME_EN 成为 新任 埃及 总 统。 媒体 称 其 理念 获 下 层 民 众 支持。	埃及 大 选 昨晚 结 束 新 总统 穆尔西 被 认 为 具 有 改革 魄 力	埃及 总 统 选举 结 果
920	上 周 TAG_NAME_EN 广东 华 兴 银行 在 央 行 宣布 降 息 和 调整 存 贷 款 波 幅 的 第二 天 TAG_NAME_EN 立即 宣布 首 套 房 贷 利率 最低 执 行 七 折 优 惠。 一 石 激 起 千 层 浪 TAG_NAME_EN 随 之 而 起 的 “ 房 贷 七 折 利率 重 出 江 湖 ” 和 “ 房 地 产 调 控 松 绑 ” 的 谣 言 四 起。	房 贷 “ 七 折 利率 ” 真 相 调 查 TAG_NAME_EN 符 合 条 件 的 几 乎 为 零	银 监 会 否 认 房 贷 房 贷 利率

预测并计算ROUGE评估

```
运行predict.py, 同时调用eval.py 中的 evaluate(X, Y, method = "rouge_n", n = 2) 方法计算ROUGE分

#代码2-2 linux shell
folder_path=`pwd`
input_dir=${folder_path}/news/test/content-test.txt
reference_dir=${folder_path}/news/test/title-test.txt
summary_dir=${folder_path}/news/test/summary.txt

python predict.py $input_dir $reference_dir $summary_dir

# 输出:
# 中央 气象台 发布 暴雨 蓝色 预警
# Evaluated Rouge-2 score is 0.1818
# ...
```

下面我们将具体介绍tensorflow的seq2seq模型如何实现，首先先简单回顾模型的结构。

Seq2Seq+Attention模型回顾

Seq2Seq模型有效地建模了基于输入序列，预测未知输出序列的问题。模型有两部分构成，一个编码阶段的"Encoder"和一个解码阶段的"Decoder"。如下图的简单结构所示，Encoder的RNN每次输入一个字符代表的embedding向量，如依次输入A,B,C,及终止标志，将输入序列编码成一个固定长度的向量；之后解码阶段的RNN会一个一个字符地解码，如预测为X，之后在训练阶段会强制将前一步解码的输出作为下一步解码的输入，如X会作为下一步预测Y时的输入。

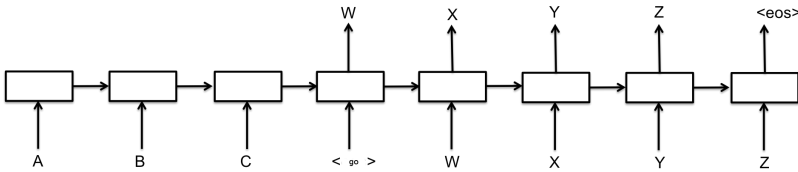


图1 Seq2Seq model

定义输入序列 $x = [x_1, x_2, \dots, x_{T_x}]$ ，由 T_x 个固定长度为 d 的向量构成；输出序列为 $y = [y_1, y_2, \dots, y_{T_y}]$ ，由 T_y 个固定长度为 d 的向量构成；定义输入序Encoder阶段的RNN隐藏层为 h_j , Decoder阶段的RNN隐藏层为 S_i

Attention注意力分配机制

LSTM模型虽然具有记忆性，但是当Encoder阶段输入序列过长时，解码阶段的LSTM也无法很好地针对最早的输入序列解码。基于此，Attention注意力分配的机制被提出，就是为了解决这个问题。在Decoder阶段每一步解码，都能够有一个输入，对输入序列所有隐藏层的信息 h_1, h_2, \dots, h_{T_x} 进行加权求和。打个比方就是每次在预测下一个词时都会把所有输入序列的隐藏层信息都看一遍，决定预测当前词时和输入序列的那些词最相关。

Attention机制代表了在解码Decoder阶段，每次都会输入一个Context上下文的向量 C_i , 隐藏层的新状态 S_i 根据上一步的状态 S_{i-1} , Y_i , C_i 三者的一个非线性函数得出。

$$s_i = f(s_{i-1}, y_i, c_i)$$

Context向量在解码的每一步都会重新计算，根据一个MLP模型计算出输出序列 i 对每个输入序列 j 的隐含层的对应权重 a_{ij} , 并对所有隐含层加权平均。文章中说Alignment Model就是代表这种把输入序列位置 j 和输出序列位置 i 建立关系的模型。

$$c_i = \sum_{j=1}^{T_x} a_{ij} h_j$$

a_{ij} 即可以理解为Decoder解码输出序列的第 i 步，对输入序列第 j 步分配的注意力权重。

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

$$e_{ij} = v_a^T \tanh(W_a s_{i-1} + U_a h_j)$$

e_{ij} 为一个简单的MLP模型激活的输出； a_{ij} 的计算是对 e_{ij} 做softmax归一化后的结果。

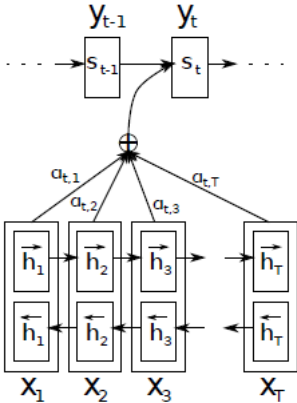


Figure 1: The graphical illustration of the proposed model trying to generate the t -th target word y_t given a source sentence (x_1, x_2, \dots, x_T) .

图2 Translate Seq2Seq alignment model

Soft Attention和Hard Attention区别

Soft Attention通常是指以上我们描述的这种全连接(如MLP计算Attention 权重)，对每一层都可以计算梯度和后向传播的模型；不同于Soft attention那样每一步都对输入序列的所有隐藏层 $h_j(j=1 \dots T_x)$ 计算权重再加权平均的方法，Hard Attention是一种随机过程，每次以一定概率抽样，以一定概率选择某一个隐藏层 h_j^* ，在估计梯度时也采用蒙特卡罗抽样Monte Carlo sampling的方法。

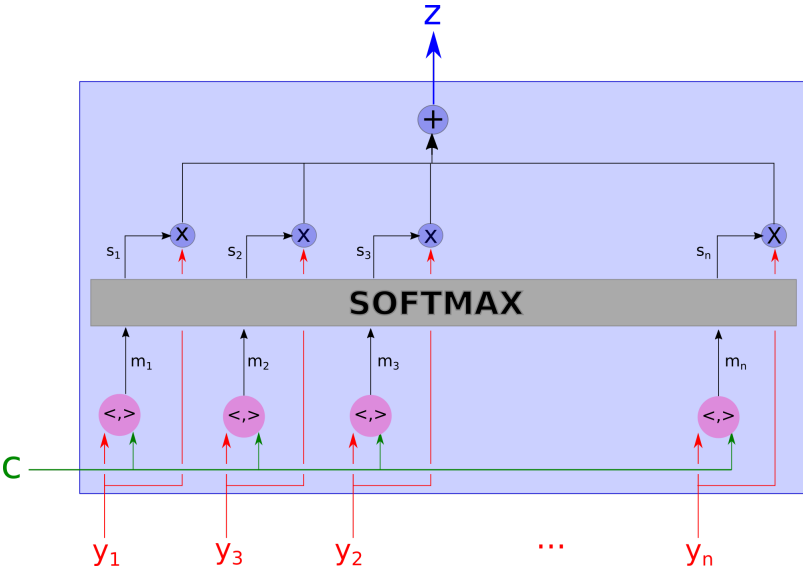


图3 Soft Attention 模型

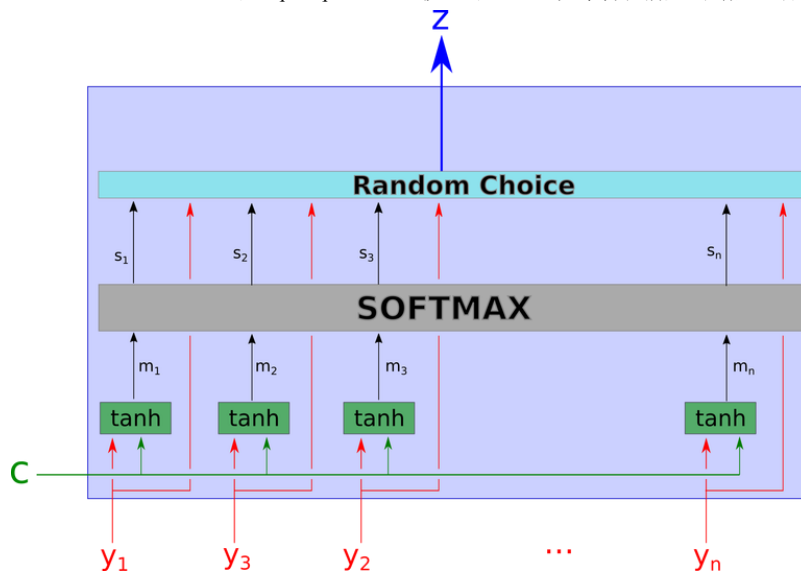


图4 Hard Attention 模型

模型实现

我们对Tensorflow基本教程里的translate英语法语翻译例子里的seq2seq_model.py类稍加修改，就能够符合我们textsum例子使用，另外我们还会分析针对英文的textsum教程中构建双向Bi-LSTM的Encoder-Decoder的例子。

1.Seq2Seq模型文件: seq2Seq_model.py

单向LSTM的Encoder-Decoder结构

教程中的例子很长，但是将实例代码分解来看不是那么复杂，下面将分三段来介绍官方tutorial里的如何构建seq2seq模型。

定义基本单元: 多层LSTM cell

```
#代码3-1
# Create the internal multi-layer cell for our RNN.
single_cell = tf.nn.rnn_cell.GRUCell(size) # default use GRU
if use_lstm:
    single_cell = tf.nn.rnn_cell.BasicLSTMCell(size, state_is_tuple=True)
cell = single_cell
if num_layers > 1:
    cell = tf.nn.rnn_cell.MultiRNNCell([single_cell] * num_layers, state_is_tuple=True)
```

定义前向过程的 seq2seq_f 函数，利用tf.nn.seq2seq.embedding_attention_seq2seq 返回output

```
# 代码3-2
# The seq2seq function: we use embedding for the input and attention.
def seq2seq_f(encoder_inputs, decoder_inputs, do_decode):
    return tf.nn.seq2seq.embedding_attention_seq2seq(
        encoder_inputs,
        decoder_inputs,
        cell,
        num_encoder_symbols=source_vocab_size,
        num_decoder_symbols=target_vocab_size,
        embedding_size=size,
        output_projection=output_projection,
        feed_previous=do_decode,
        dtype=tf.float32)
```

桶Bucket的机制: 应用Bucket机制，核心的思想是把输入序列的句子按照长度的相似程度分到不同的固定长度的Bucket里面，长度不够的都添加PAD字符。之所以有Bucket的原因是工程效率: "RNN在数学上是可以处理任意长度的数据的。我们在TensorFlow 中使用 bucket 的原因主要是为了工程实现的效率" (摘自知乎 JQY 的回答 <https://www.zhihu.com/question/42057513>)

```
# 代码3-3
# Training outputs and losses.
if forward_only:
    self.outputs, self.losses = tf.nn.seq2seq.model_with_buckets(
        self.encoder_inputs, self.decoder_inputs, targets,
        self.target_weights, buckets, lambda x, y: seq2seq_f(x, y, True),
        softmax_loss_function=softmax_loss_function)
    # If we use output projection, we need to project outputs for decoding.
    if output_projection is not None:
        for b in xrange(len(buckets)):
            self.outputs[b] = [
                tf.matmul(output, output_projection[0]) + output_projection[1]
                for output in self.outputs[b]
            ]
else:
    self.outputs, self.losses = tf.nn.seq2seq.model_with_buckets(
```

```
self.encoder_inputs, self.decoder_inputs, targets,
self.target_weights, buckets, lambda x, y: seq2seq_f(x, y, False),
softmax_loss_function=softmax_loss_function)
```

tf.nn.seq2seq.model_with_buckets()结果返回output序列: 'buckets = [(120, 30), (200, 35), (300, 40), (400, 40), (500, 40)]'

https://github.com/tensorflow/tensorflow/blob/64edd34ce69b4a8033af5d217cb8894105297d8a/tensorflow/contrib/legacy_seq2seq/python/ops/seq2seq.py

双向LSTM的Encoder-Decoder结构

Encoder阶段自己定义了Bi-LSTM结构

```
# 代码4-1
# URL: https://github.com/tensorflow/models/tree/master/textsum
# Encoder: Multi-Layer LSTM, Output: encoder_outputs
for layer_i in xrange(hps.enc_layers):
    with tf.variable_scope('encoder%d'%layer_i), tf.device(
        self._next_device()):
        cell_fw = tf.nn.rnn_cell.LSTMCell(
            hps.num_hidden,
            initializer=tf.random_uniform_initializer(-0.1, 0.1, seed=123),
            state_is_tuple=False)
        cell_bw = tf.nn.rnn_cell.LSTMCell(
            hps.num_hidden,
            initializer=tf.random_uniform_initializer(-0.1, 0.1, seed=113),
            state_is_tuple=False)
        (emb_encoder_inputs, fw_state, _) = tf.nn.bidirectional_rnn(
            cell_fw, cell_bw, emb_encoder_inputs, dtype=tf.float32,
            sequence_length=article_lens)
    encoder_outputs = emb_encoder_inputs

with tf.variable_scope('output_projection'):
    w = tf.get_variable(
        'w', [hps.num_hidden, vsz], dtype=tf.float32,
        initializer=tf.truncated_normal_initializer(stddev=1e-4))
    w_t = tf.transpose(w)
    v = tf.get_variable(
        'v', [vsz], dtype=tf.float32,
        initializer=tf.truncated_normal_initializer(stddev=1e-4))
```

Decoder阶段, 利用内置的tf.nn.seq2seq.attention_decoder()函数返回output

```
# 代码4-2
with tf.variable_scope('decoder'), tf.device(self._next_device()):
    # When decoding, use model output from the previous step
    # for the next step.
    loop_function = None
    if hps.mode == 'decode':
        loop_function = _extract_argmax_and_embed(
            embedding, (w, v), update_embedding=False)

    cell = tf.nn.rnn_cell.LSTMCell(
        hps.num_hidden,
        initializer=tf.random_uniform_initializer(-0.1, 0.1, seed=113),
        state_is_tuple=False)

    encoder_outputs = [tf.reshape(x, [hps.batch_size, 1, 2*hps.num_hidden])
                        for x in encoder_outputs]
    self._enc_top_states = tf.concat(1, encoder_outputs)
    self._dec_in_state = fw_state
    # During decoding, follow up _dec_in_state are fed from beam_search.
    # dec_out_state are stored by beam_search for next step feeding.
    initial_state_attention = (hps.mode == 'decode')
    decoder_outputs, self._dec_out_state = tf.nn.seq2seq.attention_decoder(
        emb_decoder_inputs, self._dec_in_state, self._enc_top_states,
        cell, num_heads=1, loop_function=loop_function,
        initial_state_attention=initial_state_attention)
```

Attention注意力矩阵的可视化

Attention 注意力分配机制的权重矩阵 A_{ij} 可以反映在Decoder阶段第 i 个输出字符对Encoder阶段的第 j 个字符的注意力分配的权重 a_{ij} 。我们可以通过绘制Heatmap来可视化seq2seq模型中Decoder的Y对Encoder的X每个字符的权重。

获取attention_mask的值

attention_mask 即为我们感兴趣的注意力权重分配的tensor, 我们首先来看tensorflow的源码seq2seq.py这个ops的实现, 容易发现, 计算attention_mask 变量a的代码出现在 attention_decoder()函数内的attention()函数体内, $a = \text{nn_ops.softmax}(s)$ 这句。我们把该变量添加到return语句中的返回值, 同时也修改所有调用了attention_decoder()的上层的函数, 为了最终能够在主函数中将attn_mask这个变量抽取出来。具体需要修改的脚本参考textsum项目下的seq2seq_attn.py这个文件。之后我们在主函数中利用 $\text{attn_out} = \text{session.run}(\text{self.attn_masks}[\text{bucket_id}], \text{input_feed})$, 对变量进行session.run() 就可以获得当前这个样本的attention矩阵的值。

```
# 代码5-1
# URL: https://github.com/rockydingo/deepnlp/blob/master/deepnlp/textsum/seq2seq_attn.py

def attention_decoder():
```

```

## some code

def attention(query):
    """Put attention masks on hidden using hidden_features and query."""
    ds = []
    if nest.is_sequence(query):
        query_list = nest.flatten(query)
        for q in query_list:
            ndims = q.get_shape().ndims
            if ndims:
                assert ndims == 2
            query = array_ops.concat_v2(query_list, 1)
    for a in xrange(num_heads):
        with variable_scope.variable_scope("Attention_%d" % a):
            y = linear(query, attention_vec_size, True)
            y = array_ops.reshape(y, [-1, 1, 1, attention_vec_size])
            # Attention mask is a softmax of v^T * tanh(...).
            s = math_ops.reduce_sum(v[a] * math_ops.tanh(hidden_features[a] + y),
                                   [2, 3])

            # Tensor a 即为我们需要抽取的attention_mask
            a = nn_ops.softmax(s)

            d = math_ops.reduce_sum(
                array_ops.reshape(a, [-1, attn_length, 1, 1]) * hidden, [1, 2])
            ds.append(array_ops.reshape(d, [-1, attn_size]))
    return ds, a

```

利用matplotlib可视化

我们利用matplotlib包中绘制heatmap的函数，可以简单地将上一步抽取出的attn_matrix可视化。在eval.py模块中我们整合了一个eval.plot_attention(data, X_label=None, Y_label=None) 函数来简单绘制attention权重矩阵。运行 predict_attn.py 脚本，输入分好词的待分析新闻文本，然后自动生成的jpg图片就保存在./img目录下。

```
# 代码5-2
# 输入文本，查看Attention的heatmap:
# > 中央 气象台 TAG_DATE TAG_NUMBER 时 继续 发布 暴雨 蓝色 预警 TAG_NAME_EN 预计 TAG_DATE TAG_NUMB

python predict attn.py
```

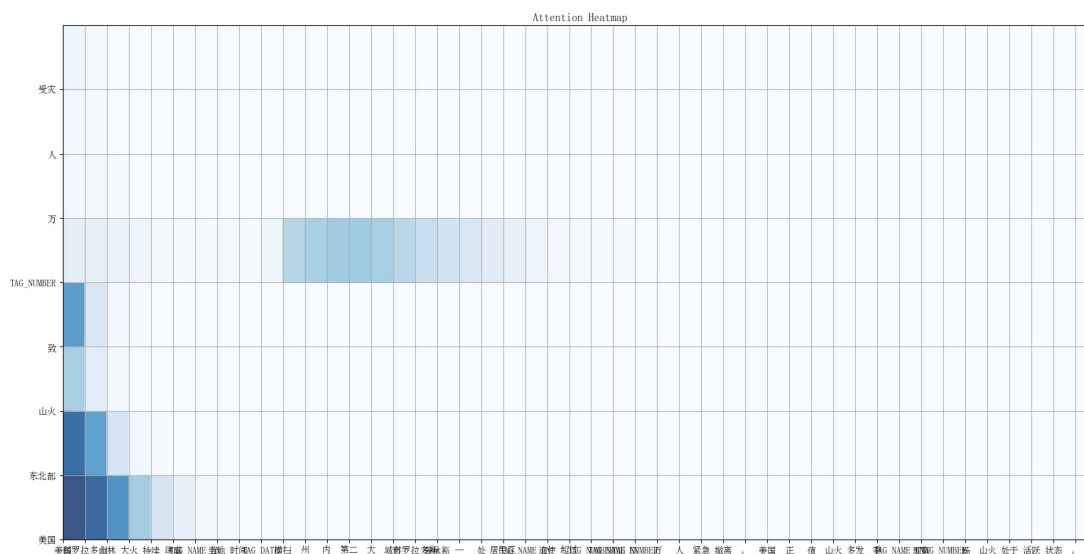


图5 Attention注意力权重分配的Heatmap

可视化部分代码参考项目中的 `predict_attn.py` `seq2seq_attn.py` `seq2seq_model_attn.py` 这三个文件。

预测阶段Decode策略: 贪心算法和Beam Search

贪心算法 Beam Search

拓展阅读

Github源码textsum

Tensorflow seq2seq.py

Tensorflow Bi-LSTM textsum examples

ATTENTION MECHANISM

打赏

[关于我们](#) [协议与隐私](#) [友情链接](#) [京ICP备16066061](#)
© 2017 深语人工智能 DeepNLP