端对端记忆网络学习笔记

Hang Hang Li

2018 年 6 月 1 日

论文地址：https://arxiv.org/abs/1503.08895

数据集实现代码：https://github.com/lihanghang/ML/tree/master/MemNN

1 网络结构概述

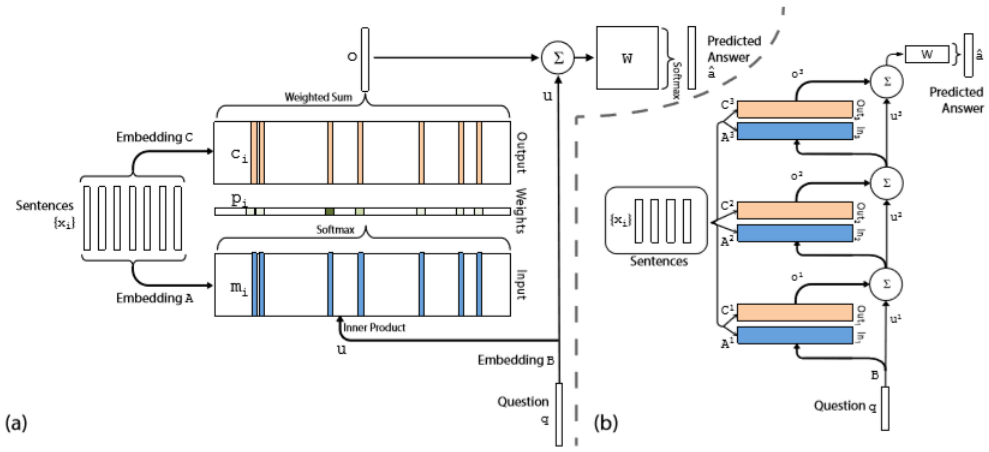

图 1: 端对端记忆网络单（a）、多（b）层架构图

1 其中三个 Embedding 矩阵 A、B、C，目的是将: text and question of input encode to word vector; W 为最终输入的权重矩阵

2 输入 Sentences $[x_i]$，会经过 EmbeddingA、C 进行编码，分别得到 input and output 记忆模块，其中 input 和经过 Embeding B encode 后 question 的向量作乘，得到与 q 的相关性；output 则与相关性进行加权求和得到 output vector.Then that sum of q to output layer.

2 单层记忆网络架构模块分析

2.1 Input Model

1 本论文的输入模块可以与 Memory Network 论文中的 I and G 组件对应理解，其目的是将输入的文本进行向量转化并保存到记忆数组中（如单层架构图中蓝色竖条所示）。本文试图使用 BoW and Position encode. 两种方式。其中 BoW 是将一个句子中所有单词的词向量求和表示成一个向量的形式。但正如论文提到这个 encode way 存在一个明显的缺点：it cannot capture the order of the words in the sentence（不能够描述句子中次序关系），即丢失了语义信息，而语义信息对于某些任务是非常关键的；转而使用 Position encoding（位置编码），根据每个位置单词权重的不同，进行加权求和得到句子的向量表示，编码公式如下:

$$m_i = \sum_j l_j \cdot Ax_{ij}$$

l_j 是 l_{kj} 的一个列向量，其中 l_{kj} 计算公式如下：

$$l_{kj} = (1 - j/J) - (k/d)(1 - 2j/J)$$

其中 J 每个句子单词的数量，d 是词向量. 为了解决时序信息的处理，例如：Sam is in the bedroom after he is in the kitchen. 本文提出时序编码（Temporal Encoding）将记忆向量修改为如下方式计算：

$$m_i = \sum_j Ax_{ij} + T_A(i)$$

$T_A(i)$ 时序编码信息矩阵，ith 表示其矩阵行向量

2.2 Output Model

上面模块实现了将句子通过输入模型进行数据编码后，以向量形式存储在 memory slot 中. 输出模块分为 input and output 两个模块，一个用于和 Question 计算得到相关性，一个用于计算得出信息的输出。The query **q** is also embedded to obtain an internal state **u** ,Q 通过 Embedding C 转化为向量 u，最后通过与记忆单元 m_i 作内积，再进行 **Softmax** 作归一化处理后得到 p_i, 计算公式如下：

$$p_i = Softmax(u^T m_i)$$

p_i 就是 q 与 m_i 的相关性.
最后进行 output 中的记忆 c_i 通过 p_i 进行加权求和得到模型的输出向量 $\vec{o}$, 计算公式如下：

$$\mathrm{o} = \sum_i p_i c_i$$

2.3 Generating the final prediction Model

最后一步，产生根据以上信息得到答案。output vector O 和 input U 进行求和，然后和 W 权重矩阵相乘，通过 Softmax 函数产生各个单词为答案的概率，概率最大的单词就是问题的答案，并使用 standard cross-entropy loss（标准交叉熵损失函数）进行训练。计算公式如下所示：

$$\hat{a} = Sofmax(W(o + u))$$

3 多层记忆网络架构模块分析

显然，这里多层指的就是单层中输出模块即 m_i 和 c_i 有多组，成为一个 Stack, 论文中也引入一个词成为 Hop。如最上面架构图 (b) 部分所示，表示的是一个三层网络，从下至上，下一层为上一层提供输入及 the sum of o and u.

论文对于每一层 A 和 C 的处理方法提出两种方法：

1 Adjacent: 使相邻层 A and C 相等, 即 $A_{k+1} = C_k$。另外，$W = C_k$,$B = A_1$, 这样就使得参数减少了一半。

2 Layer-wise (RNN-like)：与 RNN 类似，采用共享参数的方法，即每一层所使用的参数就是相等的，大大减少了参数数量，导致模型效果很差。随即提出了一种改进方法：在每层之间增加 linear mapping H(线性映射矩阵) ，使用下面式子表示：

$$u^{k+1} = H_{u^k} + o^k$$

4 基于 TensorFlow 代码实现

4.1 数据集准备

数据集说明：

1 John travelled to the hallway.

2 Mary journeyed to the bathroom.

3 Where is John? hallway 1

4 Daniel went back to the bathroom.

5 John moved to the bedroom.

6 Where is Mary? bathroom 2

7 John went to the hallway.

8 Sandra journeyed to the kitchen.

9 Where is Sandra? kitchen 8

10 Sandra travelled to the hallway.

11 John went to the garden.

12 Where is Sandra? hallway 10

13 Sandra went back to the bathroom.

14 Sandra moved to the kitchen.

15 Where is Sandra? kitchen 14

格式：每两个句子 + 一个问题 + 一个答案 + 编号（即与答案相关行）

每 15 行为一组，即 5 个问题一组

上面数据部分从网络模块参数来看：Memory Size =10 ， Question=5

论文的一个特色是：**模型自身可学习到与问题最相关的输入**。这一点比 14 年发表的论文《Memory Network》https://arxiv.org/abs/1410.3916有了进步!

4.2 核心代码说明

1 数据预处理

```
def parse_stories(lines, only_supporting=False):
    '''Parse stories provided in the bAbI tasks format
    If only_supporting is true, only the sentences that support the answer are kept.
    supporting即为与答案的相关行
    本论文是弱监督的，因此我们将only_supoorting设置为False
    story为问题前的对话句子
    '''
    data = []
    story = []
    for line in lines:
        line = str.lower(line)
        nid, line = line.split(' ', 1)
        nid = int(nid)
        if nid == 1:
            story = []
        if '\t' in line: # question、supporting即为与答案的相关行
            q, a, supporting = line.split('\t')
            q = tokenize(q)
            #a = tokenize(a)
```

```
            # answer is one vocab word even if it's actually multiple words
            a = [a]
            substory = None

            # remove question marks
            if q[-1] == "?":
                q = q[:-1]
    #是否需要将和答案相关的句子保存起来，本论文选择不保存，由网络自行训练查找
            if only_supporting:
                # Only select the related substory
                supporting = map(int, supporting.split())
                substory = [story[i - 1] for i in supporting]
            else:
                # Provide all the substories
                substory = [x for x in story if x]

            data.append((substory, q, a))
            story.append('')
        else: # regular sentence
            # remove periods
            sent = tokenize(line)
            if sent[-1] == ".":
                sent = sent[:-1]
            story.append(sent)
    return data
```

5 实验结果与分析

5.1 实验结果

（'Epoch', 1000)
('Total Cost:', 2.9474439509212971)
('Training Accuracy:', 0.99888888888888894)
('Validation Accuracy:', 0.64000000000000001)

5.2 结果分析

几轮调参下来，测试数据的准确率基本在 66.7% 左右浮动

参考文献

[1] Arthur Szlam Jason Weston Rob Fergus.End-To-End Memory Networks.Facebook AI Research , Nov,2015.