Compare-Aggregate

题目：A COMPARE-AGGREGATE MODEL FOR MATCHING TEXT SEQUENCES

摘要：

1. 提出来一种较为通用的compare-aggregate框架，该框架先进行词级别的匹配，然后使用CNN进行聚合。
2. 主要用两种不同的比较函数去匹配两个向量。发现一些基于元素运算的简单比较函数（simple comparison functions）比一些标准的网络或者神经张量网络能更好的工作。
3. 使用四种不同的数据集进行评价模型的性能。

Introduction

传统的深度学习模型：

> 首先对句子进行embedding,得到embedding vector,然后通过RNN或者CNN将其encoder。
>
> 然后，为了匹配两个序列，最直接的方法就是将每个句子的encoder看作向量来比较。
>
> **缺点：**用单一的vector去encode整个句子并不十分有效，因此又出现了**attention机制、记忆网络**等

“compare-aggregate”框架：

> 最早由Google在《A Decomposable Attention Model for Natural Language Inference》论文中提出来。
>
> 这类方法主要是先对小单元的vector进行比较，然后再将比较的结果进行聚合，做最后的判断。
>
> **eg.**
>
> **match-LSTM：**在进行文本蕴含的时候，首先将假设中的每个单词与前提的注意加权进行比较，然后将比较的结果通过LSTM进行聚合
>
> **交互模型：**从两个句子里找出一对单词，然后进行单元比较。最后使用相似层和多层CNN将这些单词交互的结果组合在一起
>
> **可分解attention模型：**每个句子中的单词都和其他句子进行attention加权计算，产生一系列的比较向量，然后进行聚合加MLP进行最后的分类

以上的研究都展示了“**compare-aggregate**”框架的有效性，但是存在两点限制：提出来的每个模型都只会用于一种或两种任务； 这些研究并没有重视比较两个小文本单元的比较功能。即本质上是要去关注两个序列在语义上的相似度，进行序列的小元素的比较只为了选择出更加合适的比较函数。

本文贡献：

- 提出了统一的“compare-aggregate”框架，能有效应用于句子匹配任务上
- 用四种不同的数据集，分别是代表不同的任务，在其上进行了六种不同的比较函数的测试，最终发现基于元素的减法和乘法，结果最好

模型结构

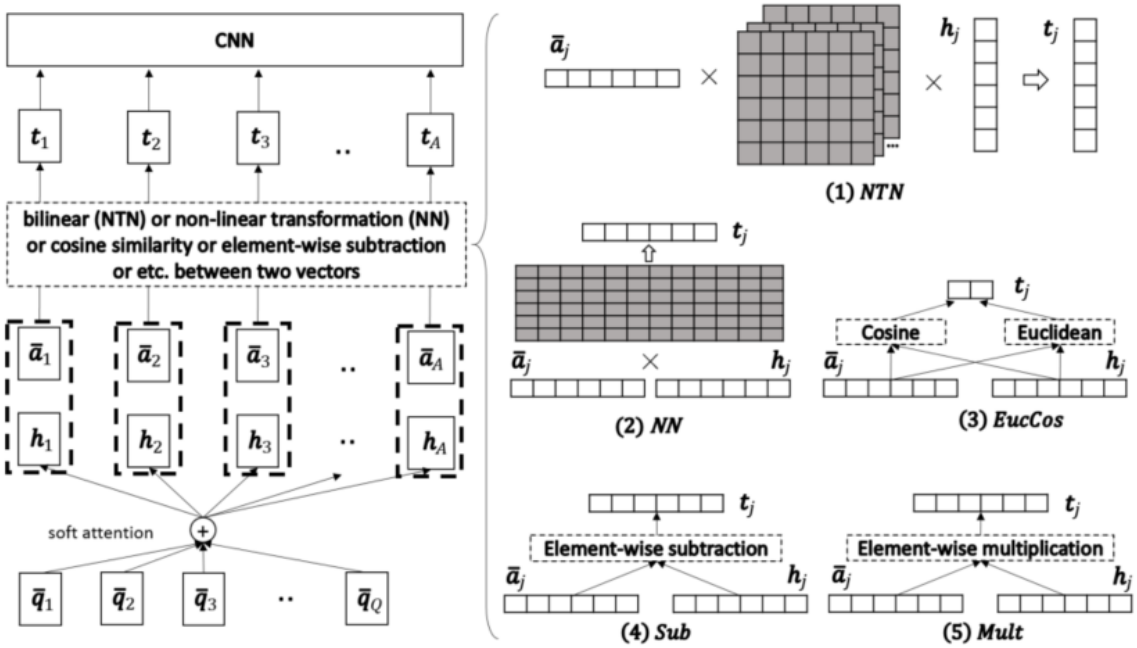

Figure 1: The left hand side is an overview of the model. The right hand side shows the details about the different comparison functions. The rectangles in dark represent parameters to be learned. × represents matrix multiplication.

主要分为四层：

Step1.Preprocessing:

> 首先对Q和A进行预处理，获得两个新的矩阵$\bar{Q}$:和 $\bar{A}$，其目的是为每个序列中的每个单词获得一个新的embedding vector，使每个词都能获得句子的上下文信息。
> 比如$\bar{Q}$里面的$\bar{q_i}$是$\bar{Q}$的第i^{th}列向量，是通过对 $\bar{Q}$中的第i^{th}个字及其在Q中的上下文进行编码。具体公式如下：

$$\begin{aligned}\overline{\mathbf{Q}} &= \sigma(\mathbf{W}^{\mathrm{i}}\mathbf{Q}+\mathbf{b}^{\mathrm{i}}\otimes\mathbf{e}_Q)\odot\tanh(\mathbf{W}^{\mathrm{u}}\mathbf{Q}+\mathbf{b}^{\mathrm{u}}\otimes\mathbf{e}_Q),\\ \overline{\mathbf{A}} &= \sigma(\mathbf{W}^{\mathrm{i}}\mathbf{A}+\mathbf{b}^{\mathrm{i}}\otimes\mathbf{e}_A)\odot\tanh(\mathbf{W}^{\mathrm{u}}\mathbf{A}+\mathbf{b}^{\mathrm{u}}\otimes\mathbf{e}_A),\end{aligned}$$

> 其中：⊗表示把b^i重复，类似于广播
>
> 原文表示为：这是将Q和A经过一个modified的LSTM/GRU，通过这种方式，使得$\bar{Q}$和$\bar{A}$记得句子里的有用信息。

```
function compAggWikiqa:new_proj_module()
    local input = nn.Identity()()
    local i = nn.Sigmoid()(nn.Linear(self.emb_dim, self.mem_dim)(input))
    local u = nn.Tanh()(nn.Linear(self.emb_dim, self.mem_dim)(input))
    local output = nn.CMulTable(){i, u}
    local module = nn.gModule({input}, {output})
    if self.proj_module_master then
        share_params(module, self.proj_module_master)
    end
    return module
end
```

Step2.Attention:

就是传统的attention机制

$$\begin{aligned}\mathbf{G} &= \text{softmax}\left((\mathbf{W}^{\mathrm{g}}\overline{\mathbf{Q}} + \mathbf{b}^{\mathrm{g}} \otimes \mathbf{e}_Q)^{\mathrm{T}}\overline{\mathbf{A}}\right), \\ \mathbf{H} &= \overline{\mathbf{Q}}\mathbf{G},\end{aligned}$$

用问题对答案做attention，可以得到权重矩阵G，即$\bar{A}$和$\bar{Q}$的注意力权重

然后将$\bar{Q}$作为V，即$\bar{Q}$的每个列向量$\bar{q_j}$和G的列向量相乘，得到对应的表示h_j

```
function compAggWikiqa:new_att_module()
    local linput, rinput = nn.Identity()(), nn.Identity()()
    --padding
    local lPad = nn.Padding(1,1)(linput)
    --local M_l = nn.Linear(self.mem_dim, self.mem_dim)(lPad)
    local M_r = nn.MM(false, true){lPad, rinput}
    local alpha = nn.SoftMax()( nn.Transpose({1,2})(M_r) )
    local Yl =  nn.MM(){alpha, lPad}
    local att_module = nn.gModule({linput, rinput}, {Yl})
    if self.att_module_master then
        share_params(att_module, self.att_module_master)
    end
    return att_module
end
```

Step3.Comparison:

将答案信息$\bar{a_j}$与h_j进行比较得到新的向量表示t_j，作者比较了多种comparison方式。其实就是找出来一个最好的函数，即论文里所谓的comparison layer。

- Neuralnet:将两个向量进行拼接，过层神经网络。

$$\text{NEURALNET (NN):} \qquad \mathbf{t}_j = f(\overline{\mathbf{a}}_j, \mathbf{h}_j) = \text{ReLU}(\mathbf{W}\begin{bmatrix}\overline{\mathbf{a}}_j \\ \mathbf{h}_j\end{bmatrix} + \mathbf{b}),$$

- Neuraltensornet: 用张量网络连接两个向量，过relu.

$$\text{NEURALTENSORNET (NTN):} \qquad \mathbf{t}_j = f(\overline{\mathbf{a}}_j, \mathbf{h}_j) = \text{ReLU}(\overline{\mathbf{a}}_j^{\mathrm{T}}\mathbf{T}^{[1\ldots l]}\mathbf{h}_j + \mathbf{b}),$$

- EUCLID+COS:

  作者发现在很多论文里，使用的最好的就是欧氏距离和cosine相似度，因此又将两者的结果拼接起来：

$$\text{EUCLIDEAN+COSINE (EUCCOS):} \qquad \mathbf{t}_j = f(\overline{\mathbf{a}}_j, \mathbf{h}_j) = \begin{bmatrix}\|\overline{\mathbf{a}}_j - \mathbf{h}_j\|_2 \\ \cos(\overline{\mathbf{a}}_j, \mathbf{h}_j)\end{bmatrix}.$$

  【注】：但是作者觉得，这样结合还是会导致这两个原始的vector里面会丢失很多有用的信息，因此考虑到了利用这两个vector里面的元素直接进行计算，于是就提出了下面的两个比较函数：

- SUB&MULT:

$$\text{SUBTRACTION (SUB):} \quad \mathbf{t}_j = f(\bar{\mathbf{a}}_j, \mathbf{h}_j) = (\bar{\mathbf{a}}_j - \mathbf{h}_j) \odot (\bar{\mathbf{a}}_j - \mathbf{h}_j),$$
$$\text{MULTIPLICATION (MULT):} \quad \mathbf{t}_j = f(\bar{\mathbf{a}}_j, \mathbf{h}_j) = \bar{\mathbf{a}}_j \odot \mathbf{h}_j.$$

【注】：然后作者还发现sub的计算方法和欧氏距离很相近，mul又比较接近于cosine.因此，又对这两个比较方式的结果进行拼接，过一层神经网络：

- SUBMULT+NN:

$$\text{SUBMULT+NN:} \quad \mathbf{t}_j = f(\bar{\mathbf{a}}_j, \mathbf{h}_j) = \text{ReLU}(\mathbf{W} \begin{bmatrix} (\bar{\mathbf{a}}_j - \mathbf{h}_j) \odot (\bar{\mathbf{a}}_j - \mathbf{h}_j) \\ \bar{\mathbf{a}}_j \odot \mathbf{h}_j \end{bmatrix} + \mathbf{b}).$$

Step4:Aggregation

最后得到了一系列的t_j，用CNN聚合即可：

$$\mathbf{r} = \text{CNN}([\mathbf{t}_1, \ldots, \mathbf{t}_A]).$$

最后利用向量**r**，根据不同的任务对模型进行训练和预测。文本蕴含做三分类，答案选择任务，则是从多个候选答案中选择正确答案。

3. Experiments

word embedding: GloVe 在GloVe里面没有的直接初始化为0
hidden layer = 150
optimize： adamaxβ_1 =0.9 , β_2 = 0.999
batch_size: 30
lr = 0.002

下面就在这四种数据集上进行了相关实验：

	MovieQA			InsuranceQA			WikiQA			SNLI		
	train	dev	test	train	dev	test	train	dev	test	train	dev	test
#Q	9848	1958	3138	13K	1K	1.8K*2	873	126	243	549K	9842	9824
#C	5	5	5	50	500	500	10	9	10	-	-	-
#w in P	873	866	914	-	-	-	-	-	-	-	-	-
#w in Q	10.6	10.6	10.8	7.2	7.2	7.2	6.5	6.5	6.4	14	15.2	15.2
#w in A	5.9	5.6	5.5	92.1	92.1	92.1	25.5	24.7	25.1	8.3	8.4	8.3

Table 2: The statistics of different data sets. Q:question/hypothesis, C:candidate answers for each question, A:answer/hypothesis, P:plot, w:word (average).

Models	MovieQA		InsuranceQA			WikiQA		SNLI	
	dev	test	dev	test1	test2	MAP	MRR	train	test
Cosine Word2Vec	46.4	45.63	-	-	-	-	-	-	-
Cosine TFIDF	47.6	**47.36**	-	-	-	-	-	-	-
SSCB TFIDF	**48.5**	-	-	-	-	-	-	-	-
IR model	-	-	52.7	55.1	50.8	-	-	-	-
CNN with GESD	-	-	65.4	65.3	61.0	-	-	-	-
Attentive LSTM	-	-	68.9	69.0	64.8	-	-	-	-
IARNN-Occam	-	-	69.1	68.9	**65.1**	**0.7341**	**0.7418**	-	-
IARNN-Gate	-	-	**70.0**	**70.1**	62.8	0.7258	0.7394	-	-
CNN-Cnt	-	-	-	-	-	0.6520	0.6652	-	-
ABCNN	-	-	-	-	-	0.6921	0.7108	-	-
CubeCNN	-	-	-	-	-	0.7090	0.7234	-	-
W-by-W Attention	-	-	-	-	-	-	-	85.3	83.5
match-LSTM	-	-	-	-	-	-	-	92.0	86.1
LSTMN	-	-	-	-	-	-	-	88.5	86.3
Decomp Attention	-	-	-	-	-	-	-	90.5	86.8
EBIM+TreeLSTM	-	-	-	-	-	-	-	93.0	**88.3**
NN	31.6	-	76.8	74.9	72.4	0.7102	0.7224	89.3	86.3
NTN	31.6	-	75.6	75.0	72.5	0.7349	0.7456	91.6	86.3
EUCCOS	71.9	-	70.6	70.2	67.9	0.6740	0.6882	87.1	84.0
SUB	64.9	-	70.0	71.3	68.2	0.7019	0.7151	89.8	**86.8**
MULT	66.4	-	76.0	75.2	**73.4**	**0.7433**	**0.7545**	89.7	85.8
SUBMULT+NN	**72.1**	**72.9**	**77.0**	**75.6**	72.3	0.7332	0.7477	89.4	**86.8**

Table 3: Experiment Results