文本匹配算法

套路知几许：

1. 1-d 卷积

分为宽卷积和窄卷积两种
宽卷积的结果是：卷积完，宽度不变为 embedding size，但是长变了
目标：

（1）为了让每个词或者字都进行同样卷积的次数（更加平等对待边缘词），不像窄卷积，最中间的词可能被卷积了 k 次，而边缘词只卷积了 1 次

（2）卷积神经网络为了达到深度的目标，可以采用 1-d 宽卷积+移动池化，这样经过一轮的卷积池化之后，图片/文本 的 size 依然还是和原来一样。这样就可以无限堆叠 cnn+pooling

窄卷积
窄卷积卷积完成之后，会造成宽度依然是 embedding size，但是长度变小。但是变小了之后依然还是可以继续 1-d 卷积的，只是越卷积长越小，意味着 1-d 卷积的层是有限的。

2. 输入层

虽然 dssm 及其改进算法是 deep learning 做 qq 匹配的鼻祖，但是由于 dssm 出生的时候 word2vec 还没有出生。导致其输入是 one-hot 的优化算法。现在都用 word2vec 了

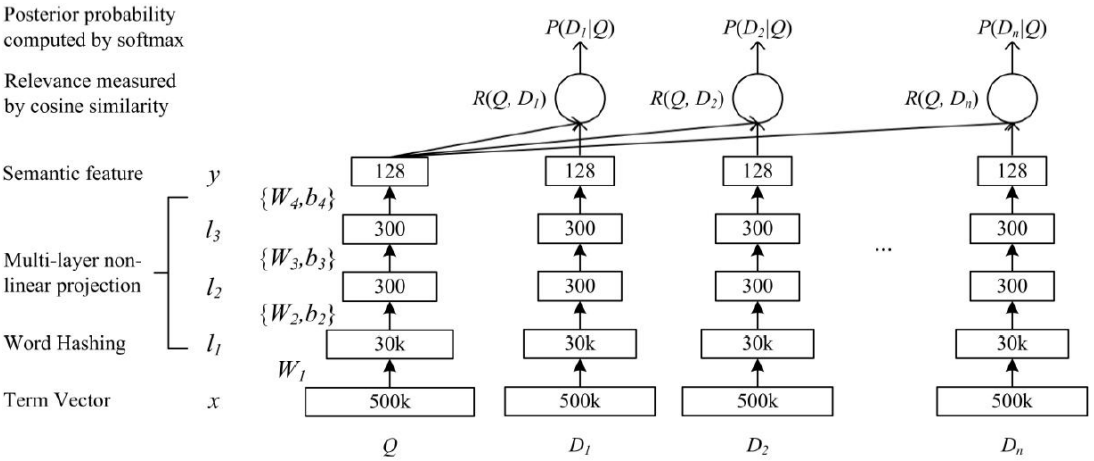
3. 文本表示层

Cnn、rnn、lstm、gru、attention、match paramid

4. 文本交互

（1）一些算法直到拼接特征进行分类前再将其拼接，之前特征之间无交叉
譬如：
Arci
Dssm
（2）也可以在前面就进行文本之间的交互

DSSM 算法

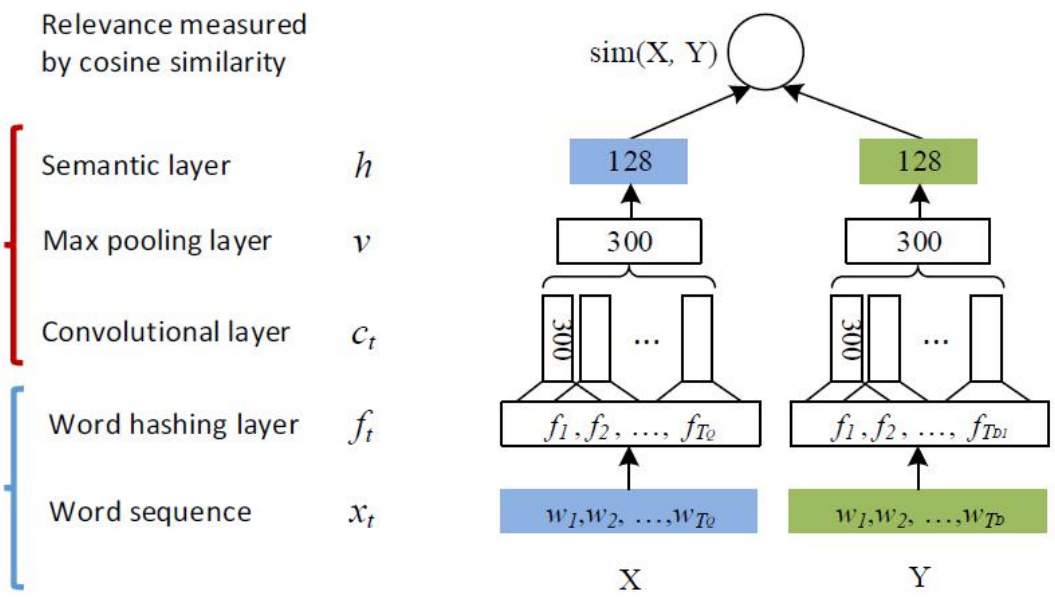


输入跟 CBOW 一样。
将字做 one-hot 表示。第一层将一句话中出现的词所对应的 one-hot 表示进行加和得到一个向量表示。
然后全连接-激活-全连接-激活最终得到一句话的 128 维的向量表示。激活函数采用 tanh。
输出层是 Q 和 D+计算 cosine 相似度，和 D-计算 cosine 相似度，负采样算法采样出 4 个负样本（论文里的超参数，可调）。
这样可以计算出一个四维的向量表示。
计算 softmax。

$$L(\Lambda) = -\log \prod_{(Q,D^+)} P(D^+|Q)$$

损失函数为：
通过反向传播即可得到参数。

CNN-DSSM
相对与 DSSM 而言：
1）输入层做了变化。 词向量表示
2）表示层发生了变化。 卷积神经网络
相对于 textcnn，cnn-dssm 的卷积核仅用了 3 维，意味着三个词三个词的卷积，作者用的是字向量。
输出层和 dssm 一致。
优点：
相对于 bow 而言，用了更多了上下文信息。但是相对于 rnn 而言，这个信息还不充分。
同时，相对于 transformer 而言深度也不够。
如果句子太长，可用 transformer-xl



LSTM-DSSM

同 cnn-dssm 一致。不过是将 cnn 改成了 lstm

ESIM

这是一个比较复杂的算法。

1. Input Encoding

   词向量作为输入，接入 bilstm

$$\bar{a}_i = BiLSTM(a, i), i \in [1, ..., l_a]$$

$$\bar{b}_j = BiLSTM(b, j), j \in [1, ..., l_b]$$

2. Local Inference modeling

   接入 attention，然后比较 attention 之前和之后的差异

这一层的任务主要是把上一轮拿到的特征值做差异性计算。这里作者采用了attention机制，其中attention weight的计算方法如下：

$$e_{ij} = \bar{a_i}^T \bar{b_j}$$

然后根据attention weight计算出a与b的权重加权后的值，计算方法如下：

$$\tilde{a_i} = \sum_{j=1}^{l_b} \frac{exp(e_{ij})}{\sum_{k=1}^{l_b} exp(e_{ik})} \bar{b_j}, i \in [1, ..., l_a]$$

$$\tilde{b_j} = \sum_{i=1}^{l_a} \frac{exp(e_{ij})}{\sum_{k=1}^{l_a} exp(e_{kj})} \bar{a_i}, j \in [1, ..., l_b]$$

注意，这里计算$\tilde{a_i}$的时候，其计算方法是与$\bar{b_j}$做加权，而不是$\bar{a_i}$本身，$\tilde{b_j}$同理。

得到encoding值与加权encoding值之后，下一步是分别对这两个值做差异性计算，作者认为这样的操作有助于模型效果的提升，论文有有两种计算方法，分别是对位相减与对位相乘，最后把encoding两个状态的值与相减、相乘的值拼接起来。

$$m_a = [\bar{a}; \tilde{a}; \bar{a} - \tilde{a}; \bar{a} \odot \tilde{a}]$$

$$m_b = [\bar{b}; \tilde{b}; \bar{b} - \tilde{b}; \bar{b} \odot \tilde{b}]$$

3. Inference Composition

在这一层中，把之前的值再一次送到了BiLSTM中，这里的BiLSTM的作用和之前的并不一样，这里主要是用于捕获局部推理信息m_a和m_b及其上下文，以便进行推理组合。

最后把BiLSTM得到的值进行池化操作，分别是最大池化与平均池化，并把池化之后的值再一次的拼接起来。

$$V_{a,ave} = \sum_{i=1}^{l_a} \frac{V_a, i}{l_a}, \quad V_{a,max} = max_{i=1}^{l_a} V_{a,i}$$

$$V_{b,ave} = \sum_{j=1}^{l_b} \frac{V_b, j}{l_b}, \quad V_{b,max} = max_{j=1}^{l_b} V_{b,j}$$

$$V = [V_{a,ave}; V_{a,max}; V_{b,ave}; V_{b,max}]$$

4. Prediction

最后把 VVV 送入到全连接层，激活函数采用的是 tanhtanhtanh，得到的结果送到 softmax 层。

Match pyramid

相似度计算可以选：

Indicator Function produces either 1 or 0 to indicate whether two words are identical.

$$\mathbf{M}_{ij} = \mathbb{I}_{\{w_i = v_j\}} = \begin{cases} 1, & \text{if } w_i = v_j \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

One limitation of the indicator function is that it cannot capture the semantic matching between similar words. To tackle this problem, we define $\otimes$ based on word embeddings, which will make the matrix more flexible to capture semantic interactions. Given the embedding of each word $\vec{\alpha_i} = \Phi(w_i)$ and $\vec{\beta_j} = \Phi(v_j)$, which can be obtained by recent `Word2Vec` (Mikolov et al. 2013) technique, we introduce the other two operators: cosine and dot product.

Cosine views angles between word vectors as the similarity, and it acts as a soft indicator function.

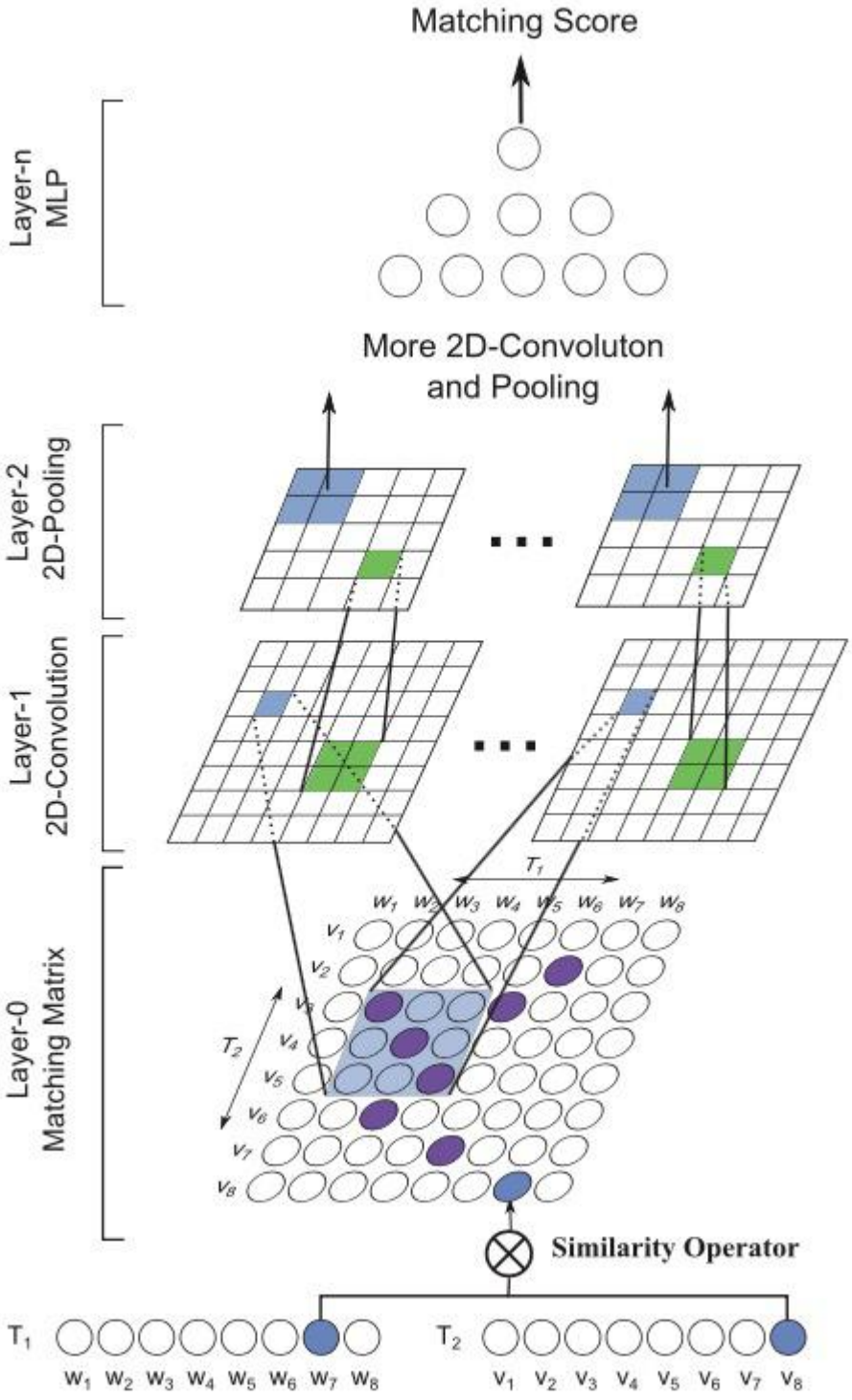
$$\mathbf{M}_{ij} = \frac{\vec{\alpha_i}^\top \vec{\beta_j}}{\|\vec{\alpha_i}\| \cdot \|\vec{\beta_j}\|}, \tag{4}$$

where $\|\cdot\|$ stands for the norm of a vector, and ℓ_2 norm is used in this paper.

Dot Product further considers the norm of word vectors, as compared to cosine.

$$\mathbf{M}_{ij} = \vec{\alpha_i}^\top \vec{\beta_j}. \tag{5}$$

2. 转换成图片之后，接着 2d 卷积、池化、mlp、logistic 之后即可。



ARCI

1. 输入是句子的向量表示
2. 进行 1-d 卷积，没有进行宽卷积

3. 并不是进行卷积，是做了 pooling
4. 中间可以继续 1-d 卷积和 pooling
5. 也可以直接 pooling 成一列向量接 mlp，然后接 logistic 回归

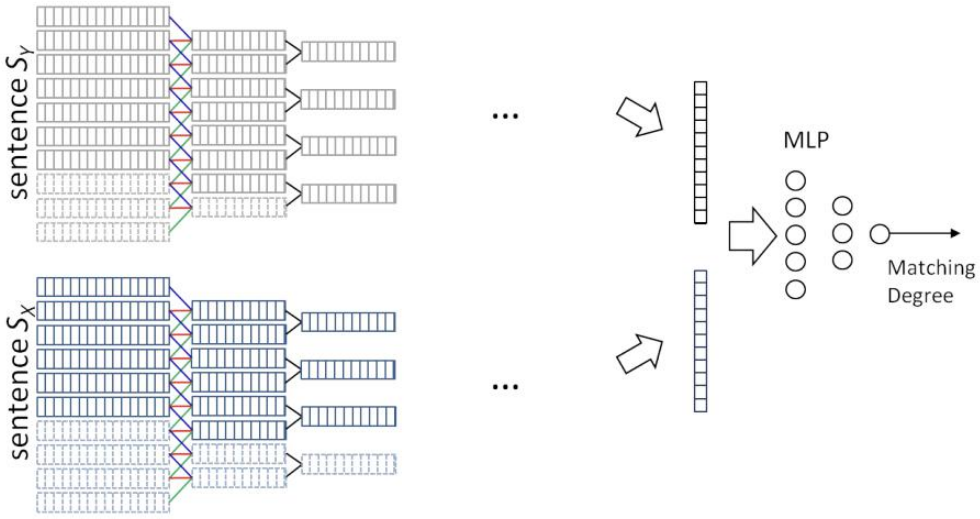


Figure 3: Architecture-I for matching two sentences.

ARCII

1. 两个句子分别做 1-d 卷积，窄卷积不是宽卷积，下面最左边的图有一定误解性，1-d 卷积进行在长的维度上进行卷积，宽的维度为 1
2. 对卷积完的图，做 2-d poolding
3. 进行 2-d 卷积，卷积核的 size 并不是宽度等于 embedding，而是可以将卷积核的 size 定作为超参数
4. 最后接最大池化，接 mlp 即可

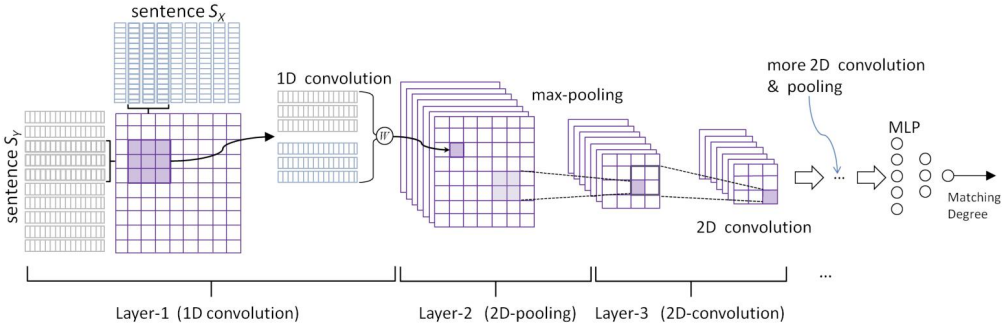


Figure 4: Architecture-II (ARC-II) of convolutional matching model

BCNN

1. 输入层为两个句子的词向量表示
2. 进行宽卷积，即 1-d 卷积，

具体做法是：

1）假如卷积核 size 为 3*1，为了使得第一个词和最后一个词都被卷积 3 次，则，需要在前后各补两列 0, 然后进行卷积操纵, 卷积完成之后得到的 feature map 为 7*embedding size 和 9*embedding size

2）进行 w-ap，1 维的移动池化，池化完成之后，feature map 为 7*embedding size 和 9*embedding size

3）宽卷积+移动池化 这两个操作可以无限加深，然后接全池化接全联接和 logistic regression 即可

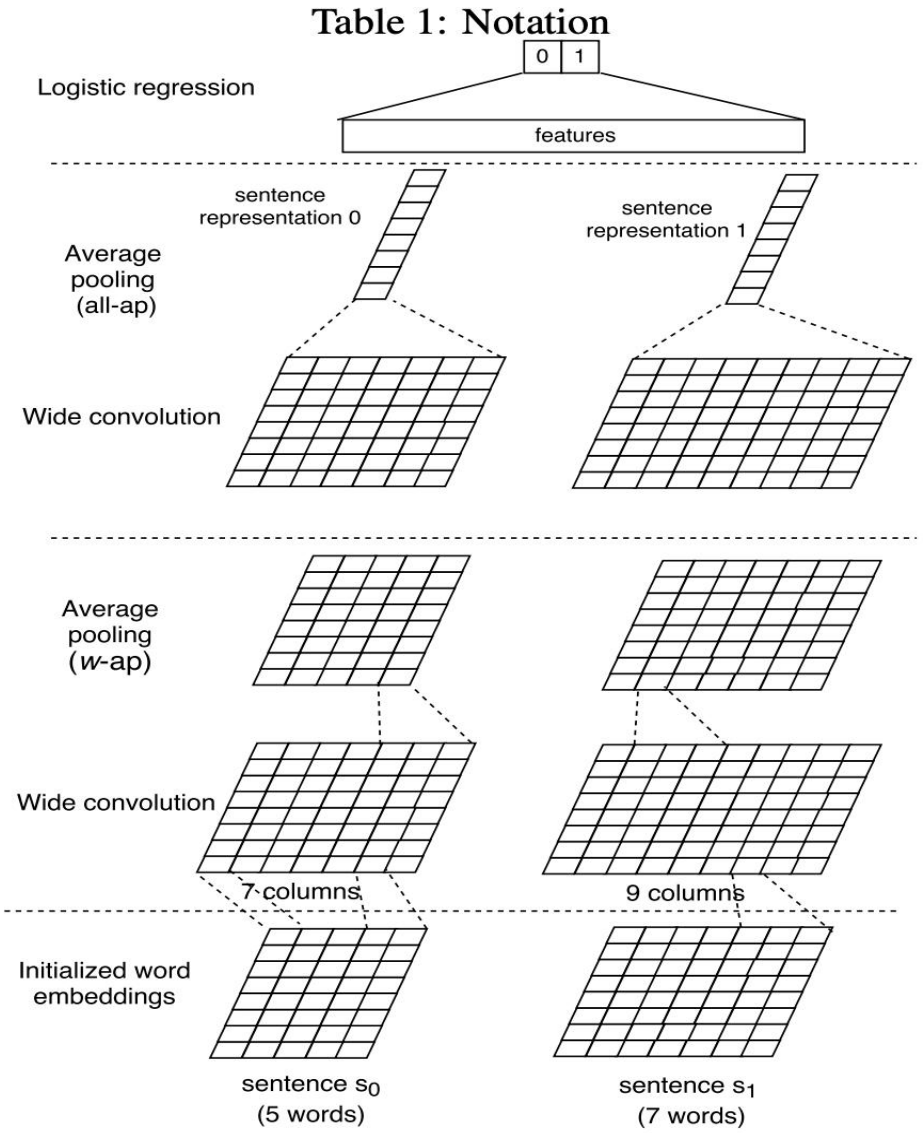


Figure 2: BCNN: ABCNN without Attention

ABCNN1

思想：
宽卷积前做 attention，对 attention 后对矩阵再做宽卷积

1. 输入和 bcnn 一样，是词向量矩阵
2. 两个句子的词向量矩阵做 attention，得到一个 attention 矩阵
3. 两个句子的词向量矩阵分别与 attention 矩阵相乘得到两个矩阵，对这两个矩阵分别进行宽卷积等操作，和 bcnn 一样

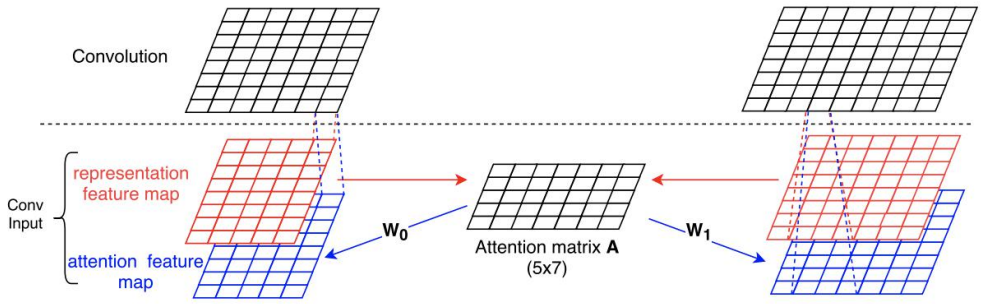
ABCNN2

思想：
对宽卷积做 attention，相对于 abcnn1 来说，attention 更加注重了边界。在 pooling 的时候，因为 pooling 是对原句子矩阵做。

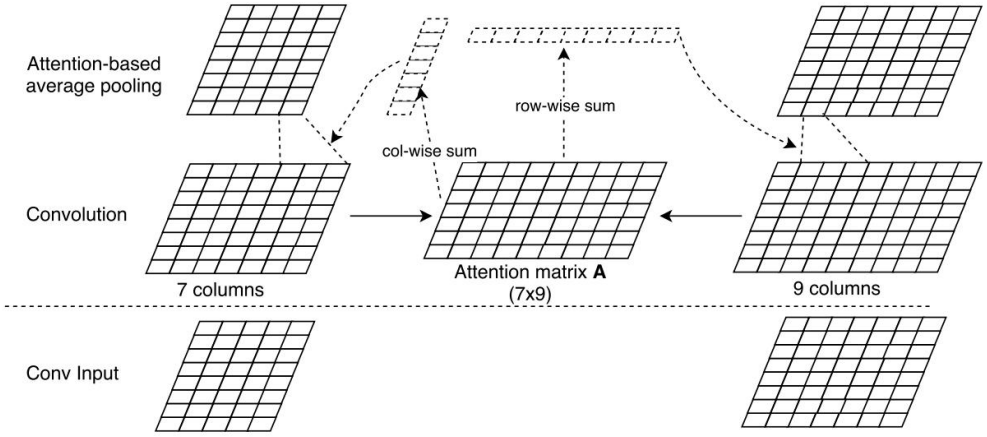
1. 输入和 bcnn 一样
2. 分别对两个 embedding 矩阵做宽卷积
3. 对两个宽卷积矩阵做 attention
4. 对 attention 矩阵对行和列分别进行加和
5. 在移动 poolding 对时候，将 attention 矩阵加和得到对两列矩阵作为权重，来做 pooling
6. 后面对就和 bcnn 一致

ABCNN3

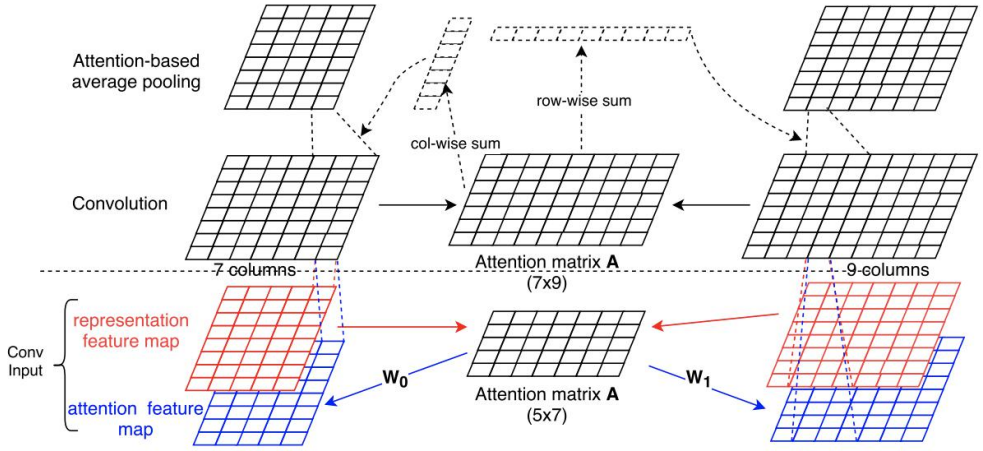
思想：
综合了 abcnn1 和 abcnn2
进行了两次 attention



(a) One block in ABCNN-1



(b) One block in ABCNN-2



(c) One block in ABCNN-3

Figure 3: Three ABCNN architectures