Learning Deep Structured Semantic Models for Web Search using Clickthrough Data

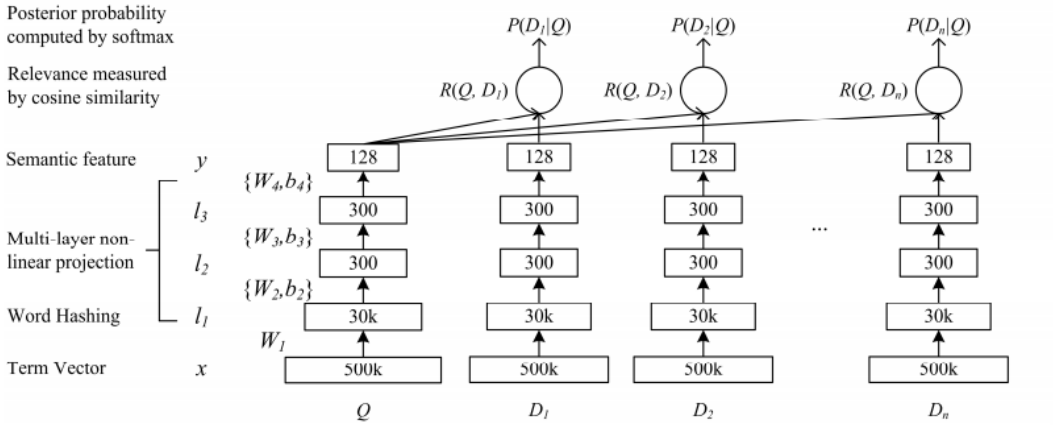


Figure 1: Illustration of the DSSM. It uses a DNN to map high-dimensional sparse text features into low-dimensional dense features in a semantic space. The first hidden layer, with 30k units, accomplishes word hashing. The word-hashed features are then projected through multiple layers of non-linear projections. The final layer's neural activities in this DNN form the feature in the semantic space.

Word Hash:

通过对单词进行 hash，解决数据规模问题。算法的核心出发点就是改变原来单词的 one-hot 编码方式，因为这种编码方式，带来了太多的数据浪费（一个 vector 中只有 1 个位置的信息起作用),所以作者想把 one-hot 改成 multi-hot。具体做法如下：

Query 以及 Doc 使用字符的 Trigram 表示，例如，#Query#，可以表示成 #Qu,Que,#uer,#ery,ry#。对新的 Trigram 表示的单词进行统计,类似于 one-hot,只不过同一个单词，出现过的三元组即在对应位置+1，最后形成一个 multi-hot 的编码。这样的 hash 方法，可能带来的一个问题是碰撞的问题，但是作者统计过，这样的碰撞概率极低，统计结果如下。发生碰撞的几率极小。

Word hashing 有一个明显的好处就是可以解决字典爆炸的问题，即问题的规模不会随着单词各个数增长出现增长，因为所有 Trigram 只有 3 个字符，每个字符出现的可能性是有限的，所以在大规模的 Web Data 是可以轻松应对的。另外一个明显的好处是，这样的 hash 算法，对 Unseen 的单词特别鲁棒，所以可以完美解决任意 Query 的问题。

作者在计算相似度的时候直接采用的最简单的方法，计算两个向量之间的余弦相似度。已知集合 D，其中包括与 Q 相关的文档D^+(D_1,D_2,…,D_k)，以及随机在所有文档中随机选取的四个不相关文档组成的D^-。计算 Q 与文档集 D 中所

有文档的相似度，然后在进行 softmax。作者在计算损失函数的时候，只是计算 D^+中文档的损失。具体如下所示：

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

最小化上述表达式。

Enhanced LSTM for Natural Language Inference

ESIM 模型本身是用于自然语言推理的（即在已知 p 的情况下能够推出 q），但是稍加改造后可以用于短文本匹配的相关任务，并且取得了不错的效果。下面来看一下模型的主要结构：

模型结构主要分为三个部分，分别为 Input Encoding,Local Inference Modeling,Inference Composition。

Input Encoding。按照作者在论文中的描述，首先将短文本进行分词，然后利用预训练好的词向量，将文本转为词向量的形式，随后利用 Bi-LSTM 进行特征提取。这里要说明一下，我们可以不使用预训练好的词向量，可以随机初始化一个词典，然后通过 look_up 函数进行查找，在训练模型的过程中不断优化这个词典。序列长度较长的时候可以试试谷歌的 Transformer 作为特征提取器。

$$\bar{\mathbf{a}}_i = \mathbf{BiLSTM}(\mathbf{a}, i), \forall i \in [1, \ldots, \ell_a], \quad (1)$$

$$\bar{\mathbf{b}}_j = \mathbf{BiLSTM}(\mathbf{b}, j), \forall j \in [1, \ldots, \ell_b]. \quad (2)$$

其中 l_a 为序列 a 的长度，l_b 为序列 b 的长度。

Local Inference Modeling。作者在这一部分首先计算了两个序列间每个词语的相似度：

$$e_{ij} = \bar{\mathbf{a}}_i^T \bar{\mathbf{b}}_j.$$

得到一个二维的相似矩阵，其中 e_ij 为词语 a_i 和 b_j 之间的相似度，再结合 a，b 随后生成彼此相似性加权后的句子，维度保持不变。

$$\tilde{\mathbf{a}}_i = \sum_{j=1}^{\ell_b} \frac{\exp(e_{ij})}{\sum_{k=1}^{\ell_b} \exp(e_{ik})} \bar{\mathbf{b}}_j, \forall i \in [1, \ldots, \ell_a], \quad (12)$$

$$\tilde{\mathbf{b}}_j = \sum_{i=1}^{\ell_a} \frac{\exp(e_{ij})}{\sum_{k=1}^{\ell_a} \exp(e_{kj})} \bar{\mathbf{a}}_i, \forall j \in [1, \ldots, \ell_b], \quad (13)$$

随后再计算 a 和 align 之后的 a 的差和点积，体现一些差异性，最后再进行一些拼接：

$$\mathbf{m}_a = [\bar{\mathbf{a}}; \tilde{\mathbf{a}}; \bar{\mathbf{a}} - \tilde{\mathbf{a}}; \bar{\mathbf{a}} \odot \tilde{\mathbf{a}}], \quad (14)$$

$$\mathbf{m}_b = [\bar{\mathbf{b}}; \tilde{\mathbf{b}}; \bar{\mathbf{b}} - \tilde{\mathbf{b}}; \bar{\mathbf{b}} \odot \tilde{\mathbf{b}}]. \quad (15)$$

Inference Composition。最后一部分则相对来说比较简单了，再一次用到 Bi-lstm 对 m_a, m_b 进行特征提取。然后同时运用 MaxPooling 和 AvgPooling 进行池化操作,作者指出这样做效果更好：

$$\mathbf{v}_{a,\text{ave}} = \sum_{i=1}^{\ell_a} \frac{\mathbf{v}_{a,i}}{\ell_a}, \quad \mathbf{v}_{a,\max} = \max_{i=1}^{\ell_a} \mathbf{v}_{a,i}, \quad (18)$$

$$\mathbf{v}_{b,\text{ave}} = \sum_{j=1}^{\ell_b} \frac{\mathbf{v}_{b,j}}{\ell_b}, \quad \mathbf{v}_{b,\max} = \max_{j=1}^{\ell_b} \mathbf{v}_{b,j}, \quad (19)$$

$$\mathbf{v} = [\mathbf{v}_{a,\text{ave}}; \mathbf{v}_{a,\max}; \mathbf{v}_{b,\text{ave}}; \mathbf{v}_{b,\max}]. \quad (20)$$

最后再将 v 通过一个全连接层，加上 tanh 激活函数，并进行 softmax 进行分类。

Learning to Rank Short Text Pairs with Convolutional Deep Neural Networks

本文介绍如何利用 CNN 来进行短文本匹配任务，非常具有代表性，后续许多相关的工作都是在这个模型上的一些改进，现在来看一下它是如何进行文本匹配的。一共从这四个方面展开说明：Sentence model，Our architecture for matching text pairs，Training，Regularization。

Sentence model：

这一部分的主要工作是利用 CNN 的相关结构对输入的句子进行编码。Sentence model 的输入是由$[w_1, .., w_{|s|}]$,组成的词串，所有输入构成了一个大小为 V 的词典。其中每个单词w_i都用分布式的向量来表示，向量在 a word embeddings matrix W 中 lookedup 得到，其中$W \in R^{d*V}$，d 为词向量的维度，W 可以为预训练好的，也可以为随机初始化的，然后再训练的过程中进行调整。因此，对于任意一句话 S，可以表示为：

$$\mathbf{S} = \begin{bmatrix} | & | & | \\ \mathbf{w}_1 & \cdots & \mathbf{w}_{|s|} \\ | & | & | \end{bmatrix}$$

其中|S|为句子 S 的长度。

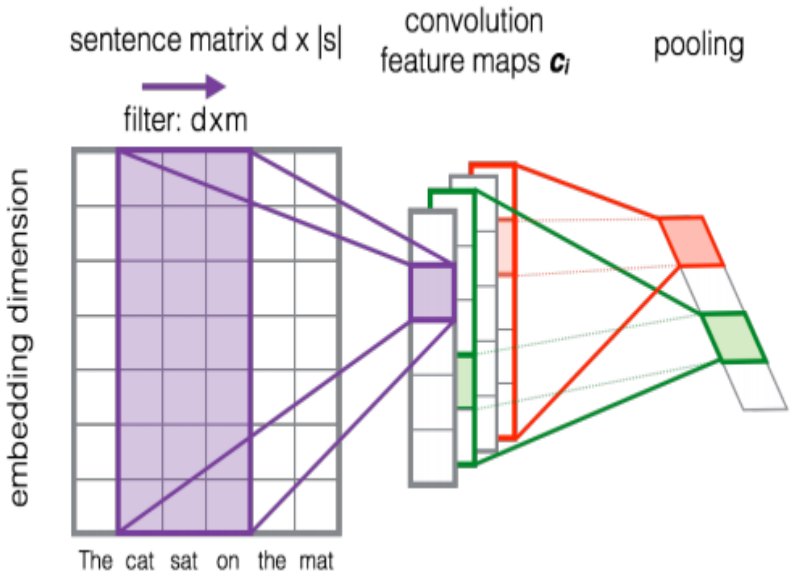
在将每一句话转为类似于 S 的矩阵后，会利用一些卷积层和池化层进行特征提取的相关工作。在这里要先说明一下 narrow convolution 和 wide convolution 的区别，大家熟知的卷积操作层即 narrow convolution，在不进行 padding 的情况下，假设 filter 的大小为 d*m，$S \in R^{d*|S|}$，根据公式可得到特征向量$\mathbf{c} \in \mathbb{R}^{|s|-m+1}$。而 wide convolution 在进行卷积时候，会在矩阵 S 的左右两边分别 padding 上（m-1）*d 的零，因此 S 的维度就变为 d*（|S|+2m-2），在进行卷积后得到的特征向量大小为|S|+m-1。wide convolution 与 narrow convolution 相比有以下几个好处：

① 语句 S 中每个词语进行卷积的次数是一样的，而在 narrow convolution 中，越靠近边缘的词语进行卷积的次数就越少；

② 其次当 S 中词语的个数少于 filter 的 w 时，narrow convolution 是不能返回特

征的，而 wide convolution 则总是可以返回值；

因此作者在这篇文章中采用 wide convolution 的卷积方法，filter 的个数为 n，则经过卷积层后得到的特征维度为 n*（|S|+m-1）。随后会通过一个激活函数，加上 bias。最后再接一个池化层，作者在文章中介绍了三种池化方法，average pooling，max pooling，k-max pooling。在 average pooling 中所有的特征值都被平均了，这样可能会弱化一些比较强的特征值。而 max pooling 则只是考虑了特征最大的值，但是可能会引起过拟合。k-max pooling 是最近几年提出的一种新的 pooling 方法，在一些比较深度的网络中可以被用到。具体的请自行查找相关资料。本文作者采用 max pooling 的方式，如下图所示：



Our architecture for matching text pairs：

这个部分作者利用卷积层得到的结果来计算 query 和 document 的相似度

$$sim(\mathbf{x}_q, \mathbf{x}_d) = \mathbf{x}_q^T \mathbf{M} \mathbf{x}_d, \qquad (2)$$

where $\mathbf{M} \in \mathbb{R}^{d \times d}$ is a similarity matrix. The Eq. 2 can be viewed

在实现的过程中如下所示：

```
# Compute similarity
with tf.name_scope("similarity"):
    W = tf.get_variable(
        "W",
        shape=[num_filters_total, num_filters_total],
        initializer=tf.contrib.layers.xavier_initializer())
    self.transform_left = tf.matmul(self.h_pool_left, W)
    self.sims = tf.reduce_sum(tf.multiply(self.transform_left, self.h_pool_right), 1, keep_dims=True)
```

随后作者将 sim(q,d)和 query，document 提取的特征进行拼接，随后传入到一个

全连接层中，最后将全连接层输出的结果传到一个 softmax classification layer, which generates a distribution over the class labels。

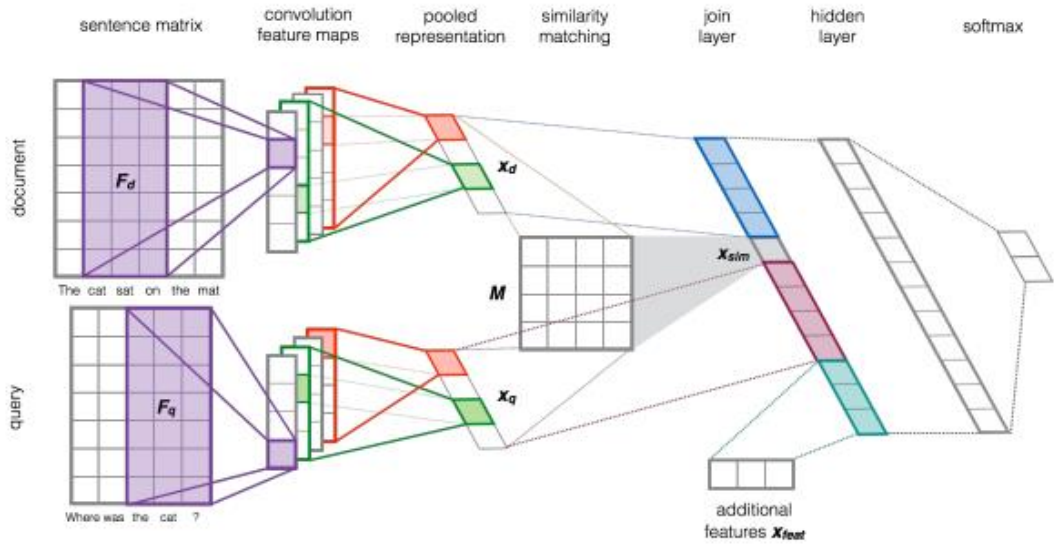


Figure 2: Our deep learning architecture for reranking short text pairs.

ABCNN: Attention-Based Convolutional Neural Network for Modeling Sentence Pairs

这篇文章结合卷积神经网络 CNN 和 Attention 机制，提出应用到文本匹配中。作者首先提出了最简单的 BCNN，然后再基于 Attention 提出了另外三种模型，ABCNN1，ABCNN2，ABCNN3。主要有以下三个方面的贡献：

第一：将 Attention 机制和 CNN 结合应用到句子建模；

第二：提出三种结合 Attention 机制的建模方式；

第三：核心是通过 Attention 机制将原先各自独立的句子，考虑句子之间的相关性，构建出包含句子上下文关系的新的句子模型；

BCNN:

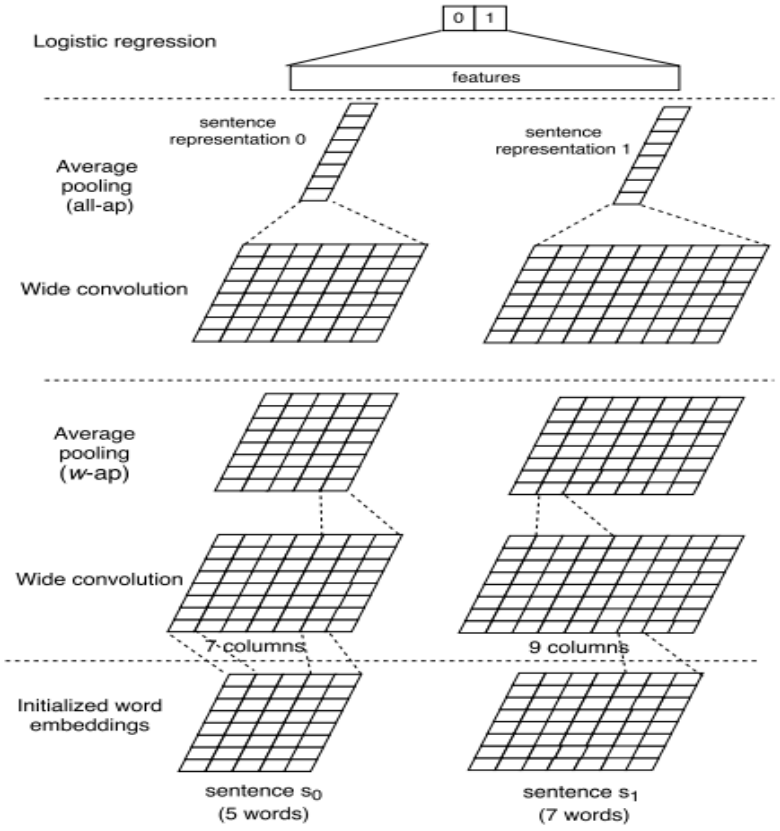
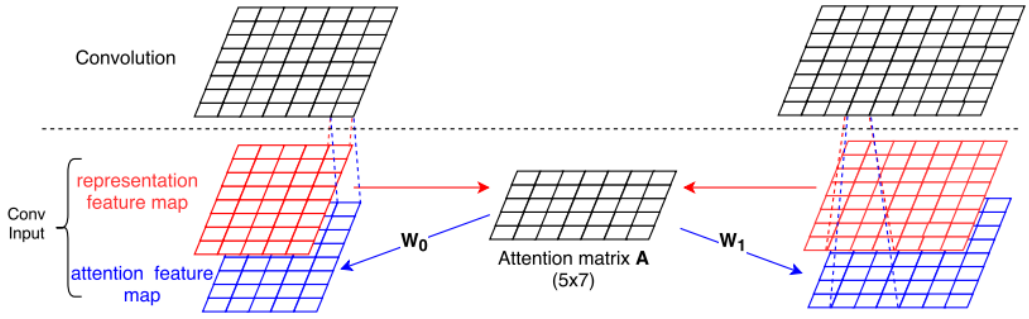


Figure 2: BCNN: ABCNN without Attention

上图所示为 BCNN 模型的图解，最下面的模块是将 sentence 进行分词，然后再转换为对应的词向量，每条语句用一个二维矩阵进行表示。接下来利用 Wide convolution 进行特征提取，w-ap 进行特征刷选。由于 w-ap 后的维度与开始

sentence 表示的维度相同，所以这个部分可以进行多次。随后会再进行一次 Wide convolution 进行特征提取，这次后面接一层 all-ap 进行最后的特征提取。w-ap 与 all-ap 的区别请下去自行查阅。随后将 sentence_1 与 sentence_2 的特征进行拼接，最后再接一个全连接层进行分类。

ABCNN1:



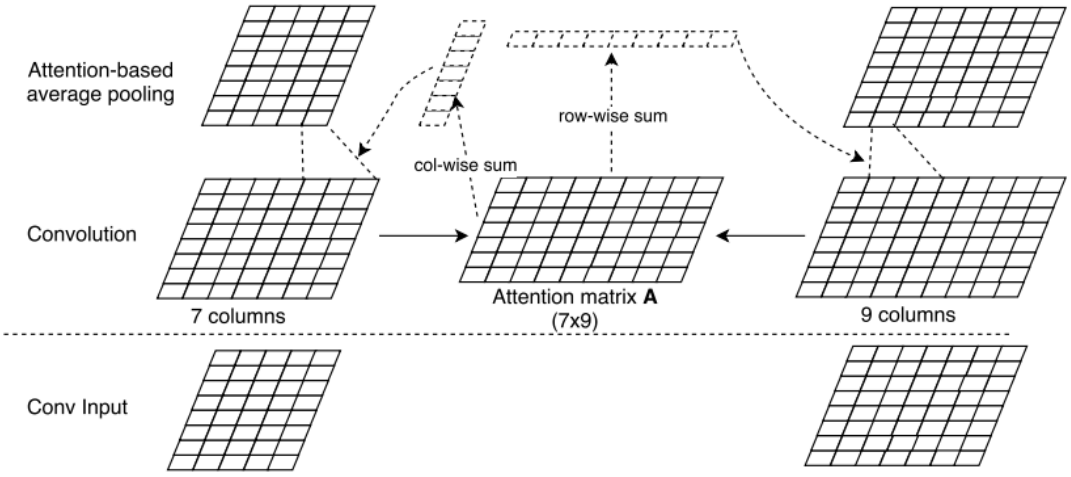
(a) One block in ABCNN-1

ABCNN1 结构中增加了一个抽象层级，就是在原有单词级别上增加了一个短语级别的抽象。单词级别的抽象文中重新命名为 unit, 作为低级别的表示，短语级别的作为更高一级的表示。图中那个红色的与 BCNN 网络中的输入是一样的，是句子的词向量矩阵，两个句子。第一个句子 5 个单词，第二个句子 7 个单词。蓝色的为短语级别高一级的词向量表示。蓝色表示是由 Attention Matrix A 和红色词向量计算生成

Attention Matrix A 是由左右两个句子的情况生成。A 中的 i 列的值是由左边句子中第 i 个单词（unit）的向量与右边句子的 Attention 值分布，最终生成一个 5*7 的矩阵。其中 Attention 值采用如下公式计算：1/(1+|x-y|)，其中|*|为欧式距离。随后再随机初始化两个矩阵 W_0, W_1。然后利用如下公式得到短语级别的表示，随后将两种表示方法进行 concat，得到一个[B,L,D,2]的矩阵，类似于图像分类中的三通道，后面的操作就和 BCNN 一样。

$$\mathbf{F}_{0,a} = \mathbf{W}_0 \cdot \mathbf{A}^\top, \quad \mathbf{F}_{1,a} = \mathbf{W}_1 \cdot \mathbf{A}$$

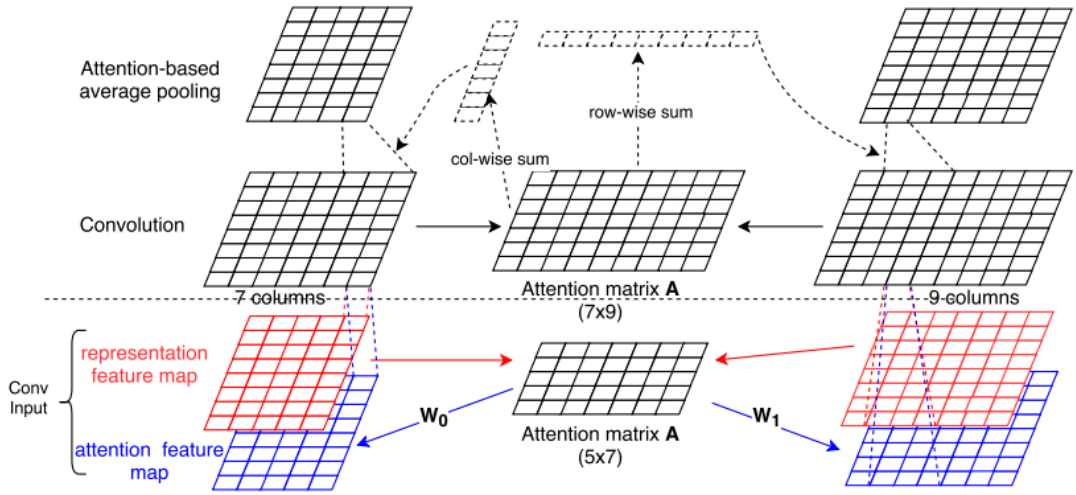
ABCNN2:



ABCNN2 结构如上图所示，在 input 经过 wide concolution 后对两个句子中每个单词进行 attention 计算，得到一个 7*9 的矩阵，计算方法和 ABCNN1 中的一样，其中 A_ij 代表第一个句子中第 i 个词语与第二个句子中第 j 个词语的相似程度。随后再分别对 A 进行按行 sum 和按列 sum，得到一个长度为 7 和长度为 9 的向量。然后再对 wide concolution 后的特征进行 ave-pooling(w-ap)，此时在进行 ave-pooling 时，每列向量如要乘上一个权重，这个权重就是 A 按行（列）sum 得到的向量，后面的步骤与 BCNN 一致。

ABCNN3:

(b) One block in ABCNN-2



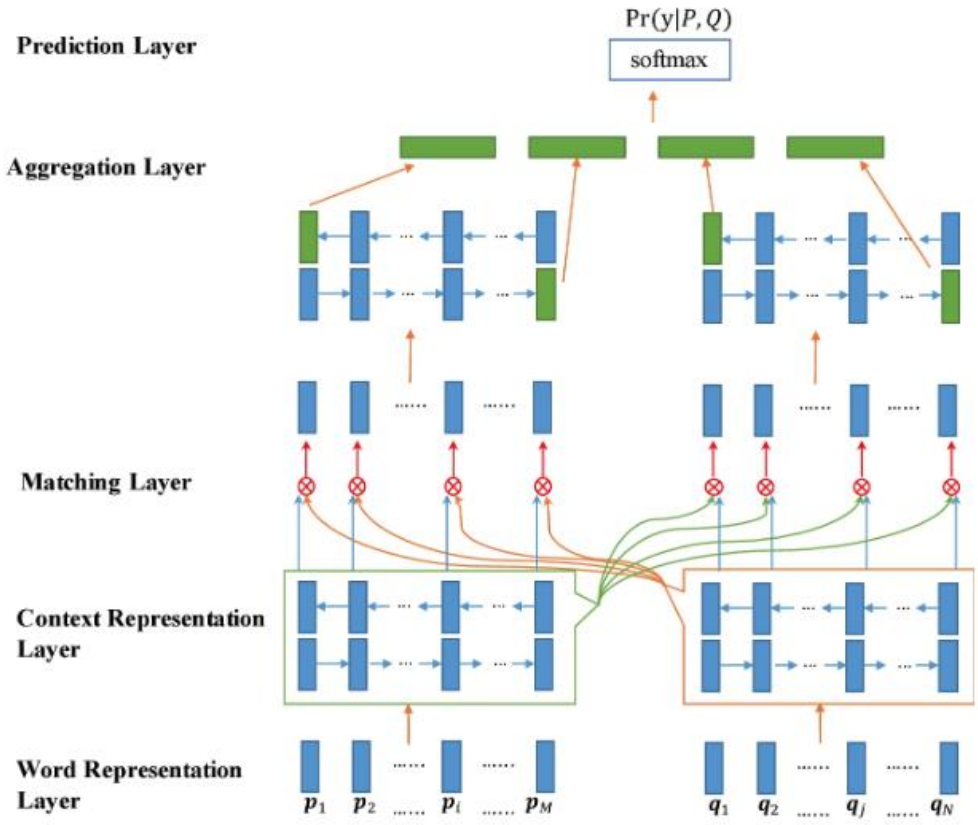
(c) One block in ABCNN-3

ABCNN3 结构如上图所示，它结合了 ABCNN1 与 ABCNN2 的两个 attention 部分，这里就不加详细描述了。

bilateral multi-perspective matching for natural language sentences

在这项工作中，我们提出了一个双边多视角匹配（BiMPM）模型。给定两个句子 P 和 Q，我们的模型首先用 BiLSTM 编码器对它们进行编码。接下来，我们将针对 Q 和 Q 的两个方向 P 上的两个编码语句与 P 进行匹配。在每个匹配方向上，一个句子的每个时间步从多个角度与另一个句子的所有时间步匹配。然后，利用另一个 BiLSTM 层将匹配结果聚合成固定长度的匹配向量。最后，基于匹配向量，通过完全连接的层进行决定。

BiMPM 模型结构图如下所示：



模型一共分为四个模块，分别为上下文表示层，匹配层，聚合层，预测层。现在分别来具体介绍这几个模块的内容：

上下文表示层：如图所示 p_i，q_j 为词语对应的词向量，然后利用 Bi-LSTM 进行上下文的特征提取。

$$\overrightarrow{\boldsymbol{h}}_i^p = \overrightarrow{\text{LSTM}}(\overrightarrow{\boldsymbol{h}}_{i-1}^p, \boldsymbol{p}_i) \qquad i = 1, ..., M$$
$$\overleftarrow{\boldsymbol{h}}_i^p = \overleftarrow{\text{LSTM}}(\overleftarrow{\boldsymbol{h}}_{i+1}^p, \boldsymbol{p}_i) \qquad i = M, ..., 1 \tag{1}$$

Meanwhile, we apply the same BiLSTM to encode Q:

$$\overrightarrow{\boldsymbol{h}}_j^q = \overrightarrow{\text{LSTM}}(\overrightarrow{\boldsymbol{h}}_{j-1}^q, \boldsymbol{q}_j) \qquad j = 1, ..., N$$
$$\overleftarrow{\boldsymbol{h}}_j^q = \overleftarrow{\text{LSTM}}(\overleftarrow{\boldsymbol{h}}_{j+1}^q, \boldsymbol{q}_j) \qquad j = N, ..., 1 \tag{2}$$

匹配层：这是模型的核心层。该层的目标是比较一个句子的每个上下文嵌入（时间步）与另一个句子的所有上下文嵌入（时间步）。我们将两个句子 P 和 Q 在两个方向上进行匹配：将 P 的每个时间步长与 Q 的所有时间步长进行匹配，并将 Q 的每个时间步长与 P 的所有时间步长进行匹配。为了将句子的一个时间步与另一个句子的所有时间步匹配，设计了一个多视角匹配操作⊗。

多视角匹配的具体操作：

作者首先定义了一个多视角余弦匹配函数，来比较两个矢量：

$$\boldsymbol{m} = f_m(\boldsymbol{v}_1, \boldsymbol{v}_2; \boldsymbol{W})$$

其中 W 为 l*d 维的可训练参数，l 为视角数。返回的 m 为一个长度为 l 的向量。$\text{m} = [m_1, m_2, ..., m_k, ..., m_l]$，$m_k \in$ m中的每个元素都是第 k 个视角的匹配值，它由两个加权向量之间的余弦相似度计算得出：

$$m_k = cosine(W_k \circ \boldsymbol{v}_1, W_k \circ \boldsymbol{v}_2)$$

作者一共定义了四种匹配策略来比较一个句子的每个时间步与另一个句子的所有时间步：

（1）完全匹配：在此策略中，将每个正向（或反向）上下文嵌入$\overrightarrow{h}_i^p$（或$\overleftarrow{h}_i^p$）与另一个语句$\overrightarrow{h}_N^q$（或$\overleftarrow{h}_1^q$）的正向（或反向）表示的最后一个时间步进比较。

```
# (batch, seq_len, hidden_size), (batch, hidden_size)
# -> (batch, seq_len, l)
mv_p_full_fw = mp_matching_func(con_p_fw, con_h_fw[:, -1, :], self.mp_w1)
mv_p_full_bw = mp_matching_func(con_p_bw, con_h_bw[:, 0, :], self.mp_w2)
mv_h_full_fw = mp_matching_func(con_h_fw, con_p_fw[:, -1, :], self.mp_w1)
mv_h_full_bw = mp_matching_func(con_h_bw, con_p_bw[:, 0, :], self.mp_w2)
```

其中，mp_matching_func 为匹配函数，mp_w1 为 l*d 的可训练参数。

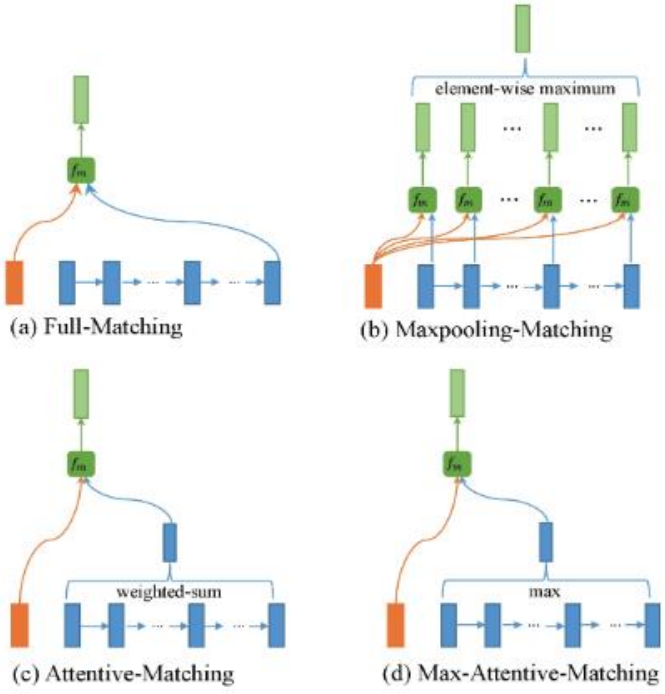
（2）最大池化匹配：在此策略中，每个向前（或向后）上下文嵌入$\overrightarrow{h}_i^p$（或$\overleftarrow{h}_i^p$）

与$\overrightarrow{h}_j^q$（或$\overleftarrow{h}_j^q$）的其他每个向前（或向后）上下文嵌入进行比较，只保留每个维度的最大值。

$$\overrightarrow{\boldsymbol{m}}_i^{max} = \max_{j\in(1\ldots N)} f_m(\overrightarrow{\boldsymbol{h}}_i^p, \overrightarrow{\boldsymbol{h}}_j^q; \boldsymbol{W}^3)$$
$$\overleftarrow{\boldsymbol{m}}_i^{max} = \max_{j\in(1\ldots N)} f_m(\overleftarrow{\boldsymbol{h}}_i^p, \overleftarrow{\boldsymbol{h}}_j^q; \boldsymbol{W}^4) \qquad (6)$$

where $\max_{j\in(1\ldots N)}$ is element-wise maximum.

随后将 p 与 q 顺序调换，又可以得到两个(batch, seq_len, l)特征。



（3）注意匹配：在此策略中，每个向前（或向后）上下文嵌入$\overrightarrow{h}_i^p$（或$\overleftarrow{h}_i^p$）与$\overrightarrow{h}_j^q$（或$\overleftarrow{h}_j^q$）的其他每个向前（或向后）上下文嵌入之间的余弦相似度。

$$\overrightarrow{\alpha}_{i,j} = cosine(\overrightarrow{\boldsymbol{h}}_i^p, \overrightarrow{\boldsymbol{h}}_j^q) \qquad j = 1, \ldots, N$$

$$\overrightarrow{\boldsymbol{h}}_i^{mean} = \frac{\sum_{j=1}^N \overrightarrow{\alpha}_{i,j} \cdot \overrightarrow{\boldsymbol{h}}_j^q}{\sum_{j=1}^N \overrightarrow{\alpha}_{i,j}}$$

$$\overrightarrow{\boldsymbol{m}}_i^{att} = f_m(\overrightarrow{\boldsymbol{h}}_i^p, \overrightarrow{\boldsymbol{h}}_i^{mean}; \boldsymbol{W}^5)$$

根据上面公式可以得出，计算 p 前向 i 时刻隐状态$\vec{h}_i^p$与 q 每个前向时刻隐状态$\vec{h}_j^q$的余弦相似度，然后再进行平均得到$\vec{h}_i^{mean}$，最后利用 mp_matching_func 进行匹配。同样可以得到其它 3 个注意匹配特征。

（4）最大注意匹配：该策略与注意匹配策略类似。然而，我们不是将所有上下文嵌入的加权总和作为注意向量，而是选择具有最高余弦相似度的上下文嵌入作为注意向量。

聚合层：该层被用来将两个匹配向量序列聚合成一个固定长度的匹配向量。我们利用另一个 BiLSTM 模型，并将其分别应用于两个匹配向量序列。然后，我们通过连接来自 BiLSTM 模型的最后一个时间步骤的（四个绿色）向量来构建固定长度的匹配向量。

由于匹配层中，每种匹配方式都能得到[B,L,l*4]个特征，4 种匹配方式一共可以得到[B,L,l*4*4]。P 和 q，每个都是[B,L,l*4*2]，将其进行 cancat 就得到 p 和 q 的特征。然后分别将其输入到一个 Bi-lstm 模型中，选取每个方向最后一个时刻的步骤，一共四个（p 和 q 各两个），然后将其拼接用与最后的预测。

预测层：我们使用两层前馈神经网络来消耗固定长度的匹配矢量，并在输出层中应用 sof tmax 函数。

代码解释参看：https://zhuanlan.zhihu.com/p/50184415

有关 WMD 的相关介绍：

SentenceA：**********

SentenceB：**********

Step1：将 Sentence 进行分词（例如 jieba 分词），去停用词；

SentenceA：[word1，word2，…，word_m]

SentenceB：[word1，word2，…，word_n]

Step2：将分词后的词语利用 word2vec（或）训练的词向量进行转换；

SentenceA：[vector1，vector2，…，vector _m]

SentenceB：[vector1，vector2，…，vector_n]

Step3：首先定义 word travel cost 目前为两个矢量的欧式距离，公式如下：

$$c(i,j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2$$

然后定义 document 中每个 word 的权重，公式如下：

$$d_i = \frac{c_i}{\sum_{j=1}^{n} c_j}.$$

其中c_i为 word_i 在这个 document 中的词频。

Step4：最优化如下线性问题：

$$\begin{aligned}
&\min_{\mathbf{T}\geq 0} \sum_{i,j=1}^{n} \mathbf{T}_{ij} c(i,j) \\
&\text{subject to: } \sum_{j=1}^{n} \mathbf{T}_{ij} = d_i \quad \forall i \in \{1,\dots,n\} \qquad (1) \\
&\qquad\qquad \sum_{i=1}^{n} \mathbf{T}_{ij} = d'_j \quad \forall j \in \{1,\dots,n\}.
\end{aligned}$$

其中T_{ij}为表示 how much of the word i in document_i travels to word j in document_j。

解决 WMD 最优方法的时间复杂度为$p^3 \log p$，所消耗的时间比较长，因此出现了另外一种已牺牲精确度来加快速度的方法 WCD：

$$\begin{aligned}
&\sum_{i,j=1}^{n} \mathbf{T}_{ij} c(i,j) = \sum_{i,j=1}^{n} \mathbf{T}_{ij} \|\mathbf{x}_i - \mathbf{x}'_j\|_2 \\
&= \sum_{i,j=1}^{n} \|\mathbf{T}_{ij}(\mathbf{x}_i - \mathbf{x}'_j)\|_2 \geq \left\| \sum_{i,j=1}^{n} \mathbf{T}_{ij}(\mathbf{x}_i - \mathbf{x}'_j) \right\|_2 \\
&= \left\| \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \mathbf{T}_{ij} \right) \mathbf{x}_i - \sum_{j=1}^{n} \left(\sum_{i=1}^{n} \mathbf{T}_{ij} \right) \mathbf{x}'_j \right\|_2 \\
&= \left\| \sum_{i=1}^{n} d_i \mathbf{x}_i - \sum_{j=1}^{n} d'_j \mathbf{x}'_j \right\|_2 = \|\mathbf{X}\mathbf{d} - \mathbf{X}\mathbf{d}'\|_2.
\end{aligned}$$

这种方法的时间复杂度十分低；此外还有另一种方法 RWMD，如有兴趣的话就自行查阅。

论文名称：From Word Embeddings To Document Distances

实现：https://blog.csdn.net/Ding_xiaofei/article/details/81034058