**任务：**

做一个问答系统，根据输入的内容，提出的问题，得到一个答案。详细见下面数据集的介绍。

**数据集：**

前期采用的数据集是英文的，因为中文数据集少，而且训练数据量太小。使用的是Facebook 的公共数据**The (20) QA bAbI tasks**，官方下载链接如下：

<https://research.fb.com/downloads/babi/>

此数据集包含20个任务，每种任务相似但又有些许差异。每种任务包含10,000多对问答对，每个问答对可以分成三部分：内容、问题和答案。内容可以是一个句子、几个句子或者一个段落一篇文章。问题是一个句子。答案是一个单词或者一个词组，并且答案一定是在内容中出现的。以下为一种数据集中一个样本的表现形式：

1 John travelled to the hallway.

2 Mary journeyed to the bathroom.

3 Where is John? hallway 1

可见输入内容是三个句子，问题是一个句子，答案为一个单词。其中答案多数情况下不是根据一个句子直接产生的，而是需要通过多个句子相互推理得出。以上为最简单的问答对，内容只有2句，有些任务的内容可能多达80句，对模型的推理能力要求更高。推理的逻辑难度也有很大的差异，有些只需要推理一次，相互推理的句子间隔很小，有些问答对的答案需要根据多个句子推理，句子和句子之间间隔很大，因此如何捕获句子间的关系，并保留有效信息是个难点。（数字1叫做支持句子编号，即答案是根据哪些编号的句子按顺序推理出来的，此次的模型不会使用到该部分信息。）以下为难度较大的例子：

1 Sandra journeyed to the bedroom.

2 Daniel travelled to the hallway.

3 Mary moved to the bathroom.

4 Daniel picked up the apple there.

5 Sandra travelled to the office.

6 Mary picked up the football there.

7 Sandra went back to the hallway.

8 Daniel discarded the apple.

10 Mary moved to the office.

11 Sandra grabbed the apple there.

13 Daniel journeyed to the kitchen.

14 Sandra left the apple there.

15 Where is the apple? hallway 14 7

可见输入的句子个数明显增加，需要句子间的推理，并且需要推理的句子之间间隔了多个无效句子。

**模型：**

模型的设计思路来自多篇论文，链接和介绍如下：

Dynamic Memory Network (DMN): <https://arxiv.org/pdf/1506.07285.pdf>

Dynamic Memory Network Plus (DMN Plus): <https://arxiv.org/pdf/1603.01417.pdf>

Tree-LSTM: <http://www.aclweb.org/anthology/P17-1129>

Dependency Parser: <https://cs.stanford.edu/~danqi/papers/emnlp2014.pdf>

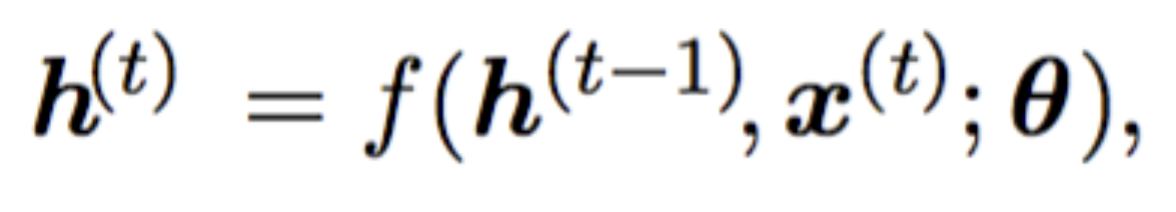
基模型（DMN）

DMN作为此次模型的基本模型，此后的改动都在其基础之上。以下详细介绍Dynamic Memory Network (DMN) 模型：

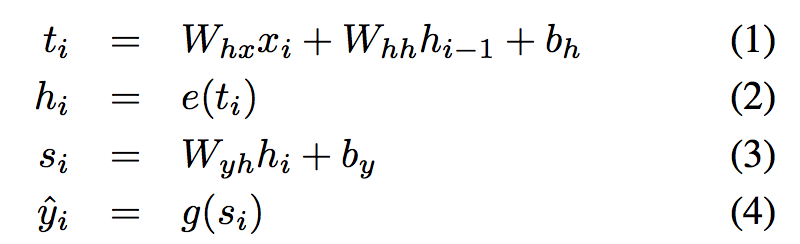
DMN由4部分组成，分别叫做输入模块（Input Module），问题模块（Question Module），事件记忆模块（Episodic Memory Module），回答模块（Answer Module）

在介绍模型之前，先对RNNs做基本的介绍，熟悉可跳过：

RNN可以看做每层网络结点有连接的神经网络。网络会对前面的信息进行记忆并应用于当前输出的计算中，即上一层隐藏层的输出会作为当前隐藏层输入的一部分，因此可以认为当前隐藏层输出的状态包含了前面所有词向量的信息。

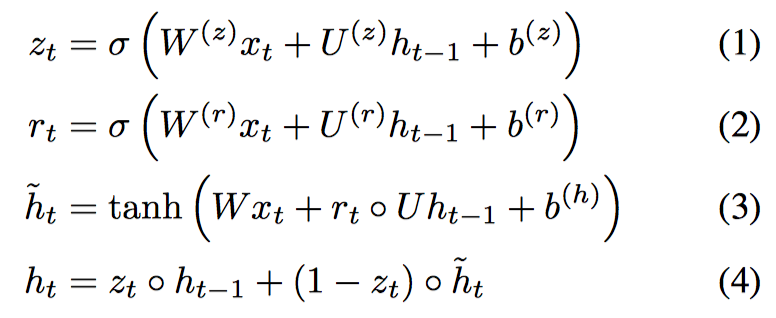


h(t)表示当前隐藏层的输出，h(t-1)表示上一时刻的隐藏层输出，x(t)表示当前时刻的输入词向量（代表当前单词的词向量），θ为函数所需要的参数（为矩阵）。可以理解为h(t)包含了前面所有单词的信息。具体的计算过程如下式所示：



Whx , Whh , Wyh , bh , by就是之前的θ，分别代表三个权重参数和偏置参数。e和g代表预定义好的非线性函数。

GRU为RNN的改进版，能够解决梯度消失的问题，因此能够更好的保留输入信息。



上式为GRU的计算公式，相比于传统的RNN，隐藏层输出的计算新增了两个参数zt , rt ，分别代表更新门（Update Gate）和重置门（Reset Gate）以及t。t代表对当前输入和上一个时刻隐藏状态的总结，zt代表有多少ht-1和t对当前的隐藏层状态带来影响，rt负责决定ht-1对当前t的重要性有多大。除此之外，GRU不存在memory的概念，而是简单的上一时刻的隐藏状态全部用作当前时刻的输入。

由此可见，GRU结构通过更新门和重置门对传入下一层网络的权重加以控制，结构上比传统RNN更复杂，也因此能更好的解决长句子信息消失的问题。而LSTM相比于GRU又多了一层控制：设置一个参数控制上一个时间点的隐藏层状态对当前隐藏层状态的影响权重。

除此之外，还引入了双向RNN的结构。以上所说的情况，当前的隐藏层状态只受之前的隐藏，可以理解为由前向后的传播。双向结构引入了由后向前的传播，使当前隐藏层状态不但受之前的影响，也受到未来时刻的影响，使信息的传播更为充分。此外，RNN还能设置为多层结构，根据需求取最后一层的状态或对每层状态加权最为最后的隐藏层状态。

输入模块（Input Module）：

做的工作是把输入的文本（在此数据集中为多个句子的集合）转换为向量表达式。

现在每个句子都是由若干个单词组成的，首先使用词嵌入层（Word Embedding）把单词转换为二维的张量（Tensor，GPU计算时支持的一种数据结构，可以理解为list。）假设单词个数为8，词嵌入层指定的词向量维度为80，那么句子进过词嵌入层后的输出即为 8 \* 80的二维Tensor。（此时暂时不考虑Batch Size的使用，此步骤是为了加快训练过程。若使用，输出为Batch size \* 8 \* 80的三维Tensor）然后把此二维Tensor输入Recurrent Neural Network （RNN），目前有多种RNN结构的变体，比如常见的LSTM,GRU等。LSTM的结构较GRU复杂，DMN的作者证明使用GRU能够得到和LSTM相近的效果，所以采用单向GRU即可。

因此，输入模块的输出即为GRU的输出，在不考虑batch size的情况下，当输入为一句话时，输出是个二维的Tensor，其中保留了对句子中所有单词的信息总结。当输入的句子是多个句子时，先对多个句子进行拼接，在每个句子尾部增加句末标识符，最后输出标识符处的隐藏层状态进行拼接。以下分析使用batch size时Tensor的维度：

在训练过程中引入batch size是为了加快计算，能够每次训练多个问答对，因此通常都会引入batch size。因为GRU的输入为Tensor格式，所以对于不同长度的句子以及句子个数不同的batch需要进行padding，通常的做法是在有效信息后面补0.

假设batch size为2，词向量维度为80，padding后每个contexts最多的句子个数为6.那么GRU的输入是：代表输入内容的2 \* 6 \* 80的tensor和代表初始隐藏层的2 \* 2 \* 80 Tensor，第二个维度的2是因为双向GRU。GRU的输出也是两部分，分别代表cell和hidden。Cell 储存每次GRU迭代的隐藏层状态，维度为2 \* 6 \* 160，第三个维度是因为考虑了双向GRU。hidden代表最后一次隐藏层的状态，维度为 2 \* 2 \* 80，第二个维度是因为考虑了双向GRU。我们把cell分成两个2 \* 6 \* 80的tensor相加最后输入模块的输出。

问题模块（Question Module）：

问题模块的特点是输入只可能是一个句子，采用的结构和输入模块相同，是输入模块句子个数为一的情况。问题模块的输出也被用作Episodic Memory Module中memory的初始状态。对应的Tensor维度为 1 \* 2 \* 80 （句子个数\*batch size\*词向量维度）

事件记忆模块（Episodic Memory Module）：

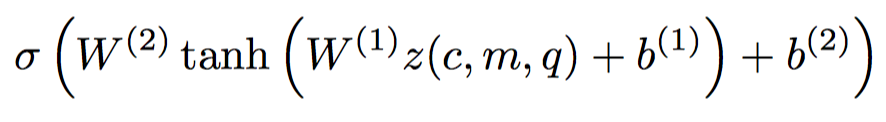
由注意力机制（Attention Mechanism）和记忆更新机制组成（Memory Update Mechanism）组成。

Attention Mechanism：

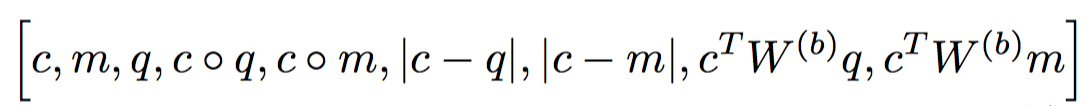
Attention Mechanism的目的是通过不断的迭代，找到那些跟产生最终答案相关的句子。第i次的迭代可由下式表示：



可见每次的gate 是输入模块的输出 ct，上一个时刻的memory mi-1，和问题模块的输出q 的函数，当t为1时，mi-1即为q，具体计算函数如下：



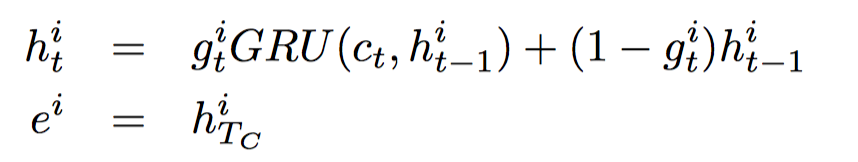
其中的z(c, m, q)是由c, m, q组成的特征向量：



°表示两个向量之间的点积，最后把所有向量拼接在一起，目的是获得输入三个向量之间的关系。函数z的输出Tensor维度应为2 \* 6 \* 560 （batch size \* 句子个数 \* 词向量维度X7个向量的组合）。最后gt为2\*6的二维Tensor，代表每个batch中每个句子的权重，权重越高的句子为越有可能产生答案的句子，在下一次迭代中会给予更高的关注。

和记忆更新机制组成（Memory Update Mechanism）：

使用GRU计算每一次迭代的事件（episode），公式如下：

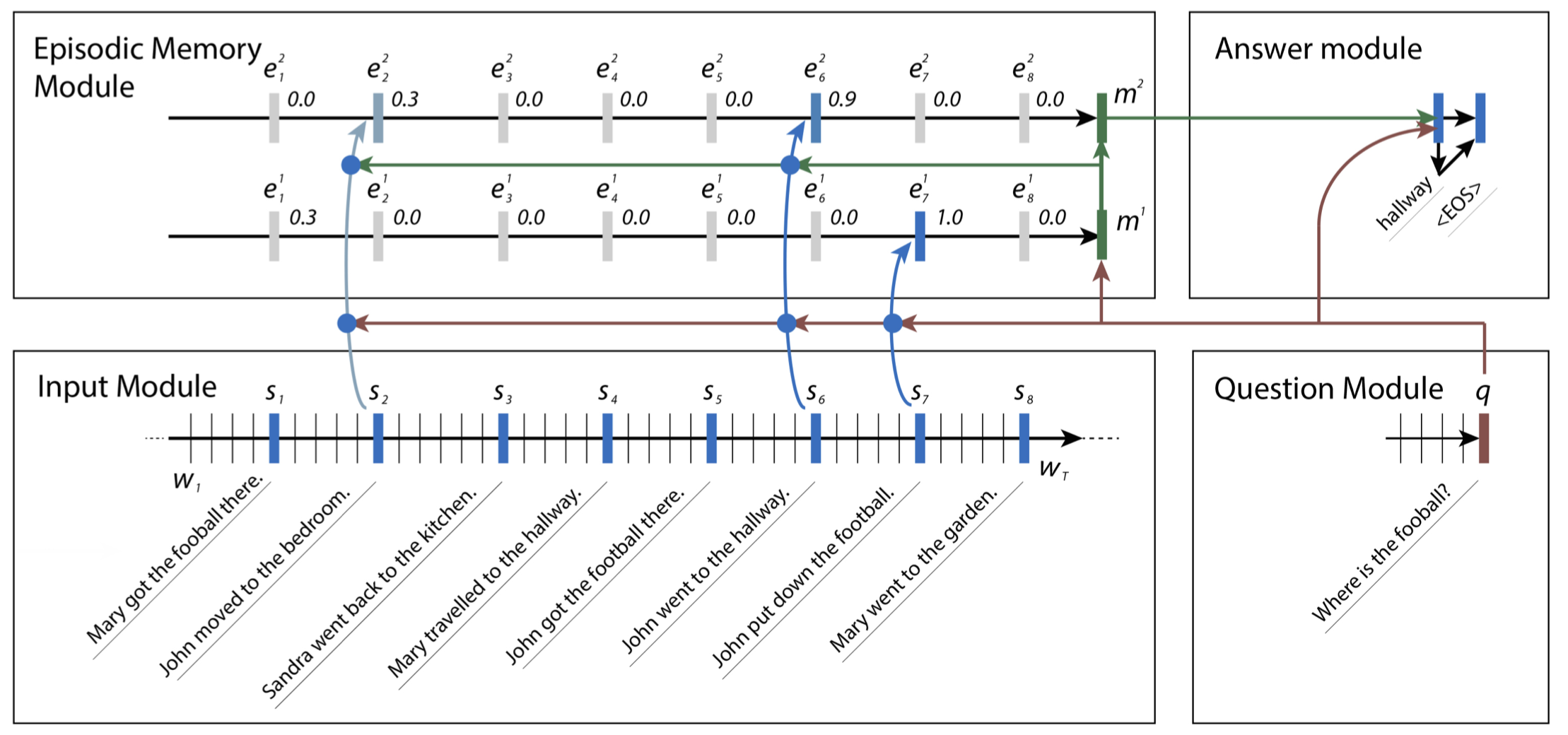


上标i表示第i次迭代，ct为输入模块的输出，ht-1为上一时刻的隐藏层状态，初始状态可以用全为0的tensor表示，此时的GRU也是对输入句子的一次迭代。把最后时刻的隐藏层状态作为episode。迭代两个式子i次，最后一次的episode作为Memory Update Mechanism部分的输出。输出的Tensor维度为 2 \* 1 \* 80.

回答模块（Answer Module）:

把Episodic Memory Module的输出和Question Module的输出拼接成 2\*160的Tensor。使用交叉熵损失函数作为loss进行反向传播训练。预测字典中每个单词的概率，把最大概率的单词作为结果。

以上为该问答系统的基模型，大致流程可用下图表示。



可见Input Module把每个句子编码成S1至S8的向量并流入Episodic Memory Module，结合Question Module的向量形式输入，不断的自我迭代更新每个句子的权重，忽略权重值接近0的句子，最后把关注点放在了S2和S6。把最后生成的memory作为Answer Module的输入生成答案。

模型更新1（DMN+）

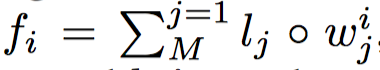
DMN模型在实际表现中存在多方面的问题。首先，其Input Module采用的是单向GRU，每个时刻的隐藏层状态包含了之前句子（单词）的总结，却没有考虑后面句子的信息。此外，当句子个数很多或句子很长时，让最后一个时刻的隐藏状态包含前面所有句子（单词）的信息很困难，容易导致信息丢失的问题。因此需要对Input Module改进。其次，在Attention Mechanism中计算GRU的update gate zt（代表有多少ht-1和t对当前的隐藏层状态带来影响）时，只考虑了来自上个时刻隐藏层和当前输入的影响，没有考虑问题和之前memory可能带来的影响。最后，在Memory Update Mechanism中，之前是根据输入模块的输出ct，上一时刻的隐藏层状态ht-1作为GRU的输入更新当前时刻隐藏层状态。这就意味着每次迭代隐藏层状态的权重是一样的，这样的模型被称为tied model，表现能力受限。

对于Input Module的改进：

首先引入一个叫做位置编码（positional encoding）的结构，把之前从单词层面的编码转换成句子层面的编码。因为问题总是跟句子相关，在句子层面上的建模显得更加合理。相当于用positional encoding代替之前的GRU结构，一方面降低了计算量，另一方面又捕获了单词位置的关系。具体的做法是对句子中每个单词的词向量根据其所在的位置乘以一个权重，最后把所有的词向量相加形成的Tensor代表对整个句子的信息归纳。positional encoding的计算规则（即词向量权重）如下：

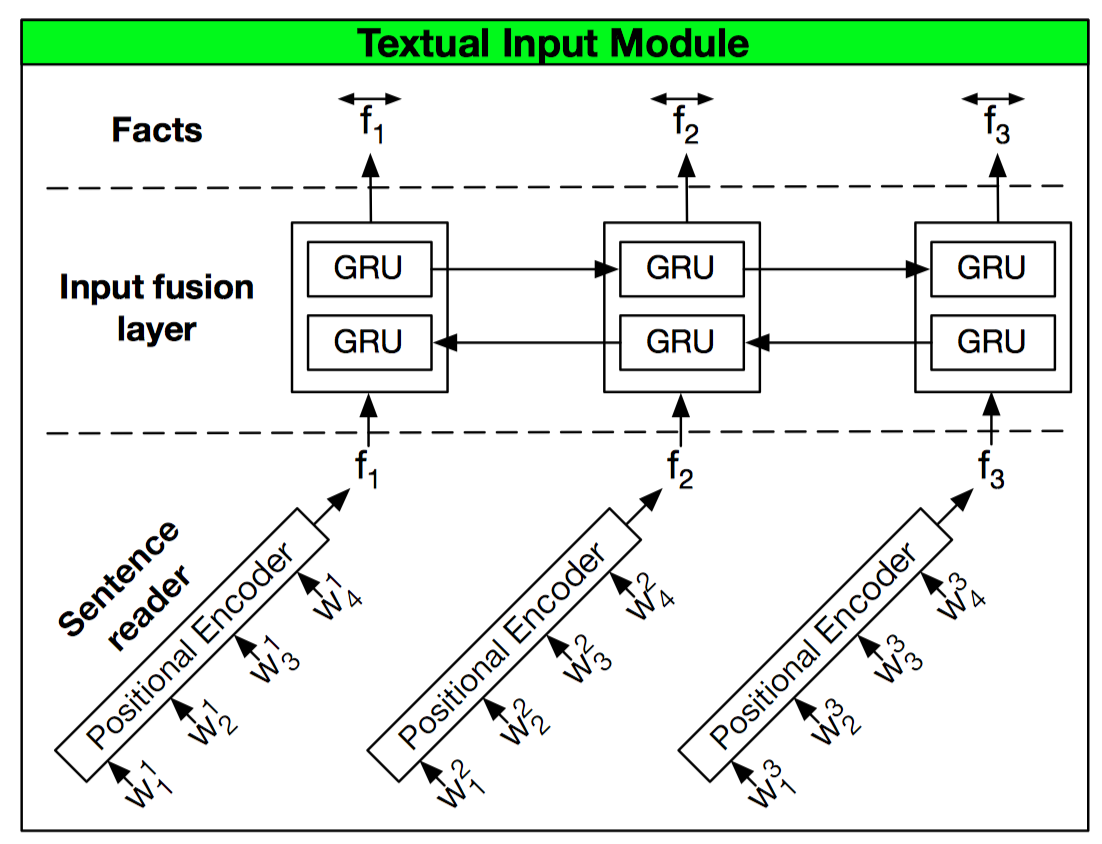


ljd为M\*D的二维列表，M表示一个问答对中的句子个数，D表示最长的句子中单词个数。此外j为句子的索引，d为句子中单词的索引。随后将该二维列表和词向量点乘，并对每个句子中的词向量求和作为positional encoding结构的输出。



得到的Tensor维度应为 句子个数\*词向量维度。

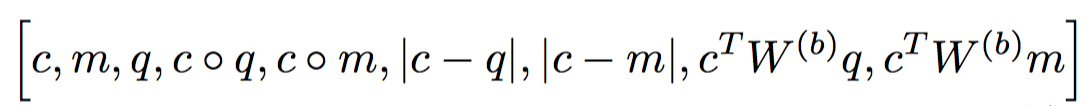
在此之后，引入Input Fusion Layer使句子之间的信息能够交互，本质上是双向GRU。



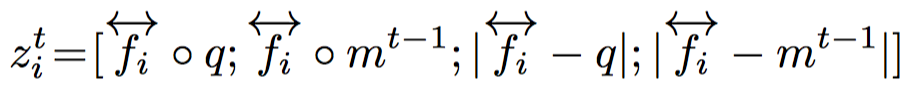
Input Fusion Layer实际做的是把一个context中的多个句子依次进入双向GRU，把每个时刻的隐藏层状态拼接作为输出。保证输入和输出的维度未发生改变。

对Attention Mechanism的改进：

首先我们改变z函数，之前的z函数由如下六部分组成，

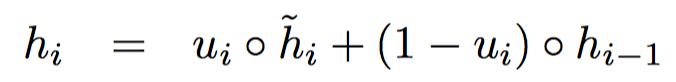


但是实验发现，只需保留如下四部分，



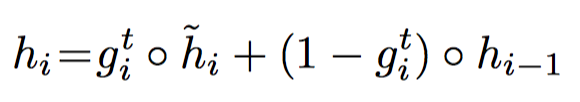
能够在降低计算量的基础上得到近似的准确率。

其次，改变Attention Mechanism中GRU的结构，得到名为Attention Based GRU结构。GRU中计算当前隐藏层状态的公式如下：



可见ht-1是update gate ui , 上一时刻的隐藏层状态 hi-1 和代表对当前输入和上一个时刻隐藏状态的总结i 的函数。

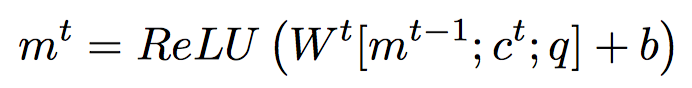
Attention Based GRU中计算当前隐藏层状态的公式如下：



唯一的区别是用attention gate gti替换update gate ui，gti为Attention Mechanism的输出，代表每个句子的权重（即重要程度），其中还包含了question的信息。

对Memory Update Mechanism的改进：

用RELU函数代替GRU对Memory进行更新：



其中的Memory mt在GRU结构中就是t时刻的隐藏层状态。c和q为输入和问题模块的输出。通过这样的改变，能带来准确率0.5%的提升。

模型更新2（TreeLSTM-DMN+）

此次的更新在是对Input Module的Positional Encoding。我们使用TreeLSTM代替Positional Encoding。原因是Positional Encoding虽然包含了单词位置信息，但是TreeLSTM能够保留单词间的依赖关系。理论上，最后隐藏层中所包含的信息更多。

首先介绍TreeLSTM结构，TreeLSTM是基于树的结构使用LSTM，从叶子结点开始不断迭代，最后把根结点的隐藏状态作为整个句子的总结。如果我们把TreeLSTM定义为一个类，那么其执行函数的输入为代表某个句子的树对象和代表每个单词的词向量形式Tensor. 可见我们需要得到代表句子的树结构。在这里使用的是 The Stanford Parser, 介绍和下载路径如下：

<https://nlp.stanford.edu/software/lex-parser.shtml>

它能够把一个句子解析为Dependency Tree（依存树） 或者 constituency Tree （组成树）。对于依赖树而言，每个树结点是一个单词，根据本文一开始提到的论文Dependency Parser ，它依靠神经网络找到单词和单词之前的依赖关系，每个结点的孩子必定是和该结点单词联系最紧密的几个单词，最后联系强的单词作为一个词组形成一个子树。举个很简单的例子 I eat food. 最后的依存树根结点为eat，I 和 food 分别为eat 的两个孩子。最终用这棵树代表整个句子，需要注意的是，每个结点的孩子个数是不固定的。而对于组成树而言，其每个叶子节点为句子中的单词，因此叶子结点的个数即为句子中单词的个数。每个非叶子结点代表多个单词组成的词组，最终的结果一个是课二叉树。还是用上面的例子，最终的树结构是：I eat food .这个词组代表根结点（称为结点0），其左孩子是I eat food （称为结点1）右孩子是 . （称为结点2）。对于结点1而言，其左孩子为 I （称为结点3），右孩子为 eat food （称为结点4）。而结点4的两个孩子分别为 eat 和 food。在这里，我们使用依存树作为TreeLSTM的树结构。

此时TreeLSTM的自我迭代不再依赖句子中单词所在的位置，当前时刻的迭代是以其孩子结点的输出作为输入。那么最后root不仅仅包含了句子中每个单词的信息，还能够捕获单词之前的结构关系信息。

通过最后实验证明，在多数任务中，该模型的测试集准确率要优于上个模型的测试集准确率。但在个别任务上（task3, task16），该模型的训练训练准确率不如上一个模型。

此时的另一个问题是训练速度慢，因为对每个句子使用TreeLSTM结构迭代的时候，每一次只能对一个句子迭代，没有实现batch size。将在下一个模型的更新实现。

模型更新3（Batch Based TreeLSTM-DMN+ ）

该部分主要由陈俏均完成。

在TreeLSTM-DMN+的基础上引入批处理技术，使得训练速度加快，但是理论上不会改变模型在测试集上的准确率。我们根据其contexts中句子的数量分批，使得每个批次中contexts的大小保持基本一致。每次取该批次下的batch size个contexts进入TreeLSTM。

由于每个context中每句话的树结构可能不同，所以无法直接对输入TreeLSTM模块的数据直接升维（在现有维度上增加batch size维度）。因此，我们的想法是先记录下一个批次中所有的树结点信息。信息包括结点类型是孩子结点还是叶节点，以及该结点的深度，孩子结点的个数，结点编号。最终进入TreeLSTM的结点必定是同一深度，并且拥有相同孩子树的孩子结点，把这个最为实现批处理的最大批次。

**结果**：

和DMN+模型相比，那些在DMN+中取得100%准确率的任务，在TreeLSTM\_DMN+中同样能取得100%的准确率，在其余任务中，除了在task3和task16表现弱于DMN+，其余任务的准确率都超过了DMN+模型的表现。而在task3和task16中表现较差的原因很有可能是input module的依存树出现过拟合现象，因为实现中发现，如果控制树的复杂度，能带来准确率上的提高，但是仍然弱于DMN+的表现。