Bert 网络详细结构

Bert里面只使用了Transformer的Encoder结构，由于**ULMFiT** (18年)的提出，Transformer和Transfer learning结合起来使用变得流行，图像领域有 OpenAI的 **GPT**和Google AI的 **BERT**.

The Encoder

Notations

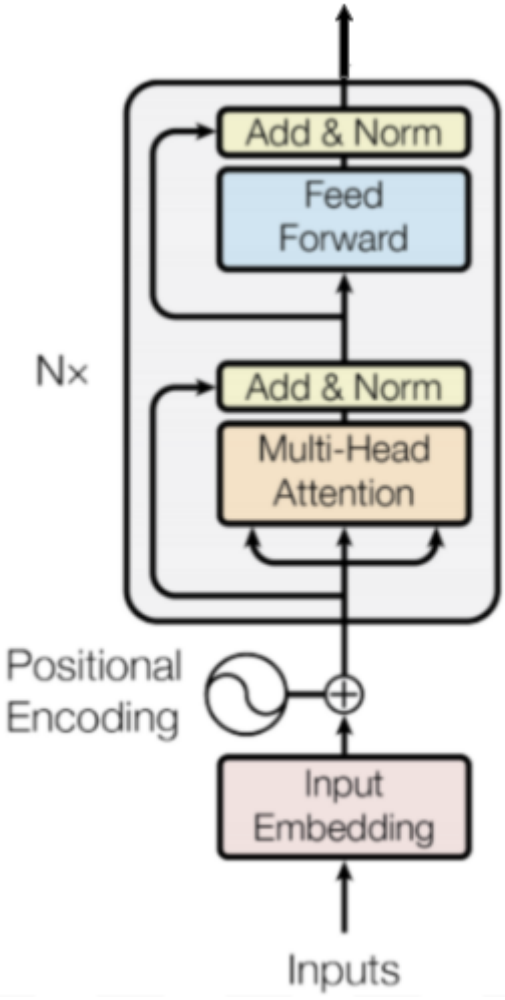
emb_dim: token embedding的维度

input_length:输入长度固定，有padding

hidden_dim: 隐藏层尺寸

vocab_size: 词典长度

DataFlow



如上图，inputs的长度为input_length,1）它首先被编码成tokens embeddings，得到input embedding，2）然后input embedding和position encoding相加，3）相加的结果一起输入到N个encoder block中，如上面block所示，有几点值得注意的是：

- N个encoder的权重不共享

- 每个encoder的输入和输出维度是一样的
- 在Bert实验中，N的选择12和24

下面来分别细讲这三个步骤：

Words to Input Embeddings

主要经历这样三个步骤：Tokenization, numericalization and word embeddings

分词

在这里，Token为文本处理的最小单元，Tokenization在Bert中分为BasicTokenizer和WordpieceTokenizer，FullTokenizer将两者结合使用，在训练和测试都要使用，因为输入的是最原始的自然语言。具体实现可参考官方代码tokenlization。

首先Bert的输入可以是1个或者两个sentence，用[CLS]和[SEP]分别标识sentence的开始和结束：

> [CLS] the man went to the store [SEP] he bought a gallon of milk [SEP]
>
> or
>
> [CLS] the man went to the store [SEP]

代码中的Segment ID就是用来指定这个token是属于哪个sentence的。

需要说一下的是WordpieceTokenizer,他将一个单词拆分成一个个的sub-word。它利用BPE（Byte-Pair Encoding）双字节编码来实现，BPE的过程可以理解为把一个单词再拆分，使得我们的此表会变得精简，并且寓意更加清晰。举个例子，"loved","loving","loves"这三个单词。其实本身的语义都是“爱”的意思，但是如果我们以单词为单位，那它们就算不一样的词，在英语中不同后缀的词非常的多，就会使得词表变的很大，训练速度变慢，训练的效果也不是太好。BPE算法通过训练，能够把上面的3个单词拆分成"lov","ed","ing","es"几部分，这样可以把词的本身的意思和时态分开，有效的减少了词表的数量。最终，Bert的词表分为以下几种类型的tokens：

1. 完整的常见单词
2. 出现在一个单词前面的子单词 (“em” as in “embeddings” is assigned the same vector as the standalone sequence of characters “em” as in “go get em”)
3. 不出现在开始部分的子单词，用##来标识，表示这个子单词前面还有其他(子)单词
4. 英语字符

建立好词表之后，bert进行分词的时候经过以下步骤；

> o tokenize a word under this model, the tokenizer first checks if the whole word is in the vocabulary. If not, it tries to break the word into the largest possible subwords contained in the vocabulary, and as a last resort will decompose the word into individual characters. Note that because of this, we can always represent a word as, at the very least, the collection of its individual characters.
>
> As a result, rather than assigning out of vocabulary words to a catch-all token like ‘OOV’ or ‘UNK,’ words that are not in the vocabulary are decomposed into subword and character tokens that we can then generate embeddings for.
>
> So, rather than assigning “embeddings” and every other out of vocabulary word to an overloaded unknown vocabulary token, we split it into subword tokens [‘em’, ‘##bed’, ‘##ding’, ‘##s’] that will retain some of the contextual meaning of the original word. We can even average these subword embedding vectors to generate an approximate vector for the original word.

详细内容可以参考wordpiece model的原始论文，以及Google的Neural Machine Translation System中的介绍。

numericalization

把每个token映射到一个单独的整数

word embeddings

可以理解为对每个token在一个embedding dicrt中查询一个向量表达，这个embedding dict可以是已经学好的，也可以是将要学习的。

如果一个batch里面sequence length是不一样长的，那么就需要padding。

Position Encoding

词向量的矩阵没有编码单词的相对位置关系，在这里就是采用了sinuosidal functions来编码单词的相对位置关系。在原始的矩阵上加了位置编码矩阵P：

$$
\begin{array}{c}
 & < & - & d_{emb\_dim} & - & > \\
Hello & \sin\left(\frac{0}{10000^{\frac{0}{emb_{dim}}}}\right) & \cos\left(\frac{0}{10000^{\frac{0}{emb_{dim}}}}\right) & \sin\left(\frac{0}{10000^{\frac{2}{emb_{dim}}}}\right) & \cos\left(\frac{0}{10000^{\frac{2}{emb_{dim}}}}\right) & \cdots \\
, & \sin\left(\frac{1}{10000^{\frac{0}{emb_{dim}}}}\right) & \cos\left(\frac{1}{10000^{\frac{0}{emb_{dim}}}}\right) & \sin\left(\frac{1}{10000^{\frac{2}{emb_{dim}}}}\right) & \cos\left(\frac{1}{10000^{\frac{2}{emb_{dim}}}}\right) & \cdots \\
how & \sin\left(\frac{2}{10000^{\frac{0}{emb_{dim}}}}\right) & \cos\left(\frac{2}{10000^{\frac{0}{emb_{dim}}}}\right) & \sin\left(\frac{2}{10000^{\frac{2}{emb_{dim}}}}\right) & \cos\left(\frac{2}{10000^{\frac{2}{emb_{dim}}}}\right) & \cdots \\
are & \sin\left(\frac{3}{10000^{\frac{0}{emb_{dim}}}}\right) & \cos\left(\frac{3}{10000^{\frac{0}{emb_{dim}}}}\right) & \sin\left(\frac{3}{10000^{\frac{2}{emb_{dim}}}}\right) & \cos\left(\frac{3}{10000^{\frac{2}{emb_{dim}}}}\right) & \cdots \\
you & \sin\left(\frac{4}{10000^{\frac{0}{emb_{dim}}}}\right) & \cos\left(\frac{4}{10000^{\frac{0}{emb_{dim}}}}\right) & \sin\left(\frac{4}{10000^{\frac{2}{emb_{dim}}}}\right) & \cos\left(\frac{4}{10000^{\frac{2}{emb_{dim}}}}\right) & \cdots \\
? & \sin\left(\frac{5}{10000^{\frac{0}{d_{emb_{dim}}}}}\right) & \cos\left(\frac{5}{10000^{\frac{0}{emb_{dim}}}}\right) & \sin\left(\frac{5}{10000^{\frac{2}{emb_{dim}}}}\right) & \cos\left(\frac{5}{10000^{\frac{2}{emb_{dim}}}}\right) & \cdots
\end{array}
$$

原始矩阵为Z：

$$
\begin{array}{c}
 & < & - & d_{emb\_dim} & - & > \\
Hello & 123.4 & 0.32 & \cdots & 94 & 32 \\
, & 83 & 34 & \cdots & 77 & 19 \\
how & 0.2 & 50 & \cdots & 33 & 30 \\
are & 289 & 432.98 & \cdots & 150 & 92 \\
you & 80 & 46 & \cdots & 23 & 32 \\
? & 41 & 21 & \cdots & 74 & 33
\end{array}
$$

结果矩阵为：X = Z + P

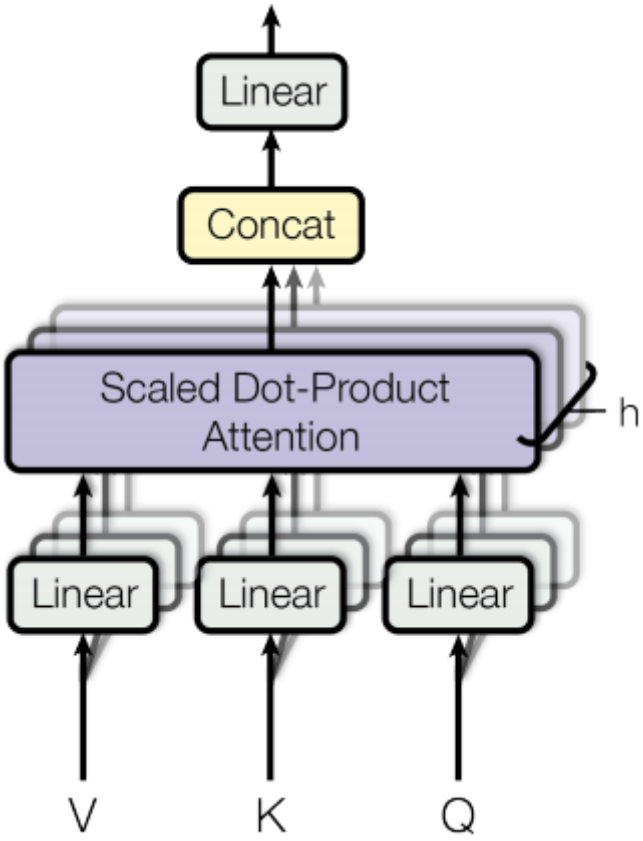
Encoder block

有N个block，每个block的作用主要是编码输入的关系(包含单词、句子之间简单和复杂的关系)。

Multi-Head Attention

> *Multi-Head Attention*, which means it computes attention *h* different times with different weight matrices and then concatenates the results together.

这些并行计算的attention被称为head，下图为head的计算和结合：



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Q,K,V为不同的输入矩阵，在这里的话，都将被X代替。

每个头都将和三个矩阵相乘，经过以下公式，我们得到$head_i$的表达：

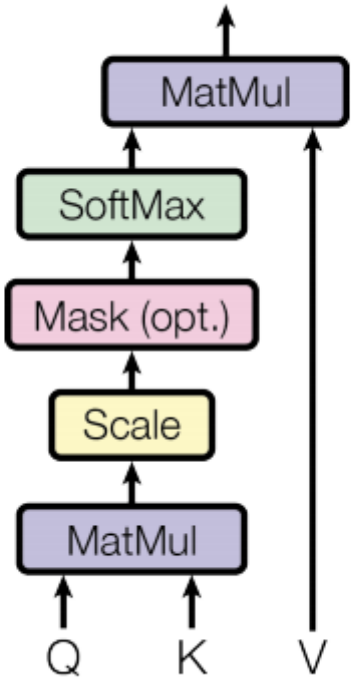
$$XW_i^K = K_i \text{ with dimensons } input\_length \text{ x } d_k$$
$$XW_i^Q = Q_i \text{ with dimensons } input\_length \text{ x } d_k$$
$$XW_i^V = V_i \text{ with dimensons } input\_length \text{ x } d_v$$

Scaled Dot-Product Attention

接下来计算 *Scaled Dot-Product Attention*:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

这一步从图形上来解释：



Position-wise Feed-Forward Network

这一步的计算为：

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Reference

https://medium.com/dissecting-bert/dissecting-bert-part-1-d3c3d495cdb3

https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/

https://www.cnblogs.com/huangyc/p/10223075.html#_label1