词向量经典模型：从word2vec、glove、ELMo到BERT

xiayto

前言

词向量技术是将自然语言中的词转化为稠密的向量，语义相似的词会有相似的向量表示。生成词向量的方法从一开始基于统计学（共现矩阵、SVD分解）到基于神经网络的语言模型，这里总结一下比较经典的语言模型：word2vec、glove、ELMo、BERT。

其中BERT是Google2018年发表的模型，在11个经典的NLP任务中测试效果超越之前的最佳模型，并且为下游任务设计了简单至极的接口，改变了以前花销的Attention、Stack等堆叠结构的玩法，应该属于NLP领域里程碑式的贡献。

word2vec

word2vec来源于2013年的论文《Efficient Estimation of Word Representation in Vector Space》，它的核心思想是通过词的上下文得到词的向量化表示，有两种方法：CBOW（通过附近词预测中心词）、Skip-gram（通过中心词预测附近的词）：

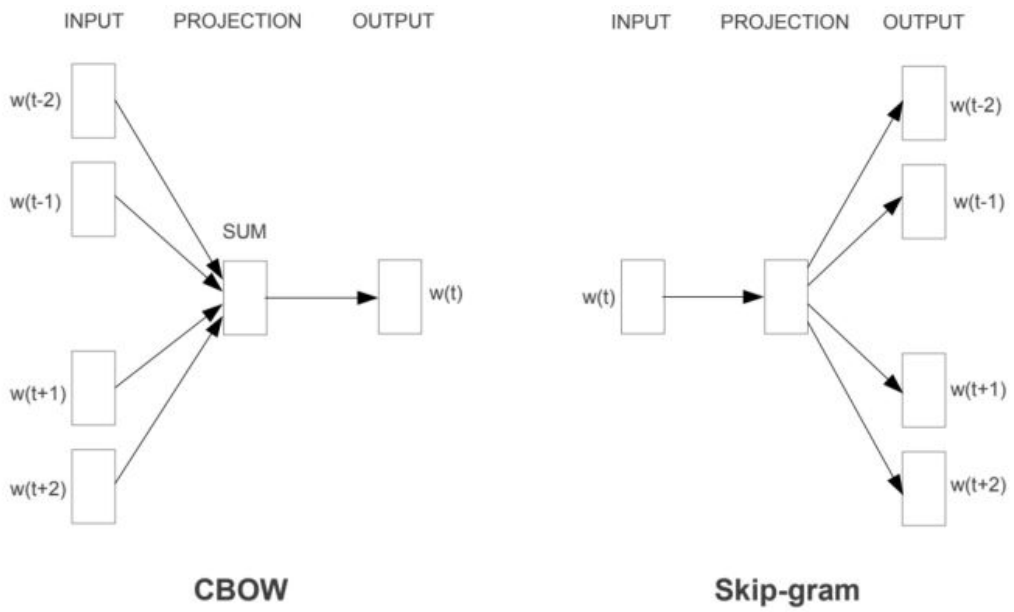


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

CBOW和Skip-gram

CBOW：

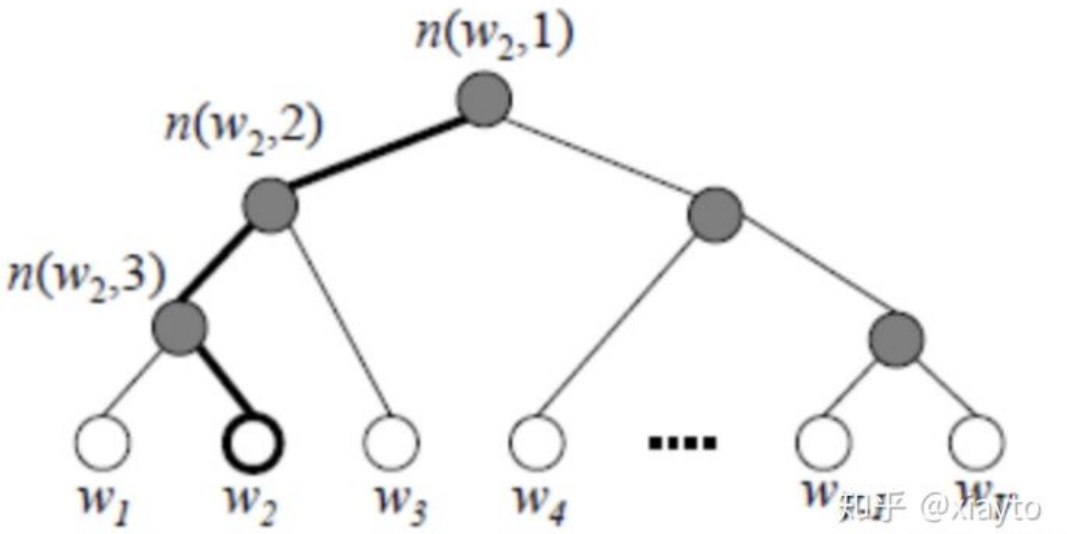
通过目标词的上下文的词预测目标词，图中就是取大小为2的窗口，通过目标词前后两个词预测目标词。具体的做法是，设定词向量的维度d，对所有的词随机初始化为一个d维的向量，然后要对上下文所有的词向量编码得到一个隐藏层的向量，通过这个隐藏层的向量预测目标词，CBOW中的做法是简单的相加，然后做一个softmax的分类，例如词汇表中一个有V个不同的词，就是隐藏层d维的向量乘以一个W矩阵（ $\mathbb{R}^{d \times V}$ ）转化为一个V维的向量，然后做一个softmax的分类。由于V词汇的数量一般是很大的，每次训练都要更新整个W矩阵计算量会很大，同时这是一个样本不均衡

的问题，不同的词的出现次数会有很大的差异，所以论文中采用了两种不同的优化方法多层Softmax和负采样。

Skip-gram：

跟CBOW的原理相似，它的输入是目标词，先是将目标词映射为一个隐藏层向量，根据这个向量预测目标词上下文两个词，因为词汇表大和样本不均衡，同样也会采用多层softmax或负采样优化。

多层softmax：



哈夫曼编码

由于单词出现的频率是不一样的，所以用哈夫曼编码构建一个二叉树，出现频率高的词放在前面可以减少计算量，用哈夫曼编码记录路径，例如图中单词2（ w_2 ，规定左边走是1，右边走是0）的哈夫曼编码就是1110，在路径上每到一个节点就是一个sigmoid的二分类，所以叫多层softmax。具体来说每个节点处会有一个d维的向量参数 θ ，每个单词也是d维的向量 x_w ，一个sigmoid的方程决定向左或者右走的概率，往左走的概率是：

$$P(+) = \sigma(x_w^T\theta) = \frac{1}{1+e^{-x_w^T\theta}}$$

将路径上的概率连乘，极大化这个概率值，就是模型的训练目标，每次用一个样本更新梯度，使用对数似然函数：

$$L = \sum^{l_w} j = 2(1-d_j^w)log[sigma(x_w^T\theta^w j-1)] + d_j^w log[1-sigma(x_w^T\theta_{j-1}^w)]$$

其中 l_w 代表目标词在哈夫曼树中的深度， d_j^w 代表在j层中目标词根据哈夫曼编码的路径应该向左还是向右走，左是1右是0，可以看到整个目标函数跟逻辑回归是基本相似的，梯度的计算也跟逻辑回归基本一样。 对于CBOW，多层softmax是将窗口内的2c个词求平均得到 x_w ，放入哈夫曼树的网络中，每一层得到的梯度更新哈夫曼树中的参数，同时梯度同步更新到2c个单词的向量中。对于Skip-gram，多层softmax是用目标词的词向量为 x_w ，然后遍历窗口内的2c个词，一个进行2c次的训练，每次将 x_w 放进哈夫曼树中，更新 x_w 和树中的参数。

负采样：

负采样也是把模型变为二元的逻辑回归，每个词对应一个词向量 x_w 和词的网络参数 θ_w ，每个词附近有2c个词，通过负采样，随机采样neg个反例，让正例词对目标词的词向量和窗口内词的网络参数的乘积尽可能的大，反例的尽可能小，是模型的训练目标。例如在skip-gram中目标词A的词向量是 x_A ，它2c的窗口内附近一个词的网络参数是 θ_B ，让这两个的乘积尽可能的大，$\sigma(x_A^T\theta_B)$ 尽可能大，反例，随机采样出来的词乘积尽可能小。同理CBOW中是将2c个词相加求平均，然后与目标词的网络参数相乘，也是让窗口内真实存在的词乘积尽可能大，随机采样出的词尽可能小。 负采样的方式是根据词出现的频率进行采样，出现频率越高，越可能被采样到，原文中是根据出现频率的3/4次方然后做一个归一化作为概率进行采样。

另外这篇论文的作者在2016年开源了一个文本分类的工具，FastText，与word2vec的原理非常类似，只不过在做文本分类时是一个有监督的分类预测文本的label。另外加入n-gram的特征，对于英文，每个单词前后加入标识，例如#good#，如果是3-gram的特征就是加入#go,goo,ood,od#，可以解决一些oov的问题（不在训练集中出现的生词）和引入一些词根的特征。对于中文“你爱我”和“我爱你”有完全不同的语义，用CNN或者highway网络处理的char embedding可以引入这部分的特征。

glove

word2vec只考虑到了词的局部信息，没有考虑到词与局部窗口外词的联系，glove利用共现矩阵，同时考虑了局部信息和整体的信息。来自论文《Glove: Global vectors for word representation》。 不知道为什么大家都用这个例子：i love you but you love him i am sad。。。可能程序员的现实就是这么残酷。。。 用这个例子介绍窗口，其实跟word2vec是一样的。

窗口标号	中心词	窗口内容
0	i	i love you
1	love	i love you but
2	you	i love you but you
3	but	love you but you love
4	you	you but you love him
5	love	but you love him i
6	him	you love him i am
7	i	love him i am sad
8	am	him i am sad
9	sad	i am sad

作者定义了几个符号：

1、 $X_{i,j}$ ：

它是通过遍历所有的窗口内容统计得到的， $X_{i,j}$ 表示以i为中心词，j出现在窗口内容中的次数。如上图所示，例如love的窗口内容是but、you、him、i，那么
$X_{love,but}$ ， $X_{love,you}$ ， $X_{love,him}$ ， $X_{love,i}$ 都加1。

2、 X_i ：

$$X_i = \sum_{j=1}^{N} X_{i,j}$$

也就是统计词i一共是多少个其他词的中心词。

3、 $P_{i,k}$ ：

$$P_{i,k} = \frac{X_{i,k}}{X_i}$$

也就是词k在词i附近占词i所有的附近词的比例。

4、 $Ratio_{i,j,k}$ ：

$$Ratio_{i,j,k} = \frac{P_{i,k}}{P_{j,k}}$$

也就是词k出现在词i和出现在词j附近的比例的比例。。。 然后作者发现一个规律：

$ratio_{i,j,k}$的值	单词j, k相关	单词j, k不相关
单词i, k相关	趋近1	很大
单词i, k不相关	很小	趋近1

然后作者就设计目标函数去拟合这个规律，这个ratio跟i,j,k有关，所以将词i,j,k的向量放进函数中，代价函数的模版应该是：

$$J = \sum_{i,j,k}^{N} (\frac{P_{i,k}}{P_j, k} - g(v_i, v_j, v_k))$$

重点就在于如何设计g(*)这个函数，首先，它是比较 v_i, v_j 在 v_k 附近的比例的差异，因为比较差异所以应该有一项减法 $(v_i - v_j)$ ，然后是在 v_k 附近时候它们的差异，所以应该跟 v_k 有关，同时ratio是一个标量，要将差异向量转化为一个标量所以变为 $(v_i - v_j)^T v_k$ ，然后要做一个变换使它符合ratio的特点，当 v_i, v_j 在 v_k 附近出现比例的比例是相似时候，ratio应该约等于1，也就是 $(v_i - v_j)$ 约等于0的时候， $g(v_i, v_j, v_k)$ 应该约等于1，同时应该满足 $g(v_i, v_j, v_k)$ 跟 v_i (对应 $P_{i,k}$)的关系是单调递增，跟 v_j (对应 $P_{i,j}$)的关系是单调递减的，这时候exp()函数符合这个规律，所以最后的损失函数变为：

$$\frac{P_{i,k}}{P_j, k} = exp((v_i - v_j)^T v_k)$$

经过一些数学的变换变为：

$$\frac{P_{i,k}}{P_j, k} = \frac{exp(v_i^T v_k)}{exp(v_j^T v_k)}$$

这里如果整个损失函数更新时间复杂度是 $O(n^3)$ ，如果等式两边对应分子等于分子，分母等于分母可以让复杂度将为 $O(n^2)$ 。两边取对数得到：

$$P_{i,j} = exp(v_i^T v_j)$$

但是存在一个问题， $P_{i,j}$ 和 $P_{j,i}$ 是不等的，但是 $exp(v_i^T v_j)$ 和 $exp(v_j^T v_i)$ 却是相等的，留意到 $X_{i,j}$ 和 $X_{j,i}$ 是相等的，可以把 $P_{i,j}$ 展开，等式写为：

$$log(X_{i,j}) - log(X_i) = v_i^T v_j$$

加入偏置项，并把 $log(X_i)$ 并入，最后代价方程变为：

$$J = \sum_{i,j}^{N} (v_i^T v_j + b_i + b_j - log(X_{i,j}))^2$$

最小化代价方程得到词的向量表示。

ELMo

ELMo来自于论文《Deep contextualized word representations》，它的官网有开源的工具：allennlp.org/elmo

word2vec和glove存在一个问题，词在不同的语境下其实有不同的含义，而这两个模型词在不同语境下的向量表示是相同的，Elmo就是针对这一点进行了优化，作者认为ELMo有两个优势：

1、能够学习到单词用法的复杂特性
2、学习到这些复杂用法在不同上下文的变化

针对点1，作者是通过多层的stack LSTM去学习词的复杂用法，论文中的实验验证了作者的想法，不同层的output可以获得不同层次的词法特征。文中5.5章节有不同任务的情况对比，对于词义消歧有需求的任务，第2层会有较大的权重，对于对词性、句法有需求的任务对第1层会有比较大的权重。

针对点2，作者通过pre-train+fine tuning的方式实现，先在大语料库上进行pre-train，再在下游任务的语料库上进行fine tuning。

具体实现方式：

ELMo来自于Embeddings from Language Models的简写，这里的language model作者采用了LSTM。一开始的训练目标可以表示为如下的形式：

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_1, t_2, \ldots, t_{k-1}).$$

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_{k+1}, t_{k+2}, \ldots, t_N).$$

具体来说是通过双方向预测单词，前向过程中，用1～k-1的词去预测第k个词，后向过程中，用k+1～N的词去预测第k个词。具体的编码方式作者采用了LSTM，可以表示为：

$$\sum_{k=1}^{N} (\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s))$$

通过LSTM编码Tokens，用第k-1个Token的隐藏层输出预测第k个Token，预测的方法是用一个softmax做一个分类，其中 Θ_x 是token的向量表示， Θ_s 是softmax的参数。这种双向LSTM堆

叠L层，stack-RNN的方式也就是每一层隐藏层的输出作为下一层的输入。 那么对于每个Token会有2L+1个向量表示，双向所以每层2个，+1是token输出层的向量表示，作者将2L+1个向量组合起来得到ELMo每个token的向量表示：

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}.$$

其中 $h_{k,j}^{LM}$ 是对于token k的第j层的隐藏层输出，当j=0的时候表示输入层token的向量表示，这个向量表示可以通过CNN或Highway等网络引入char-level的特征。 s_j^{task} 是一个softmax的归一化， γ^{task} 是一个放缩的参数，可以让目标模型对ELMo的向量进行放缩。 经过两层双向LSTM提取特征，又针对下游任务定制词向量，效果比word2vec和glove要好也在情理之中。

BERT

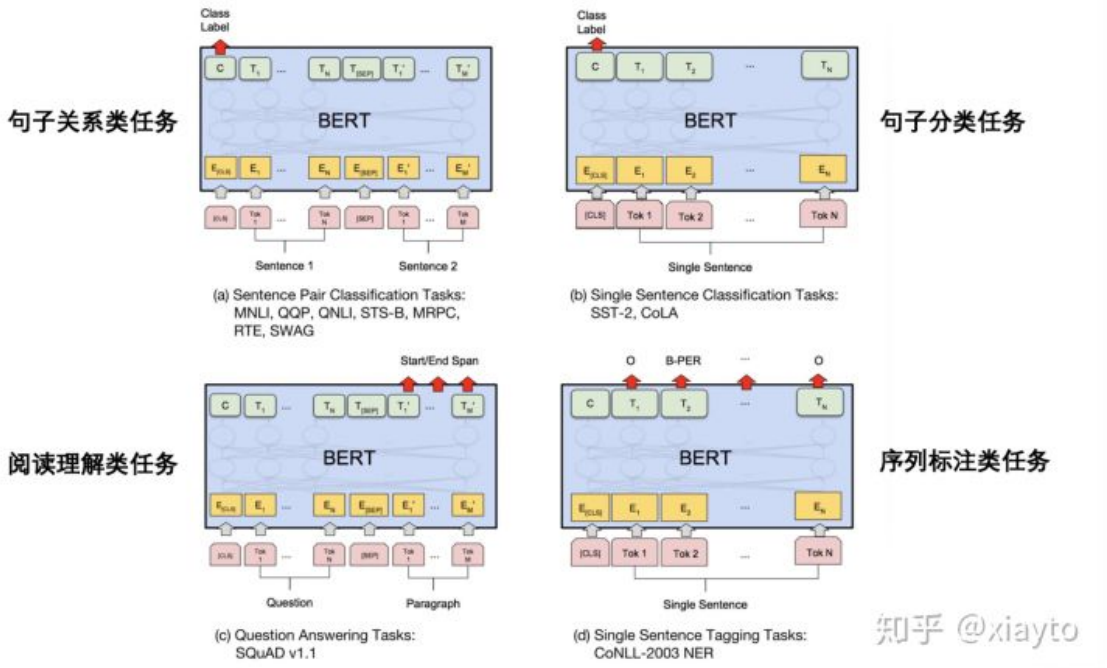
这里重点介绍一下BERT，它来自于googole发表的论文《BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding》，这个模型彻底的改变了NLP的游戏规则。

改变游戏规则：

NLP一共有4大类的任务：

序列标注：分词 / 词性标注 / 命名实体识别...
分类任务：文本分类 / 情感分析...
句子关系判断：自然语言推理 / 深度文本匹配 / 问答系统...
生成式任务：机器翻译 / 文本摘要生成...

BERT为这4大类任务的前3个都设计了简单至极的下游接口，省去了各种花哨的Attention、stack等复杂的网络结构，且实验效果全面取得了大幅度的提升。



BERT下游任务接口

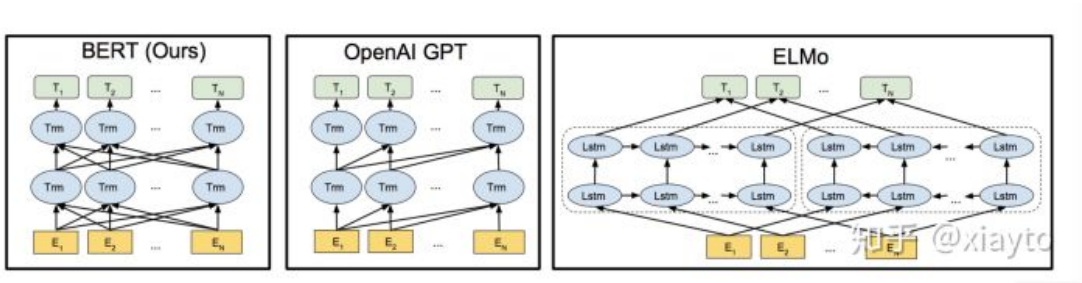
上图是BERT论文中为下游任务设计的接口，可以先将BERT看作是一个黑盒，它跟ELMo的工作方式是类似的，都是现在大规模的语料库中pre-train，然后将下游任务输入，进行比较轻量级的fine-tuning。下游任务只要做一些轻微的改造就可以放入BERT的模型中fine-tuning，可以看到对于句子关系类的任务，只要加上句子起始和结束的符号，句子之间加入分割符号，然后经过BERT模型它最后的一个位置的输出连接上一个softmax的分类器就可以了。 对于序列标注的模型，也是只要加入起始和结束的符号，对于最后BERT每个位置的输出都加入一个线性的分类器就可以了。

BERT的诞生过程：

BERT的工作方式跟ELMo是类似的，但是ELMo存在一个问题，它的语言模型使用的是LSTM，而不是google在2017最新推出的Transformer（来自论文《Attention is all you need》）。LSTM这类序列模型最主要的问题有两个：

一是它单方向的，即使是BiLSTM双向模型，也只是在loss处做一个简单的相加，也就是说它是按顺序做推理的，没办法考虑另一个方向的数据。
二是它是序列模型，要等前一步计算结束才可以计算下一步，并行计算的能力很差。

所以在ELMo之后，一个新模型GPT（来自论文《Improving Language Understanding by Generative Pre-Training》）推出了，它用Transformer来编码。但是它的推理方式跟ELMo相似，用前面的词去预测下一个词，所以它是单方向，损失掉了下文的信息。 然后BERT诞生了，它采用了Transformer进行编码，预测词的时候双向综合的考虑上下文特征。这里作者做了一个改进，原来没办法综合利用双向的特征是因为目标是用前面的词逐个的预测下一个词，如果你能看到另一个方向，那就等于偷看了答案。BERT的作者说受到了完形填空的启发，遮盖住文章中15%的词，用剩下85%的词预测这15%的词，那就可以放心的利用双向上下文的特征了。

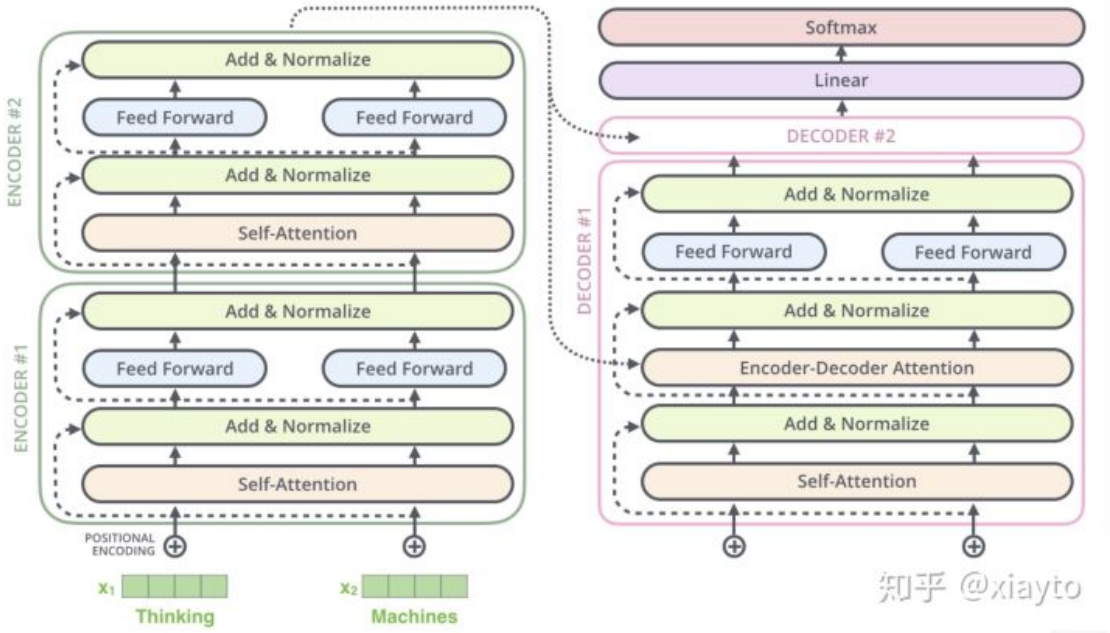


BERT、GPT、ELMo结构比较

Transformer：

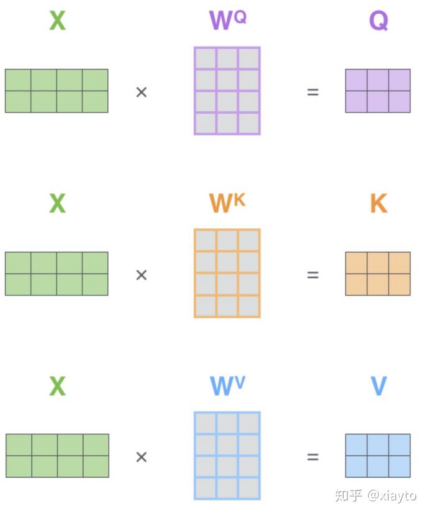
Transformer是论文《Attention is all you need》中的模型，它把attention机制从配角搬上了主角的位置，没有采用CNN或者RNN的结构，完全使用attention进行编码。Transformer应该会取代CNN和RNN成为NLP主流的编码方式，CNN提取的是局部特征，但是对于文本数据，忽略了长距离的依赖，CNN在文本中的编码能力弱于RNN，而RNN是序列模型，并行能力差，计算缓慢，而且只能考虑一个方向上的信息。Transformer可以综合的考虑两个方向的信息，而且有非常好的并行性质，简单介绍一下Transformer的结构。

jalammar.github.io/illu 这一篇博客上有非常详细的介绍，详细了解Transformer可以看一下这一个博客。



Transformer的整体结构

整个Transformer的结构如上图所示，它分为两个部分，一边是encoder，另一边是decoder，一个encoder或decoder看作一个block，encoder和decoder又有多个block串行相连，图中是2个，原文中使用了6个。 先看encoder，单独看一个block，输入一行句子，先经过一个Self-Attention，Self-Attention首先是对每个Token的embedding通过3个不同的矩阵映射成3个向量，Query(Q)、Key(K)和Value(V)。



Self-Attention中的Q、K、V向量计算方式

上图是假设句子的长度只有2个tokens，由于Q、K、V的三个W转换矩阵在不同的位置是共享参数的，所以可以使用矩阵运算。 然后Q和K向量是用于计算Attention的权重参数的，计算出的权重参数乘到V向量上，再加权求和就得到每个token的向量。

计算的过程是这样的：例如要计算位置1的向量，首先Q1和K1到Kn点乘，得到n个标量，这个标量除以 $\sqrt{d_k}$ ， d_k 是K的向量的维度，然后做一个softmax的归一化，然后归一化后的权重对应乘到V1到Vn上，求和得到位置1的Token的向量。

这个Self-Attention还会并行的叠加H个，就是有H个相互独立的，Self-Attention的机制，每个位置的Token会有H个向量，将H个向量拼接再做一个线性的转换得到最后的向量，H称为Attention的头数，作者将这个机制命名为Multi-Head的Attention机制。

然后因为这种Self-Attention的结构会忽略掉位置的信息，不同于CNN和RNN自然的就会把位置信息编码进去，位置信息对于序列文本来说是非常重要的，作者在输出处加上位置的编码，再做一个标准化，对应图上的Add&Normalize。这里的位置编码作者采用了如下的方式：

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

每个pos的位置用一个d维的向量表示，这个向量的偶数位置用sin，奇数位置用cos计算，得到-1到1之间的值，之所以用三角函数是利用了三角函数和差变换可以线性变换的特性，因为BERT中没有采用这种计算方式，所以不作详细的介绍。 然后经过一个Feed Forward的神经网络再经过Add&Normalize完成一个encoder，传向下一个encoder。

Decoder的结构和运算机制与encoder基本一致，不同的是encoder顶层的K、V向量会传到Decoder的每个Block的Encoder-Decoder Attention部件中。最后经过一个线性的变换和Softmax分类器得到最后的结果。

BERT的实现方式：

Mask Language Model

受到完形填空的启发，它不同于传统的语言模型，它是盖住整篇文章15%的词，然后用其他的词预测这15%的词。被盖住的词用[mask]这样的一个标记代替，但是由于下游任务中没有[mask]这个符号，为了削弱这个符号的影响，15%被盖住的词中：

80%的词就用[mask]符号盖住
10%的词保留原来真实的词
10%的词用随机的一个词替代

编码方式：

采用Transformer为编码方式，实验结果最好的结构是BERT：L=24，H=1024，A=16。L=24也就是Transformer中提到的Block数，H=1024是隐藏层的维度，也就是Token经过矩阵映射之后K、Q、V向量的维度，A=16是Attention的头数，叠加了16层相互独立的Self-Attention。非常的google，money is all you need，如此复杂的结构，一般人是玩不了的，所幸的是google开源了pre-train的参数，只要用pre-train的参数对下游任务进行fine-tuning就可以使用google的BERT。

BERT的Transformer结构与Attention is all you need中有一个不同是对于位置信息的编码，论文中的Transformer是采用cos和sin函数计算，而BERT用了更为简单粗暴的方法，每一个位置赋予一个向量，例如句子的截断长度固定为50，那么0-49这50个位置各赋予一个向量，将这个向量加到self-attention的embedding上。

获取句间的关系：

目前为止只获得了Token级别的特征，但是对于一些句间关系的推理，对话系统、问答系统需要捕捉一些句子的特征。BERT采用给定2个句子，判断它们是否是连续的句子的方式捕捉句子级别的特征：

Input = [CLS] the man went to [MASK] store [SEP]

he bought a gallon [MASK] milk [SEP]

Label = IsNext

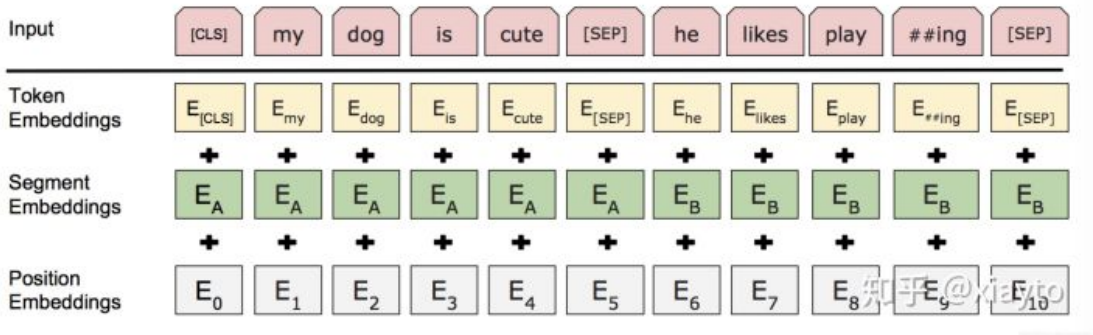
Input = [CLS] the man [MASK] to the store [SEP]

penguin [MASK] are flight ##less birds [SEP]

Label = NotNext

BERT捕捉句间关系，负采样二分类任务

具体的实现方式是两个连续的句子，开始和结束打上符号，两句之中打上分隔符，然后中一个二分类，反例的生成采用类似于word2vec的负采样。



BERT的Embedding整体结构

这是整体的编码方式，Token Embedding是Transformer的embedding，加上segment embedding A和B，捕捉句子级别的特征，区分第一句和第二句，如果只有一个句子的情况就只有embedding A，加上每个位置赋予的Position embedding得到最终的embedding。

BERT的实验结果：

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$BERT_{BASE}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$BERT_{LARGE}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

SQUAD

System	Dev EM	Dev F1	Test EM	Test F1
Leaderboard (Oct 8th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
#1 Single - nlnet	-	-	83.5	90.1
#2 Single - QANet	-	-	82.5	89.3
Published				
BiDAF+ELMo (Single)	-	85.8	-	-
R.M. Reader (Single)	78.9	86.3	79.5	86.6
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
$BERT_{BASE}$ (Single)	80.8	88.5	-	-
$BERT_{LARGE}$ (Single)	84.1	90.9	-	-
$BERT_{LARGE}$ (Ensemble)	85.8	91.8	-	-
$BERT_{LARGE}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$BERT_{LARGE}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

NER

System	Dev F1	Test F1
ELMo+BiLSTM+CRF	95.7	92.2
CVT+Multi (Clark et al., 2018)	-	92.6
$BERT_{BASE}$	96.4	92.4
$BERT_{LARGE}$	**96.6**	**92.8**

SWAG

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
$BERT_{BASE}$	81.6	-
$BERT_{LARGE}$	**86.6**	**86.3**
Human (expert)†	-	85.0
Human (5 annotations)†	-	88.0

两个有人类参与的指标中超越人类，11个不同的经典NLP任务超越SOTA的模型，里程碑式的改变，这很google。

参考文献：

1、word2vec：Efficient Estimation of Word Representation in Vector Space

2、Glove：Global vectors for word representation

3、ELMo：Deep contextualized word representations

4、Transformer：Attention is all you need

5、GPT：Improving Language Understanding by Generative Pre-Training

6、BERT： Pre-training of Deep Bidirectional Transformers for Language Understanding

编辑于 2018-12-10