链滴

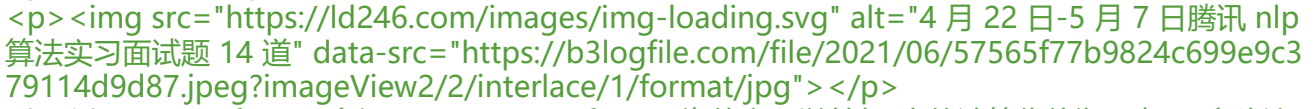
4 月 22 日 -5 月 7 日腾讯 nlp 算法实习面试题 14 道

作者：julyedu

原文链接：https://ld246.com/article/1623062353029

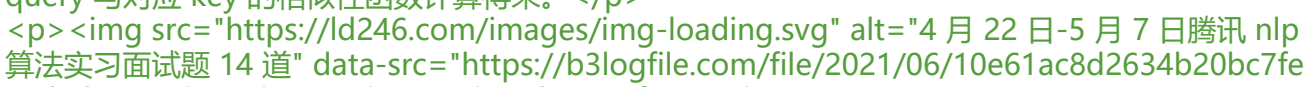
来源网站：链滴

<h2 id="1-介绍下word2vec-有哪两种实现-可以用什么方法提高性能-分层softmax介绍一下原理-采样怎么做--负采样和原始做法的优缺点比较-">1.介绍下 word2vec，有哪两种实现，可以用什么方提高性能，分层 softmax 介绍一下原理，负采样怎么做，，负采样和原始做法的优缺点比较。</h2>
参考答案：
在进行最优化的求解过程中，从隐藏层到输出的 softmax 层放入计算量很大，因为要计算所有的 softmax 概率，再去找概率最大的值，可以使用层次 softmax 和负采样两种方法；
层次 softmax 采用对输出层进行优化的策略，输出层从原始模型利用 softmax 计算概率值改为用 huffman 树进行计算概率值，其优点：由于是二叉树，之前计算量为 V，现在变为了 logV；由于用 Huffman 树是高频的词靠近树根，这样高频词需要更少的时间被找到，符合贪心优化思想；其缺：如果训练样本中的中心词 w 是一个很生僻的词，那么就需要在 Huffman 树中向下走很久。
负采样过程：比如我们有一个训练样本，中心词是 w,它周围上下文共有 2c 个词，记为 context()。由于这个中心词 w 和 context(w)相关存在，因此它是一个真实的正例。通过 Negative Sampling 采样，我们得到 neg 个和 w 不同的中心词 wi,i=1,2,...neg，这样 context(w)和 wi 就组成了 neg 个不真实存在的负例。利用这一个正例和 neg 个负例，我们进行二元逻辑回归，得到负采样对应每个词 wi 对应的模型参数 θi，和每个词的词向量。
负采样相比原始做法：每次只需要更新采样的词的权重，不需要更新所有词的权重，可以减少了练过程的计算负担，加速模型的计算，另一方面也可以保证模型训练的效果，提高了其结果词向量的量。
<h2 id="2-介绍LSTM和GRU有什么不同-">2.介绍 LSTM 和 GRU 有什么不同？</h2>
（1）LSTM 和 GRU 的性能在很多任务上不分伯仲；
（2）GRU 参数更少，因此更容易收敛，但是在大数据集的情况下，LSTM 性能表现更好；
（3）从结构上说，GRU 只有两个门，LSTM 有三个门，GRU 直接将 hidden
state 传给下一个单元，而 LSTM 则用 memory cell 把 hidden state 包装起来。
<h2 id="3-介绍一下LSTM单元里面的运算过程-">3.介绍一下 LSTM 单元里面的运算过程。</h2>


data-src="https://b3logfile.com/file/2021/06/c1947557846640c889cef02551d9d7b.jpeg?imageView2/2/interlace/1/format/jpg">
<h2 id="4-CRF的损失函数是什么-具体怎么算-">4.CRF 的损失函数是什么，具体怎么算？</h2>
在训练过程中，CRF 损失函数只需要两个分数：真实路径的分数和所有可能路径的总分数。所有可能路径的分数中，真实路径分数所占的比例会逐渐增加。
损失函数：


data-src="https://b3logfile.com/file/2021/06/57565f77b9824c699e9c379114d9d87.jpeg?imageView2/2/interlace/1/format/jpg">
<h2 id="5-transformer介绍一下原理-transformer为什么可以并行-它的计算代价瓶颈在哪-多头注力机制计算过程-">5.transformer 介绍一下原理，transformer 为什么可以并行，它的计算代价瓶颈哪？多头注意力机制计算过程？</h2>
Transformer 结构（原理）：
Transformer 本身还是一个典型的 encoder-decoder 模型，Encoder 端和 Decoder 端均有 6 Block，Encoder 端的 Block 包括两个模块，Decoder 端的 Block 包括三个模块，需要注意：Encoder 端和 Decoder 端中的每个模块都有残差层和 Layer Normalization 层。
Encoder 端的 Block：多头 self-attention 模块以及一个前馈神经网络模，

多头 self-attention 模块

query、key、value 的纬度均为 64，输出为 values 的加权和，其中分配给每个 value 的权重由 query 与对应 key 的相似性函数计算得来。


data-src="https://b3logfile.com/file/2021/06/10e61ac8d2634b20bc7feaedad950e9.jpeg?imageView2/2/interlace/1/format/jpg">

前馈神经网络模块

由两个线性变换组成，中间有一个 ReLU 激活函数。输入和输出的纬度均为 64，内层维度为 204。
<img src="https://ld246.com/images/img-loading.svg" alt="4 月 22 日-5 月 7 日腾讯 nlp 算法实习面试题 14 道" data-src="https://b3logfile.com/file/2021/06/265e1cfe7a4e4401b62ec1caba78c3f.png?imageView2/2/interlace/1/format/jpg">
Decoder 端的 Block：多头 self-attention 模块，多头 Encoder-Decoder attention 交互模块，以及一个前馈神经网络模块；

多头 self-attention 模块

与 Encoder 基本一致，但注意需要做 mask。

多头 Encoder-Decoder attention 交互模块

Q 来源于 Decoder 自身子模块的输出，K，V 来源于整个 Encoder 端的输出。

前馈神经网络模块

该部分与 Encoder 端的一致。
Transformer 为什么可以并行？
Encoder 支持并行化：
自注意力机制就是利用 xi 之间两两相关性作为权重的一种加权平均将每一个 xi 映射到 zi，在计每一个 zi 时只需要 xi，并不依赖 zi-1 。
Decoder 支持部分并行化的原因：
teacher force：是指在每一轮预测时，不使用上一轮预测的输出，而强制使用正确的单词。
masked self attention
Decoder 的并行化仅在训练阶段，在测试阶段，因为我们没有正确的目标语句，t 时刻的输入必依赖 t-1 时刻的输出，这时跟之前的 seq2seq 就没什么区别了。
transformer 的计算瓶颈：
在传统的 Transformer 中，每个 block 中都有 Multi-head Attention 和全连接层，随着序列度 N 的增加，全连接层的计算量是线性增长，而 attention 的计算量则是平方增长，因此，当序列长比较长的时候，attention 就会有很大的计算量。
多头注意力机制计算过程？
多次 attention 综合的结果至少能够起到增强模型的作用，也可以类比 CNN 中同时使用多个卷核的作用，直观上讲，多头的注意力有助于网络捕捉到更丰富的特征/信息。
多头 self-attention 模块，则是将 Q,K,V 通过参数矩阵映射后(给 Q,K,V 分别接一个全连接层)，后再做 self-attention，将这个过程重复 6 次，最后再将所有的结果拼接起来，再送入一个全连接层可。
<h2 id="6-BERT介绍一下原理-怎么用BERT计算文本相似度-有哪两种计算方法-">6.BERT 介绍一下理，怎么用 BERT 计算文本相似度，有哪两种计算方法？</h2>
参考答案：
BERT 原理：
BERT 句向量：取最后一两层 embedding 的平均或直接用[CLS]对应的 embedding，使用 BERT 计算文本相似度的效果是不好的。
原因：
一：BERT 句向量的空间分布是不均匀的，受到词频的影响。因为词向量在训练的过程中起到连上下文的作用，词向量分布受到了词频的影响，导致了上下文句向量中包含的语义信息也受到了破坏
二：BERT 句向量空间是各向异性的，高频词分布较密集且整体上更靠近原点，低频词分布较稀且整体分布离原点相对较远。因为高词频的词和低频词的空间分布特性，导致了相似度计算时，相似过高或过低的问题。在句子级：如果两个句子都是由高频词组成，那么它们存在共现词时，相似度可会很高，而如果都是由低频词组成时，得到的相似度则可能会相对较低；在单词级：假设两个词在语上是等价的，但是它们的词频差异导致了它们空间上的距离偏差，这时词向量的距离就不能很好的表语义相关度。

解决办法：
BERT-Flow：利用标准化流(Normalizing Flows)将 BERT 句向量分布变换为一个光滑的、各向性的标准高斯分布。
Sentence-Bert：在微调的时候直接把损失函数改写为余弦相似度，然后采用平方损失函数。
<h2 id="7-transformer里自注意力机制的计算过程-为什么要进行缩放-为什么要用多头-">7.transfomer 里自注意力机制的计算过程，为什么要进行缩放，为什么要用多头？</h2>
参考答案：
自注意力机制的计算过程：
query、key、value 的纬度均为 64，输出为 values 的加权和，其中分配给每个 value 的权重由 query 与对应 key 的相似性函数计算得来。

进行缩放的原因：

除以缩放因子，不让较大的值经过 softmax 后变得非常大
避免出现梯度消失

使用多头的原因：
多次 attention 综合的结果至少能够起到增强模型的作用，也可以类比 CNN 中同时使用多个卷核的作用，直观上讲，多头的注意力有助于网络捕捉到更丰富的特征/信息。
<h2 id="8-介绍下bert位置编码和transformer的区别-哪个好-为什么-">8.介绍下 bert 位置编码和 transformer 的区别，哪个好，为什么？</h2>
参考答案：
Transformer 解决并行计算问题的法宝，就是 Positional Encoding，简单点理解就是，对于一文本，每一个词语都有上下文关系，而 RNN 类网络由于其迭代式结构，天然可以表达词语的上下文系，但 transformer 模型没有循环神经网络的迭代结构，所以我们必须提供每个字的位置信息给 transormer，才能识别出语言中的顺序关系，为了解决这个问题，谷歌 Transformer 的作者提出了 position encoding。
原版的 Transformer 中，谷歌对 learned position embedding 和 sinusoidal position encodig 进行了对比实验，结果非常相近。而 Sinusoidal encoding 更简单、更高效、并可以扩展到比训练本更长的序列上，因此成为了 Transformer 的默认实现。
对于机器翻译任务，encoder 的核心是提取完整句子的语义信息，它并不关注某个词的具体位置什么，只需要将每个位置区分开（三角函数对相对位置有帮助）；而 Bert 模型对于序列标注类的下任务，是要给出每个位置的预测结果的。
<h2 id="9-sigmoid函数的缺点-为什么会产生梯度消失-不是以0为中心的话-为什么会收敛慢-">9.sigmoid 函数的缺点，为什么会产生梯度消失？不是以 0 为中心的话，为什么会收敛慢？</h2>
参考答案：
sigmoid 函数
特点：
它能够把输入的连续实值变换为 0 和 1 之间的输出，特别的，如果是非常大的负数，那么输出是 0；如果是非常大的正数，输出就是 1。
缺点：
缺点 1：在深度神经网络中梯度反向传递时导致梯度消失，其中梯度爆炸发生的概率非常小，而度消失发生的概率比较大。
缺点 2：Sigmoid 的 output 不是 0 均值（即 zero-centered）。
缺点 3：其解析式中含有幂运算，计算机求解时相对来讲比较耗时。对于规模比较大的深度网络这会较大地增加训练时间。
<h2 id="10-Layer-Normalization和Batch-Normalization-的区别">10.Layer Normalization 和 Bach Normalization 的区别</h2>
参考答案：
Batch Normalization 的处理对象是对一批样本， Layer Normalization 的处理对象是单个样。Batch Normalization 是对这批样本的同一维度特征做归一化， Layer Normalization 是对这单个

本的所有维度特征做归一化。
BatchNorm 的缺点：
1.需要较大的 batch 以体现整体数据分布
2.训练阶段需要保存每个 batch 的均值和方差，以求出整体均值和方差在 infrence 阶段使用

3.不适用于可变长序列的训练，如 RNN
Layer Normalization
Layer Normalization 是一个独立于 batch size 的算法，所以无论一个 batch 样本数多少都不影响参与 LN 计算的数据量，从而解决 BN 的两个问题。LN 的做法是根据样本的特征数做归一化。
LN 不依赖于 batch 的大小和输入 sequence 的深度，因此可以用于 batch-size 为 1 和 RNN 对边长的输入 sequence 的 normalize 操作。但在大批量的样本训练时，效果没 BN 好。
实践证明，LN 用于 RNN 进行 Normalization 时，取得了比 BN 更好的效果。但用于 CNN 时效果并不如 BN 明显。
<h2 id="11-Leetcode-8--字符串转换整数--atoi--考虑科学计数法">11.Leetcode 8. 字符串转换整（atoi），考虑科学计数法</h2>
解析：有三种方法：正常遍历、有限状态机和正则表达式，这里只提供正则表达式的参考答案。
s.lstrip()表示把开头的空格去掉
使用正则表达式：
^：匹配字符串开头
[+-]：代表一个 + 字符或-字符
?：前面一个字符可有可无
\d：一个数字
+：前面一个字符的一个或多个
max(min(数字, 231 - 1), -231) 用来防止结果越界
代码如下：

<h2 id="12-最长递增子序列">12.最长递增子序列</h2>

如上图所示，用 nums 表示原数组，results[i]表示截止到 nums[i] 当前最长递增子序列长度为 results[i]，初始值均为 1；
因为要找处递增序列，所以我们只需要找出 nums[i]>nums[j]（其中 j < i）的数，并将对的 results[j]保存在 tmp 临时列表中，然后找出最长的那个序列将 nums[i]附加其后面；
示例：假设 nums[i] = 5, i = 3
更新 results[i]时只需要考虑前面小于 results[i]的项，所以 results[0-2]均为 1；
此时 nums[i]前面小于 5 的项有 nums[2]，需要将 results[2]保存报 tmp 中，然后选取 tmp 中大的数值并加 1 后赋值给 results[i], 即 results[5] = 2
tmp 是临时列表更新 results 之后需要清空；

根据上述流程更新所有 results[i]，即可得到下图：

此时我们只需要找出 results 中最大的那个值，即为最长递增子序列的长度。
参考代码如下：

a1d9ea6e58.png?imageView2/2/interlace/1/format/jpg">
<h2 id="13-全排列">13.全排列</h2>
解析 1：
定义一个长度为 len(nums)的空表 output，从左往右一次填入 nums 中的数字，并且每个数只使用一次。可以枚举所有困难，从左往右每一个位置都一次填入一个数，用 used 表记录 nums[i]是已填入 output 中，如果 nums[i]木有填入 output 中则填入并标记，当 output 的长度为 len(nums 时，得到一个满足条件的结果，将 output 存入最终结果。然后将 output 回溯到未添加 nums[i]状，used 回溯为未标记 nums[i]状态，继续下一次尝试。
<img src="https://ld246.com/images/img-loading.svg" alt="4 月 22 日-5 月 7 日腾讯 nlp 算法实习面试题 14 道" data-src="https://b3logfile.com/file/2021/06/abc96d59701f420fbcafa0ad497d416.png?imageView2/2/interlace/1/format/jpg">
解析 2：
在解析 1 中，我们使用了两个辅助空间，output 和 used；为节省空间，我们可以除掉他们；们可以将给定的 nums 数组分成左右两个部分，用 n 表示数组个数，depth 表示当前需要填充的位，左边表示已经填好的内容[0~~depth)， 右边表示待填充的内容[depth~~n)，假设当前填的位置是 i，填充后为了保持上述结构，需要 nums[i]和 nums[depth]交换，在完成一次填充后的回过程，还需要再次交换，保持原有内容。整个流程如下图所示：
<img src="https://ld246.com/images/img-loading.svg" alt="4 月 22 日-5 月 7 日腾讯 nlp 算法实习面试题 14 道" data-src="https://b3logfile.com/file/2021/06/bbe52d3e474248b386c03ba3ba06160.jpeg?imageView2/2/interlace/1/format/jpg">
参考代码：
<img src="https://ld246.com/images/img-loading.svg" alt="4 月 22 日-5 月 7 日腾讯 nlp 算法实习面试题 14 道" data-src="https://b3logfile.com/file/2021/06/23a76e01dba341f7ae9c93b9d17da0d.png?imageView2/2/interlace/1/format/jpg">
<h2 id="14-合并两个有序数组并去重">14.合并两个有序数组并去重</h2>
<img src="https://ld246.com/images/img-loading.svg" alt="4 月 22 日-5 月 7 日腾讯 nlp 算法实习面试题 14 道" data-src="https://b3logfile.com/file/2021/06/b33da618929d42c0a4ace766107920f.png?imageView2/2/interlace/1/format/jpg">