Introduction of Pointer Network and Global Pointer Network

Pointer Network

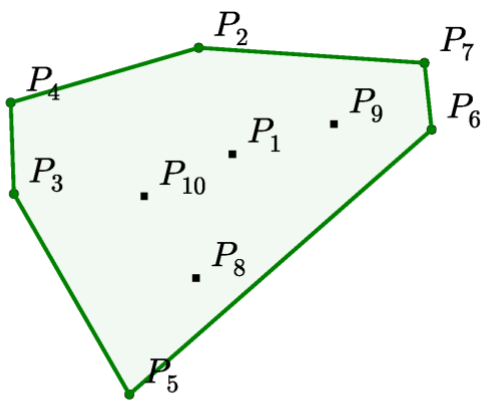
https://zhuanlan.zhihu.com/p/260745795

解决的问题

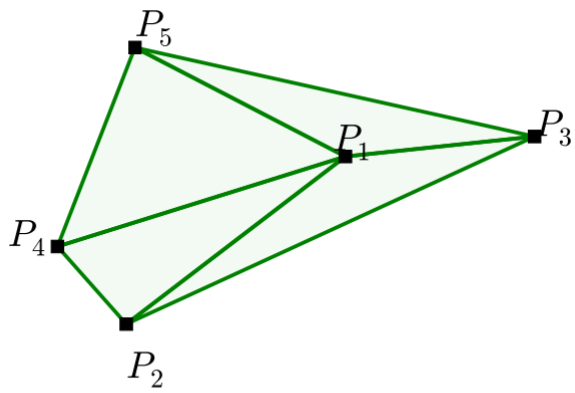
组合优化问题：从一堆候选集中，找到k个候选集，并达到最优。

- 现在你有n个各有特色的项目可以写到简历上，但你只能选3个项目，如何找到最优解？
- 桌上有m个苹果，它们的大小色泽都不一样，选择3个苹果放在橱窗上展示，如何找到最优解？
- 新闻推荐场景，给定30个新闻，选取3个在首页展示。
- 凸包求解（计算几何问题），有一堆的点，要找出最外围的点，使得这些点连起来，能够包围所有的点。
- Delaunay Triangulation 完美三角剖分：平面上的点集P是一种三角剖分，使得P中没有点严格处于剖分后中任意一个三角形外接圆的内部。

> 更多相关问题见：深度学习或强化学习在组合优化方面有哪些应用？



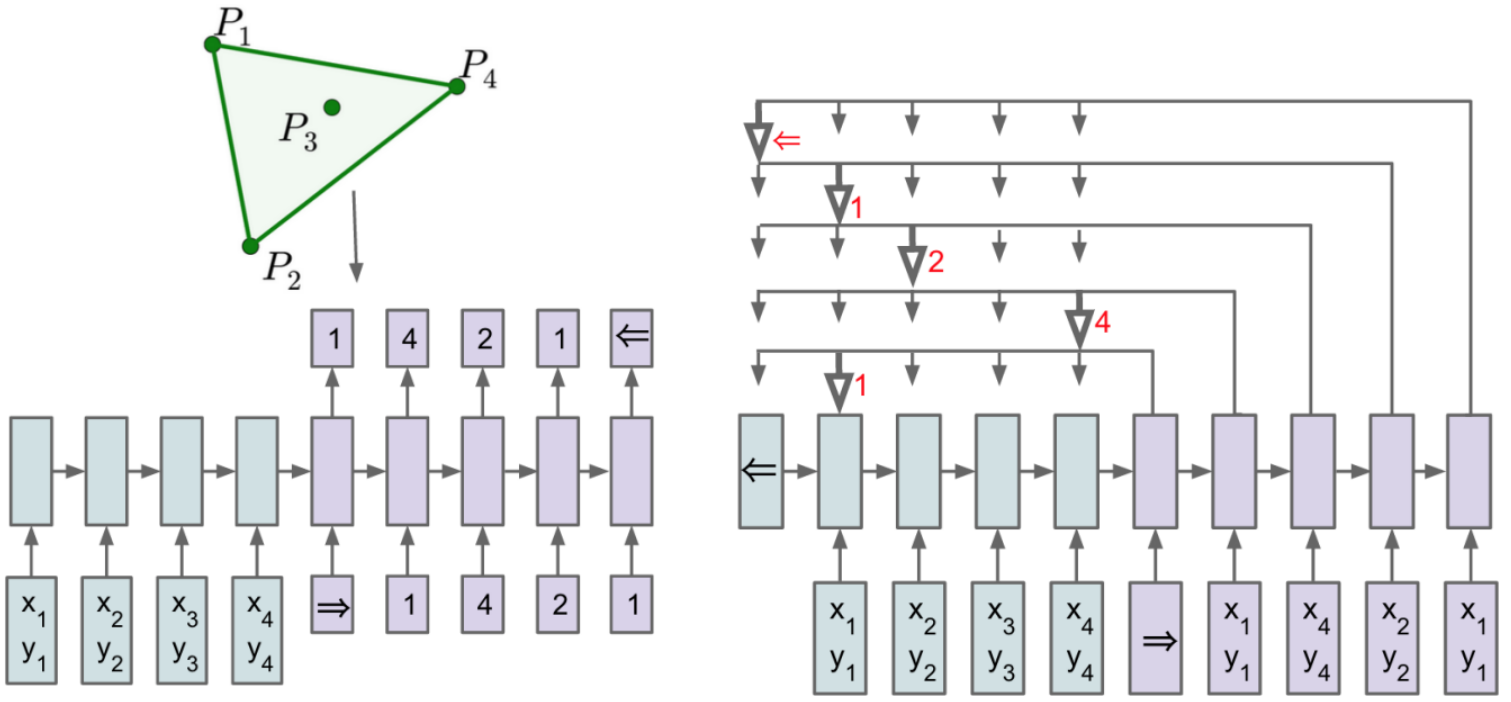
(a) Input $\mathcal{P} = \{P_1, \ldots, P_{10}\}$, and the output sequence $\mathcal{C}^{\mathcal{P}} = \{\Rightarrow, 2, 4, 3, 5, 6, 7, 2, \Leftarrow\}$ representing its convex hull.



(b) Input $\mathcal{P} = \{P_1, \ldots, P_5\}$, and the output $\mathcal{C}^{\mathcal{P}} = \{\Rightarrow, (1, 2, 4), (1, 4, 5), (1, 3, 5), (1, 2, 3), \Leftarrow\}$ representing its Delaunay Triangulation.

Figure 2: Input/output representation for (**a**) convex hull and (**b**) Delaunay triangulation. The tokens $\Rightarrow$ and $\Leftarrow$ represent beginning and end of sequence, respectively.

模型结构

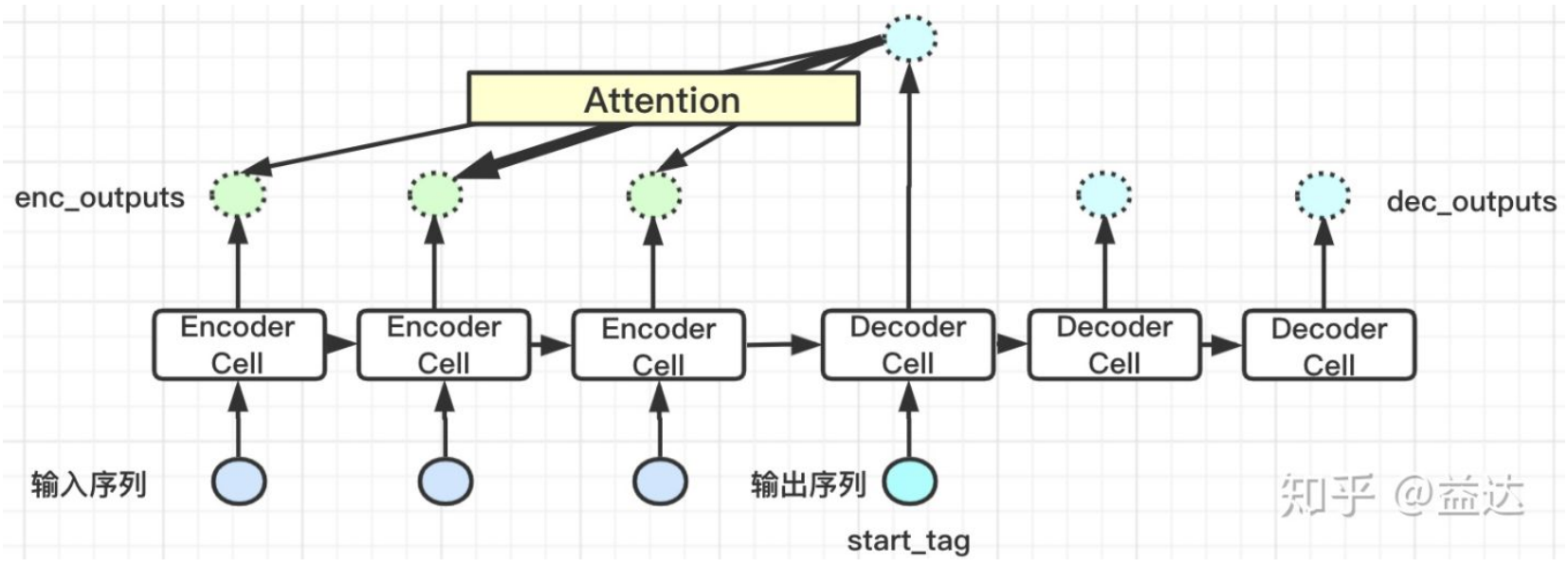


(a) Sequence-to-Sequence

(b) Ptr-Net

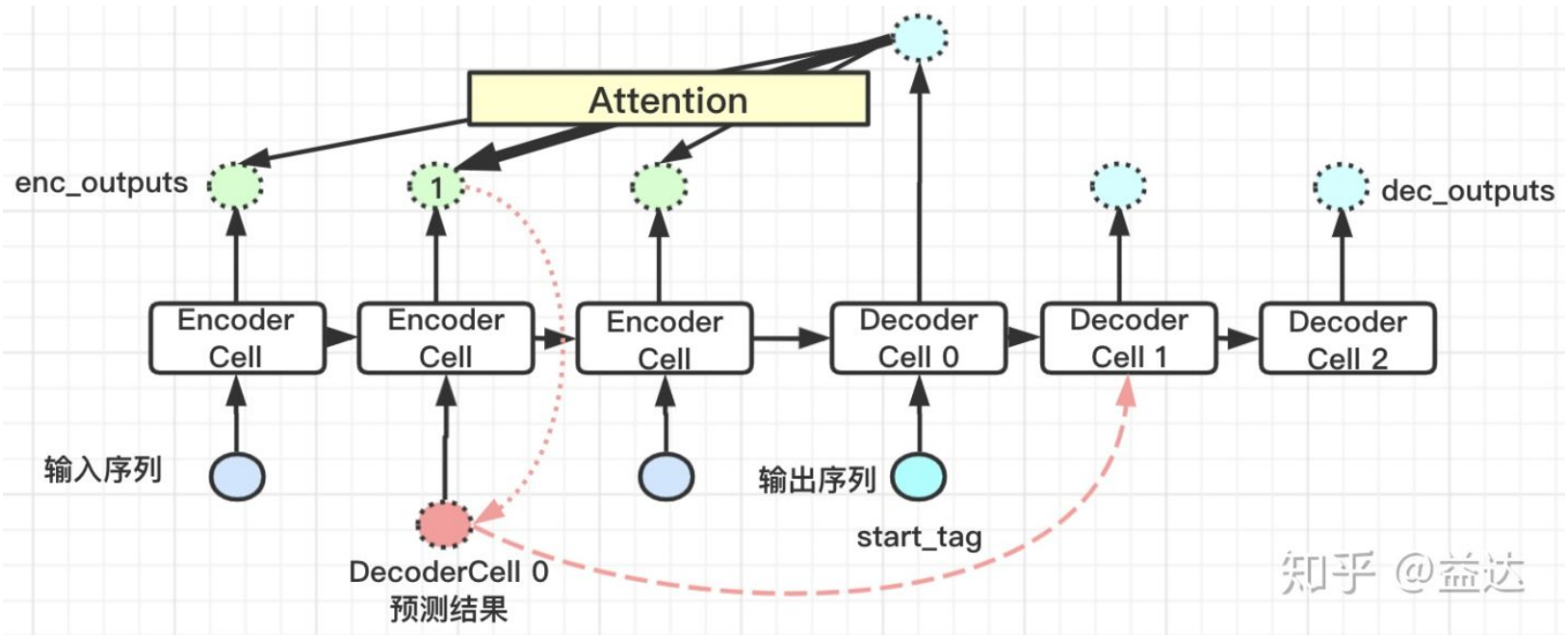
Figure 1: **(a)** *Sequence-to-Sequence* - An RNN (blue) processes the input sequence to create a code vector that is used to generate the output sequence (purple) using the probability chain rule and another RNN. The output dimensionality is fixed by the dimensionality of the problem and it is the same during training and inference [1]. **(b)** *Ptr-Net* - An encoding RNN converts the input sequence to a code (blue) that is fed to the generating network (purple). At each step, the generating network produces a vector that modulates a content-based attention mechanism over inputs ([5, 2]). The output of the attention mechanism is a softmax distribution with dictionary size equal to the length of the input.

Attention机制



在enc_outputs中，与当前DecoderCell输出做Attention，其实就是计算相关性然后softmax一下。

Pointer机制



在Attention机制的基础上，按照softmax结果最大的下标，选取对应的输入序列元素。**注意这个阶段只发生在预测阶段（因为Train阶段是Teach-Force不需要选）**，因此我们选出来的输入序列的第1个item，将作为DecoderCell1的输入并产生新的输出。将新的输出与enc_output做Attention，重复Pointer选取的步骤，能够得到另一个预测结果。

这里就显示出**Pointer机制**了，因为我们是通过**“引用”输入序列，作为我们的预测结果**，而不是预测了一个新的东西。就像C语言里的指针（Pointer）一样，指针没有单独开辟内存空间(除了自身所占的几个byte)，但却能指向具有内容的内存地址，使我们通过指针能直接访问相应的内容。

优点

传统的seq2seq模型是无法解决输出序列的词汇表会随着输入序列的改变而改变的问题，而pointer network的结构天生具备输出元素来自输入元素这样的特点，于是它非常适合用来实现“复制”这个功能，从而解决OOV问题。

> Multi-Source Pointer Network for Product Title Summarization
>
> 用户在移动设备上使用电子商务软件进行购物的时候，由于移动设备屏幕的限制，往往在列表页无法看到商品的完整名称，为了弄清楚这商品具体是什么不得不点开商品的详细信息页（即使用户还没有决定购买该商品），而这会有损用户体验。那么为了让用户在列表页就准确知道该商品的主要信息，就有必要对较长的商品标题做一个摘要，用短短几个词准确表达商品信息。

Global Pointer Network

https://kexue.fm/archives/8373

> 利用全局归一化的思路来进行命名实体识别（NER），可以无差别地识别嵌套实体和非嵌套实体。

基本思想

以实体为基本单位进行判别。如果有m种实体类型需要识别，那么就做成m个“$n(n+1)/2$选k”的多标签分类问题。

	北	京	大	学	在	深	圳	有	分	校	吗	？
北	0	0	0	0	0	0	0	0	0	0	0	0
京		0	0	0	0	0	0	0	0	0	0	0
大			0	0	0	0	0	0	0	0	0	0
学				0	0	0	0	0	0	0	0	0
在					0	0	0	0	0	0	0	0
深						0	0	0	0	0	0	0
圳							0	0	0	0	0	0
有								0	0	0	0	0
分									0	0	0	0
校										0	0	0
吗											0	0
？												0

Head-1：人名

	北	京	大	学	在	深	圳	有	分	校	吗	？
北	0	1	0	0	0	0	0	0	0	0	0	0
京		0	0	0	0	0	0	0	0	0	0	0
大			0	0	0	0	0	0	0	0	0	0
学				0	0	0	0	0	0	0	0	0
在					0	0	0	0	0	0	0	0
深						0	1	0	0	0	0	0
圳							0	0	0	0	0	0
有								0	0	0	0	0
分									0	0	0	0
校										0	0	0
吗											0	0
？												0

Head-2：地名

	北	京	大	学	在	深	圳	有	分	校	吗	？
北	0	0	0	1	0	0	0	0	0	0	0	0
京		0	0	0	0	0	0	0	0	0	0	0
大			0	0	0	0	0	0	0	0	0	0
学				0	0	0	0	0	0	0	0	0
在					0	0	0	0	0	0	0	0
深						0	0	0	0	0	0	0
圳							0	0	0	0	0	0
有								0	0	0	0	0
分									0	0	0	0
校										0	0	0
吗											0	0
？												0

Head-3：机构名

数学形式

设长度为n的输入t经过编码后得到向量序列$[h_1, h_2, \cdots, h_n]$，通过变换$q_{i,\alpha} = W_{q,\alpha}h_i + b_{q,\alpha}$和$k_{i,\alpha} = W_{k,\alpha}h_i + b_{k,\alpha}$我们可以得到序列向量序列$[q_{1,\alpha}, q_{2,\alpha}, \cdots, q_{n,\alpha}]$和$[k_{1,\alpha}, k_{2,\alpha}, \cdots, k_{n,\alpha}]$,它们是识别第$\alpha$种类型实体所用的向量序列。此时我们可以定义

$$s_\alpha(i,j) = q_{i,\alpha}^\top k_{j,\alpha} \tag{1}$$

作为从i到j的连续片段是一个类型为α的实体的打分。也就是说，**用$q_{i,\alpha}$与$k_{j,\alpha}$的内积，作为片段$t[i:j]$是类型为α的实体的打分（logits）**，这里的$t[i:j]$指的是序列t的第i个到第j个元素组成的连续子串。在这样的设计下，GlobalPointer事实上就是Multi-Head Attention的一个简化版而已，有多少种实体就对应多少个head，相比Multi-Head Attention去掉了V相关的运算。

位置编码

https://kexue.fm/archives/8130

纯粹的Attention模块无法捕捉输入的顺序信息，即无法区分不同位置的Token，需显式建模位置信息。

满足以下性质：

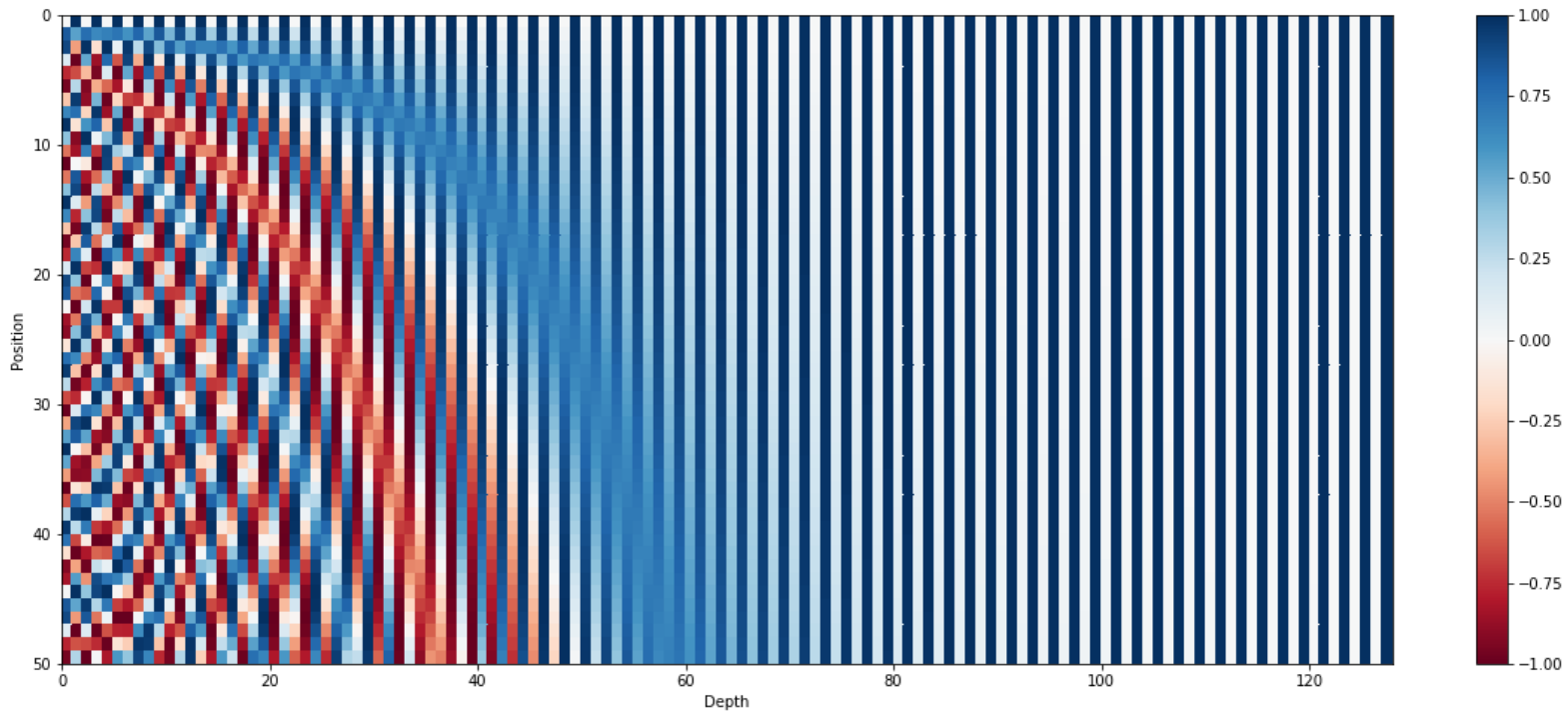
- It should output a unique encoding for each time-step (word’s position in a sentence)
- Distance between any two time-steps should be consistent across sentences with different lengths.

- Our model should generalize to longer sentences without any efforts. Its values should be bounded.
- It must be deterministic.

目前已有的位置编码方式主要有：

1. **绝对位置编码**：在输入的第k个向量x_k中加入位置向量p_k变为x_{k+p}，其中p_k只依赖于位置编号k。
   - **训练式**：直接将位置编码当作可训练参数。缺点是外推性不好，例如BERT一般的最大位置设为了512，因此顶多只能处理512个token，多出来的部分就没有位置编码可用了。
   - **三角式**：Sinusoidal位置编码。《Attention is All You Need》

$$\begin{cases} \boldsymbol{p}_{k,2i} = \sin\left(k/10000^{2i/d}\right) \\ \boldsymbol{p}_{k,2i+1} = \cos\left(k/10000^{2i/d}\right) \end{cases} \tag{2}$$



The 128-dimensional positonal encoding for a sentence with the maximum lenght of 50. Each row represents the embedding vector

   - **递归式**: 原则上来说，RNN模型不需要位置编码，它在结构上就自带了学习到位置信息的可能性（因为递归就意味着我们可以训练一个“数数”模型），因此，如果在输入后面先接一层RNN，然后再接Transformer，那么理论上就不需要加位置编码了。
2. **相对位置编码**: 相对位置并没有完整建模每个输入的位置信息，而是在算Attention的时候考虑当前位置与被Attention的位置的相对距离，由于自然语言一般更依赖于相对位置，所以相对位置编码通常也有着优秀的表现。

$$\begin{cases} \boldsymbol{q}_i = (\boldsymbol{x}_i + \boldsymbol{p}_i)\boldsymbol{W}_Q \\ \boldsymbol{k}_j = (\boldsymbol{x}_j + \boldsymbol{p}_j)\boldsymbol{W}_K \\ \boldsymbol{v}_j = (\boldsymbol{x}_j + \boldsymbol{p}_j)\boldsymbol{W}_V \\ a_{i,j} = softmax\left(\boldsymbol{q}_i \boldsymbol{k}_j^{\top}\right) \end{cases} \tag{3}$$

$$\left\{ \boldsymbol{o}_i = \sum_j a_{i,j} \boldsymbol{v}_j \right.$$

$$\boldsymbol{q}_i \boldsymbol{k}_j^\top = \boldsymbol{x}_i \boldsymbol{W}_Q \boldsymbol{W}_K^\top \boldsymbol{x}_j^\top + \boldsymbol{x}_i \boldsymbol{W}_Q \boldsymbol{W}_K^\top \boldsymbol{p}_j^\top + \boldsymbol{p}_i \boldsymbol{W}_Q \boldsymbol{W}_K^\top \boldsymbol{x}_j^\top + \boldsymbol{p}_i \boldsymbol{W}_Q \boldsymbol{W}_K^\top \boldsymbol{p}_j^\top \tag{4}$$

- **经典式**:

  起源于Google的论文《Self-Attention with Relative Position Representations》，华为开源的NEZHA模型也用到了这种位置编码.

$$a_{i,j} = softmax\left(\boldsymbol{x}_i \boldsymbol{W}_Q \left(\boldsymbol{x}_j \boldsymbol{W}_K + \boldsymbol{R}_{i,j}^K\right)^\top\right) \tag{5}$$

$$\boldsymbol{o}_i = \sum_j a_{i,j} \left(\boldsymbol{x}_j \boldsymbol{W}_V + \boldsymbol{R}_{i,j}^V\right) \tag{6}$$

$$\begin{aligned} \boldsymbol{R}_{i,j}^K &= \boldsymbol{p}_K \left[\text{clip}(i-j, p_{\min}, p_{\max})\right] \\ \boldsymbol{R}_{i,j}^V &= \boldsymbol{p}_V \left[\text{clip}(i-j, p_{\min}, p_{\max})\right] \end{aligned} \tag{7}$$

- **T5式**:

  丢弃一些部分，在Attention矩阵的基础上加一个可训练的偏置项, 可以直接将它作为参数训练出来。

$$\boldsymbol{q}_i \boldsymbol{k}_j^\top = \boldsymbol{x}_i \boldsymbol{W}_Q \boldsymbol{W}_K^\top \boldsymbol{x}_j^\top + \boldsymbol{\beta}_{i,j} \tag{8}$$

  特别的，T5对相对位置进行了一个“分桶”处理：

$i-j$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$f(i-j)$	0	1	2	3	4	5	6	7	8	8	8	8	9	9	9	9
$i-j$	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	$\cdots$
$f(i-j)$	10	10	10	10	10	10	10	11	11	11	11	11	11	11	11	$\cdots$

(9)

- **旋转位置编码**:

  配合Attention机制能达到“绝对位置编码的方式实现相对位置编码”的设计。而也正因为这种设计，它还是目前唯一一种可用于线性Attention的相对位置编码。

$$\underbrace{\begin{pmatrix} \cos m\theta_0 & -\sin m\theta_0 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_0 & \cos m\theta_0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_1 & -\sin m\theta_1 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_1 & \cos m\theta_1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2-1} & -\sin m\theta_{d/2-1} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2-1} & \cos m\theta_{d/2-1} \end{pmatrix}}_{\boldsymbol{\mathcal{R}}_m} \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_{d-2} \\ q_{d-1} \end{pmatrix} \tag{10}$$

$$(\boldsymbol{\mathcal{R}}_m \boldsymbol{q})^\top (\boldsymbol{\mathcal{R}}_n \boldsymbol{k}) = \boldsymbol{q}^\top \boldsymbol{\mathcal{R}}_m^\top \boldsymbol{\mathcal{R}}_n \boldsymbol{k} = \boldsymbol{q}^\top \boldsymbol{\mathcal{R}}_{n-m} \boldsymbol{k} \tag{11}$$

值得指出的是，$\boldsymbol{\mathcal{R}}_m$**是一个正交矩阵，它不会改变向量的模长，因此通常来说它不会改变原模型的稳定性。**

使用旋转位置编码可得：

$$s_\alpha(i,j) = (\boldsymbol{\mathcal{R}}_i \boldsymbol{q}_{i,\alpha})^\top (\boldsymbol{\mathcal{R}}_j \boldsymbol{k}_{j,\alpha}) = \boldsymbol{q}_{i,\alpha}^\top \boldsymbol{\mathcal{R}}_i^\top \boldsymbol{\mathcal{R}}_j \boldsymbol{k}_{j,\alpha} = \boldsymbol{q}_{i,\alpha}^\top \boldsymbol{\mathcal{R}}_{j-i} \boldsymbol{k}_{j,\alpha} \tag{12}$$

损失函数

https://kexue.fm/archives/7359/

这是一个用于多标签分类的损失函数，它是单目标多分类交叉熵的推广，特别适合总类别数很大、目标类别数较小的多标签分类问题。

$$\log\left(1 + \sum_{(i,j)\in P_\alpha} e^{-s_\alpha(i,j)}\right) + \log\left(1 + \sum_{(i,j)\in Q_\alpha} e^{s_\alpha(i,j)}\right) \tag{13}$$

其中P_α是该样本的所有类型为αα的实体的首尾集合，Q_α是该样本的所有非实体或者类型非α的实体的首尾集合，注意我们只需要考虑$i \leq j$的组合，即:

$$\begin{aligned} \Omega &= \big\{(i,j)\,\big|\, 1 \leq i \leq j \leq n\big\} \\ P_\alpha &= \big\{(i,j)\,\big|\, t_{[i:j]}\text{是类型为}\alpha\text{的实体}\big\} \\ Q_\alpha &= \Omega - P_\alpha \end{aligned} \tag{14}$$

而在解码阶段，所有满足$s_\alpha(i,j)>0$的片段$t[i:j]$都被视为类型为α的实体输出。

简要推导

1. **从交叉熵出发**

CrossEntropy形式：

$$-\log\frac{e^{s_t}}{\sum_{i=1}^{n}e^{s_i}}=-\log\frac{1}{\sum_{i=1}^{n}e^{s_i-s_t}}=\log\sum_{i=1}^{n}e^{s_i-s_t}=\log\left(1+\sum_{i=1,i\neq t}^{n}e^{s_i-s_t}\right)\tag{15}$$

凸优化课程学过：**极大值函数的解析逼近是 log-sum-exp**，

$$\begin{aligned}\max\left\{x_1\ldots,x_n\right\}&=\log\left(\exp\left(\max x_i\right)\right)\\&\leq\log\left(\exp\left(x_1\right)+\cdots+\exp\left(x_n\right)\right)\\&\leq\log\left(n\cdot\exp\left(\max x_i\right)\right)\\&=\max\left\{x_1,\ldots,x_n\right\}+\log\left(n\right)\end{aligned}\tag{16}$$

即:$\max\left\{x_1,\ldots,x_n\right\}<\mathrm{LSE}\left(x_1,\ldots,x_n\right)\leq\max\left\{x_1,\ldots,x_n\right\}+\log\left(n\right)$

有：

$$\log\left(1+\sum_{i=1,i\neq t}^{n}e^{s_i-s_t}\right)\approx max\begin{pmatrix}0\\s_1-s_t\\\vdots\\s_{t-1}-s_t\\s_{t+1}-s_t\\\vdots\\s_n-s_t\end{pmatrix}\tag{17}$$

这个loss的特点是：所有的非目标类得分$\left\{s_1,\cdots,s_{t-1},s_{t+1},\cdots,s_n\right\}$跟目标类得分$s_t$两两作差比较，它们的**差的最大值都要尽可能小于零**，所以实现了“**目标类得分都大于每个非目标类的得分**”的效果。

2. **固定类别输出数量多标签分类**

假如是有多个目标类的多标签分类场景，我们也希望“**每个目标类得分都不小于每个非目标类的得分**”，可写出如下形式：

$$\log\left(1+\sum_{i\in\Omega_{neg},j\in\Omega_{pos}}e^{s_i-s_j}\right)=\log\left(1+\sum_{i\in\Omega_{neg}}e^{s_i}\sum_{j\in\Omega_{pos}}e^{-s_j}\right)\tag{18}$$

其中Ωpos,Ωneg分别是样本的正负类别集合。

如果n选k的多标签分类中k是固定的话，那么直接用上式作为loss就行了，然后预测时候直接输出得分最大的k个类别。

3. **不定类别输出数量多标签分类**

对于k不固定的多标签分类来说，我们就需要一个阈值来确定输出哪些类。为此，我们同样引入一个额外的0类，希望目标类的分数都大于s_0，非目标类的分数都小于s_0，所以上式变成：

$$\begin{aligned}&\log\left(1+\sum_{i\in\Omega_{neg},j\in\Omega_{pos}}e^{s_i-s_j}+\sum_{i\in\Omega_{neg}}e^{s_i-s_0}+\sum_{j\in\Omega_{pos}}e^{s_0-s_j}\right)\\=&\log\left(e^{s_0}+\sum_{i\in\Omega_{neg}}e^{s_i}\right)+\log\left(e^{-s_0}+\sum_{j\in\Omega_{pos}}e^{-s_j}\right)\end{aligned}\tag{19}$$

如果指定阈值为0，那么就简化为:

$$\log\left(1+\sum_{i\in\Omega_{neg}}e^{s_i}\right)+\log\left(1+\sum_{j\in\Omega_{pos}}e^{-s_j}\right)\tag{20}$$

终得到的Loss形式了——“softmax + 交叉熵”在多标签分类任务中的自然、简明的推广，它没有类别不均衡现象，因为它不是将多标签分类变成多个二分类问题，而是变成目标类别得分与非目标类别得分的两两比较，并且借助于logsumexp的良好性质，自动平衡了每一项的权重。