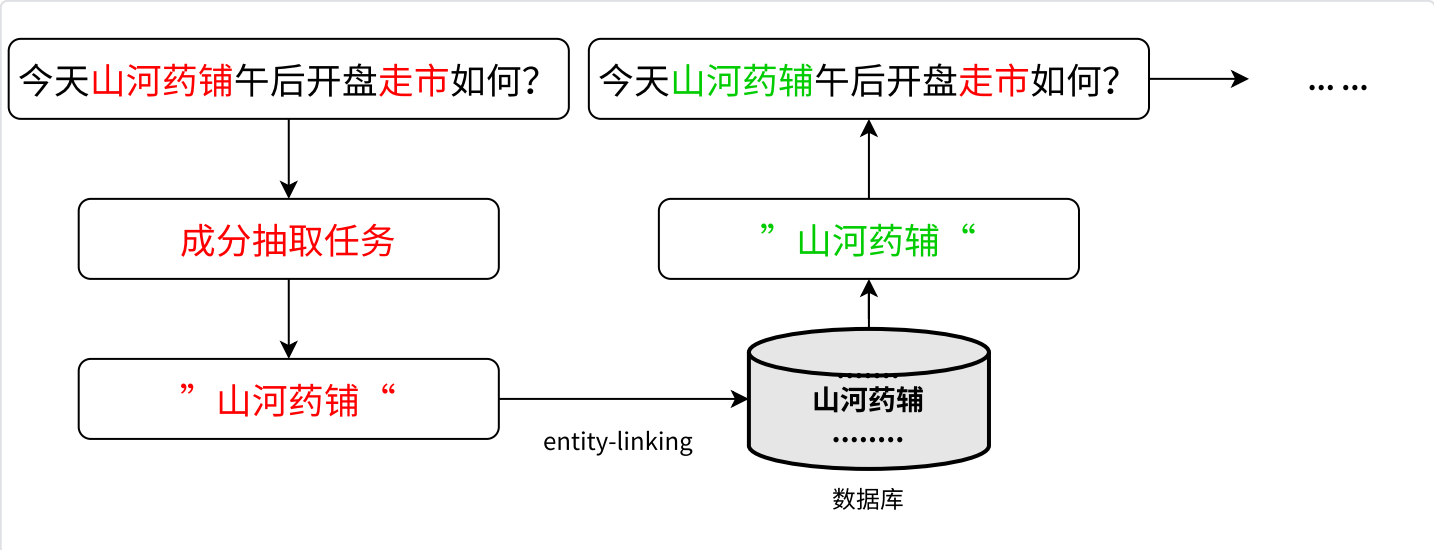
# 第三讲：成分抽取

小象学院

## 前言

文本的错误类型有多种，但是没有一种纠错技术可以纠正所有类型的文本错误，对于特定的错误类型来说，需要设计特定的方式去纠错，其中，垂直领域下的专属名词，例如：金融领域的股票和基金名称、医疗领域的药物和疾病名称、地图和本地生活领域的poi名称，等等，这些名词一旦发生错误，很难通过端到端的方式直接进行纠错，需要定制化的策略去应对。



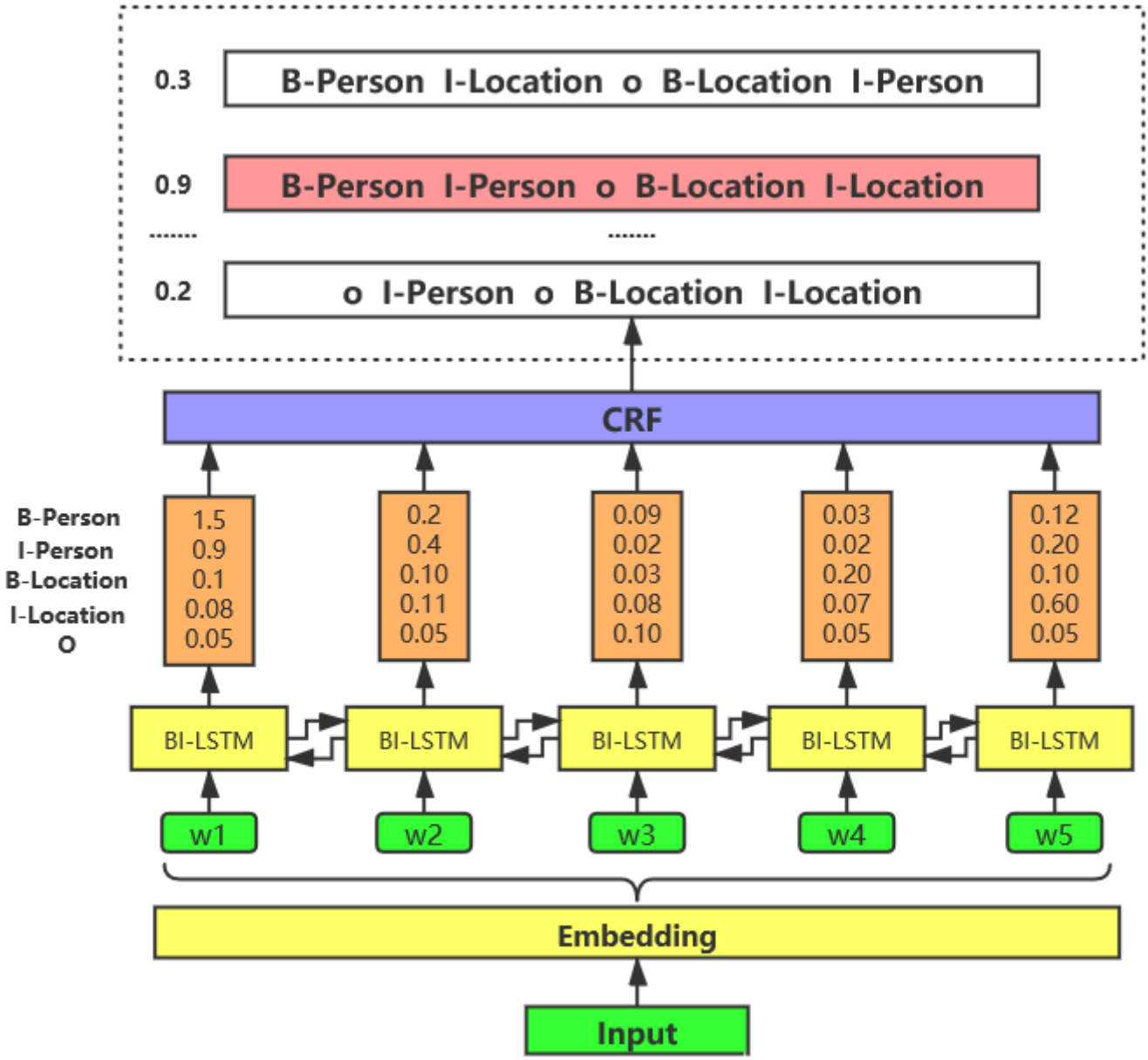

## 成分抽取常见方案

成分抽取的方案众多，这里讲解几个经典的，常用的抽取框架，例如序列标注、指针网络、片段排列等技术方案。

## 基于序列标注的抽取框架

实现序列标注的常用方法有BiLSTM/IDCNN/BERT+CRF等，与BiLSTM/IDCNN/BERT+Softmax相比，CRF使模型考虑到了标签之间的相关性，标签之间的相关性就是标签之间的转移概率，CRF层可以学习到标签之间的转移概率。

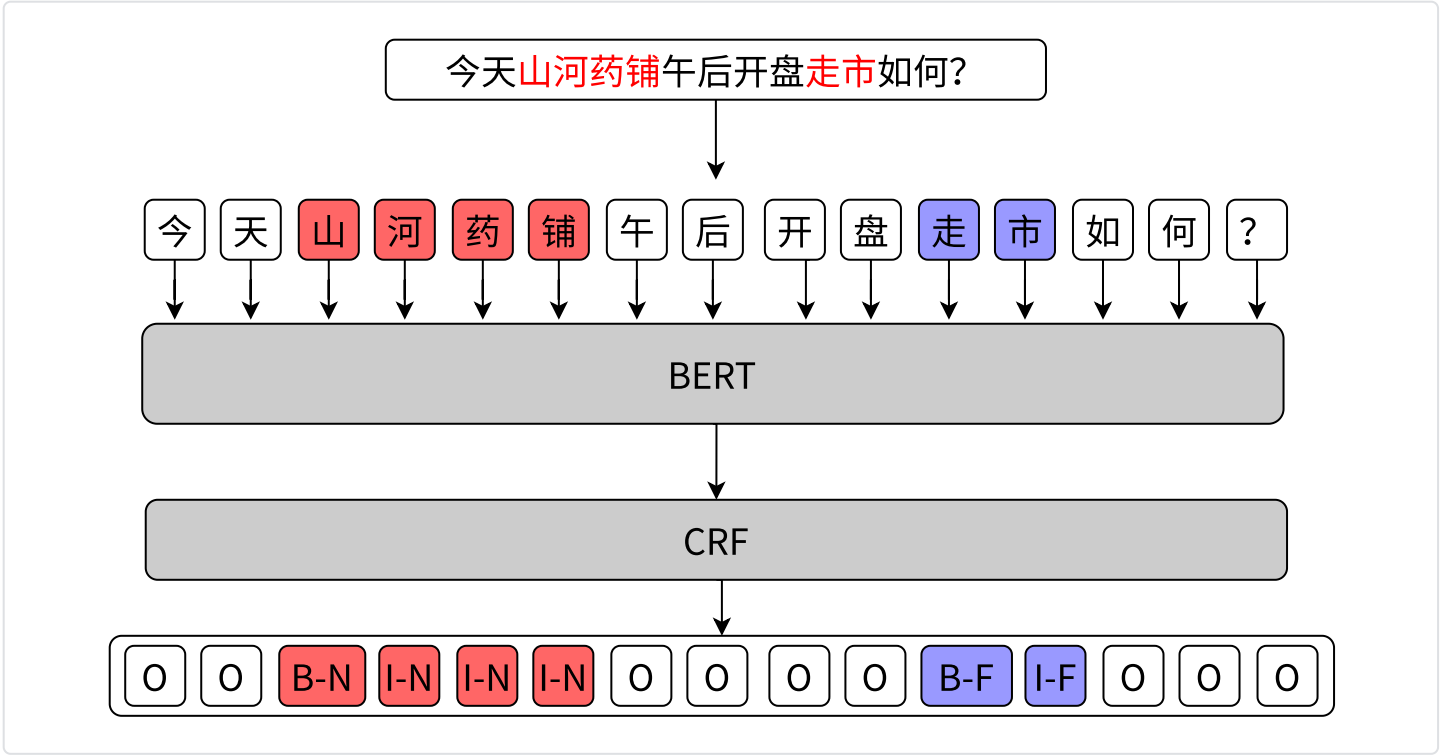
### BiLSTM+CRF




| | start | B-person | I-person | B-location | I-location | O | end |
|---|---|---|---|---|---|---|---|
| start | 0 | 0.8 | 0.009 | 0.76 | 0.00006 | 0.9 | 0..07 |
| B-person | 0 | 0.61 | 0.91 | 0.23 | 0.0007 | 0.67 | 0.008 |
| I-person | -1 | 0.56 | 0.53 | 0.56 | 0.0008 | 0.87 | 0.0078 |
| B-location | 0.9 | 0.51 | 0.0004 | 0.26 | 0.86 | 0.76 | 0.007 |
| I-location | -0.9 | 0.36 | 0.008 | 0.8 | 0.64 | 0.71 | 0.3 |
| o | 0 | 0.68 | 0.0006 | 0.8 | 0.00007 | 0.92 | 0.09 |
| end | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## BERT+CRF

自从bert横空出世之后，现在基本上都是基于bert这个强大的编码器来做的序列标注。

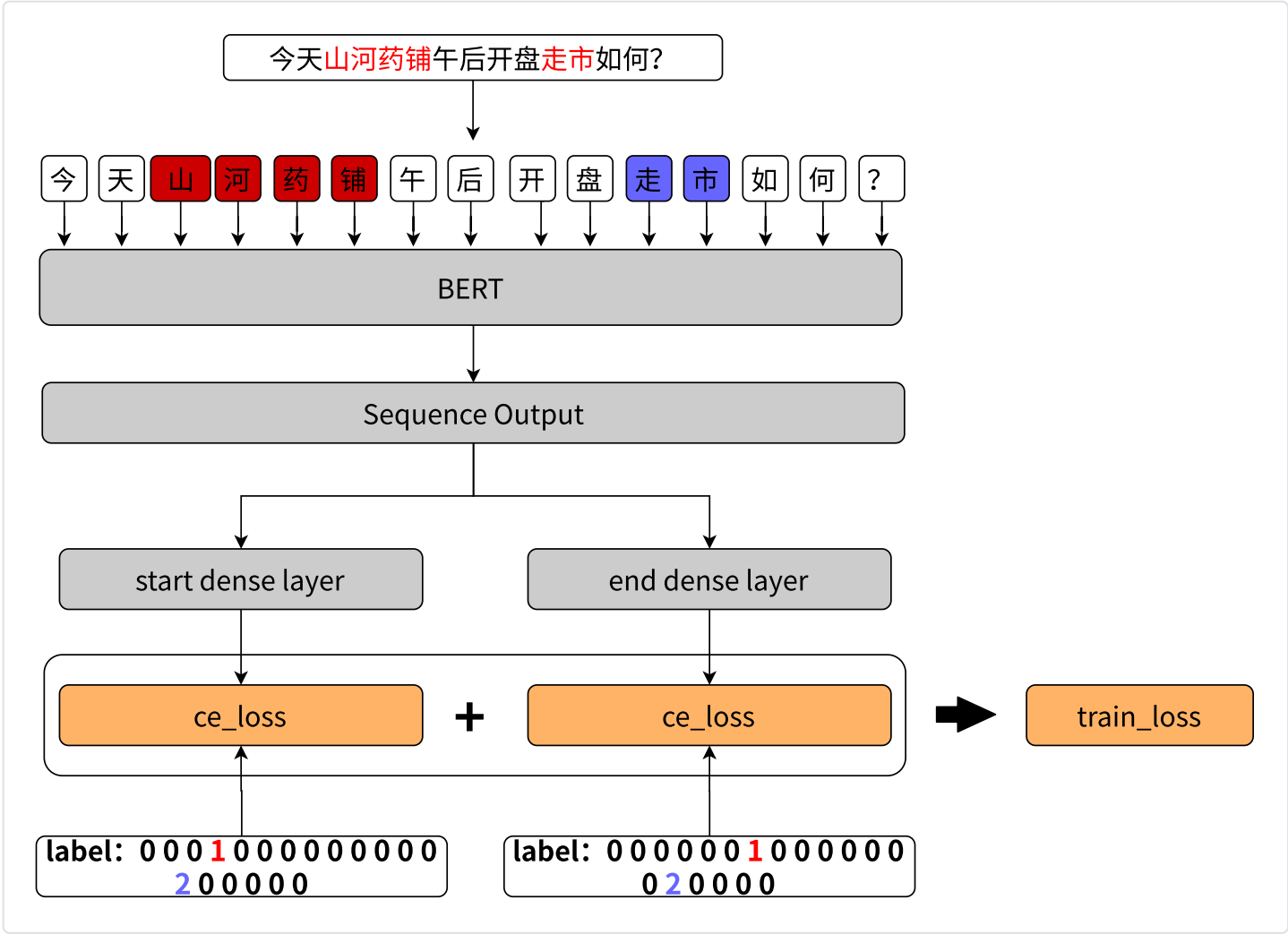



## 基于指针网络的抽取框架

### BERT+SPAN

从句子中正确的抽取出实体，需要知道三个要素：

- 开始位置；
- 结束位置；
- 实体类型




## BERT+MRC

BERT-MRC模型在不同的数据集上的表现差异较大，并且不同的数据处理对模型的性能影响也很大，需要使用者理解其内部的原理后采用合适的方法来使用。

假设现在现在待抽取的成分有三种：

- 股票名称：从公司全称中精选出2到4个字的称呼；
- 金融指标：用于衡量公司经营状况金融描述；
- 人名：人物的名称；

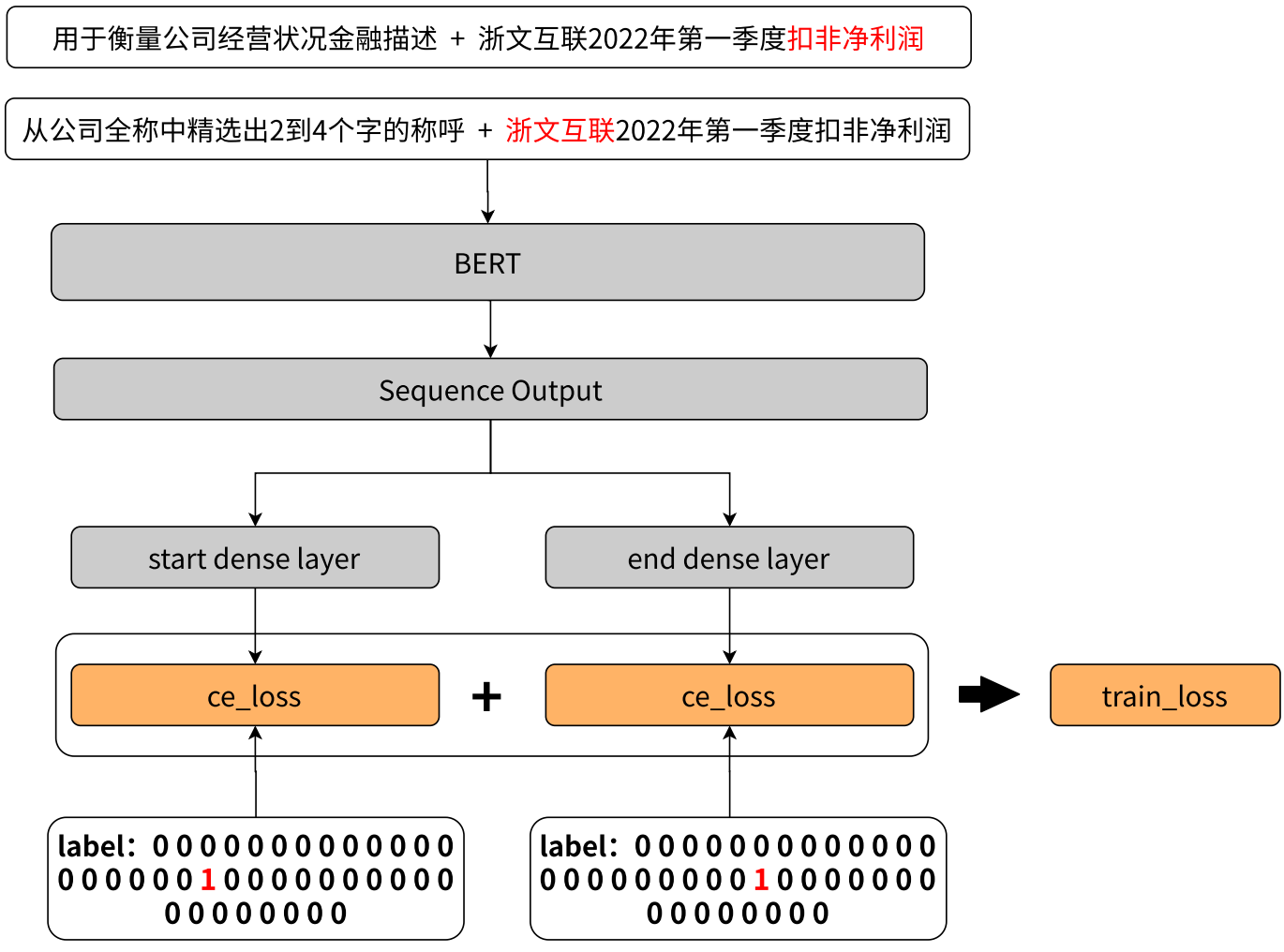
在数据预处理阶段需要将实体类型的描述作为query，拼接到每个输入句子的前面，然后针对性的修改原有的标注。例如下面的例子：

| | | |
|---|---|---|
| 从公司全称中精选出2到4个字的称呼 | + | 浙文互联2022年第一季度扣非净利润 |
| 用于衡量公司经营状况金融描述 | + | 浙文互联2022年第一季度扣非净利润 |
| 人物的名称 | + | 浙文互联2022年第一季度扣非净利润 |

**数据预处理：**

- 数据扩充：根据实体类型扩充；
- 类别均衡：平衡负例；

**模型结构**




所以，总结起来，BERT-MRC模型能取得比其他模型更好的性能的原因是它特殊的数据处理方法，在输入文本前加上了实体类型的描述信息，这些实体类型的描述作为先验知识提高了模型抽取的效果，所以BERT-MRC模型在数据量匮乏的场景下，通过在输入文本前面拼接的query获得了一定的先验信息，提升了性能。

## 基于片段排列的抽取框架

### GlobalPointer

GlobalPointer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，在非嵌套（Flat NER）的情形下它能取得媲美CRF的效果，而在嵌套（Nested NER）情形它也有不错的效果。


假设输入 $\boldsymbol{t}$ 的长度为 $\boldsymbol{n}$ ,经过编码( $\boldsymbol{lstm/bert}$ )后得到向量序列 $[\boldsymbol{h}_1, \boldsymbol{h}_2, \cdots, \boldsymbol{h}_n]$ ，通过变换 $\boldsymbol{q}_{i,\alpha} = \boldsymbol{W}_{q,\alpha}\boldsymbol{h}_i + \boldsymbol{b}_{q,\alpha}$ 和 $\boldsymbol{k}_{i,\alpha} = \boldsymbol{W}_{k,\alpha}\boldsymbol{h}_i + \boldsymbol{b}_{k,\alpha}$ ，我们可以得到序列向量序列 $[\boldsymbol{q}_{1,\alpha}, \boldsymbol{q}_{2,\alpha}, \cdots, \boldsymbol{q}_{n,\alpha}]$ 和 $[\boldsymbol{k}_{1,\alpha}, \boldsymbol{k}_{2,\alpha}, \cdots, \boldsymbol{k}_{n,\alpha}]$ ，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:

$$\boldsymbol{s}_\alpha(i,j) = \boldsymbol{q}_{i,\alpha}^\top \boldsymbol{k}_{j,\alpha}$$

作为从 $\boldsymbol{i}$ 到 $\boldsymbol{j}$ 的连续片段是一个类型为 $\boldsymbol{\alpha}$ 的实体的打分。也就是说，用 $\boldsymbol{q}_{i,\alpha}$ 与 $\boldsymbol{k}_{j,\alpha}$ 的内积，作为片段 $\boldsymbol{t}_{[i:j]}$ 是类型为 $\boldsymbol{\alpha}$ 的实体的打分（logits），这里的 $\boldsymbol{t}_{[i:j]}$ 指的是序列 $\boldsymbol{t}$ 的第 $\boldsymbol{i}$ 个到第 $\boldsymbol{j}$ 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 $\boldsymbol{V}$ 相关的运算。

### 相对位置(旋转编码)

理论上来说，上面那个设计就足够了，但实际上训练语料比较有限的情况下，它的表现往往欠佳，因为它没有显式地包含相对位置信息。相对位置信息很重要，加不加相对位置信息，效果可以相差30个百分点以上。

人民日报NER实验结果

| | 验证集F1 | 测试集F1 | 训练速度 | 预测速度 |
|---|---|---|---|---|
| CRF | 96.39% | 95.46% | 1x | 1x |
| GlobalPointer (w/o RoPE) | 54.35% | 62.59% | 1.61x | 1.13x |
| GlobalPointer (w/ RoPE) | 96.25% | 95.51% | 1.56x | 1.11x |

比如，我们要识别出公司名，输入的内容是：“腾讯：马化腾，阿里：马云，百度：李彦宏，京东：刘强东等人出席会议...”，这时候要识别出来的实体有很多，如果没有相对位置信息输入的话，GlobalPointer对实体的长度和跨度都不是特别敏感，因此很容易把任意两个实体的首尾组合都当成目标预测出来（即预测出“腾讯：马化腾，阿里”这样的实体）。相反，有了相对位置信息之后，GlobalPointer就会对实体的长度和跨度比较敏感，因此能更好地分辨出真正的实体出来。

加入旋转位置编码之后：

$$\boldsymbol{s}_\alpha(i,j) = \boldsymbol{q}_{i,\alpha}^\top \boldsymbol{k}_{j,\alpha}$$

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

旋转式位置编码参见苏剑林科学空间：https://spaces.ac.cn/archives/8265

## 对抗训练

当前，在各大NLP竞赛中，对抗训练已然成为上分神器，尤其是fgm和pgd使用较多，下面来说说吧。对抗训练是一种引入噪声的训练方式，可以对参数进行正则化，提升模型鲁棒性和泛化能力。

### FGM

FGM的全称是Fast Gradient Method, 出现于Adversarial Training Methods for Semi-supervised Text Classification 这篇论文，FGM是根据具体的梯度进行scale，得到更好的对抗样本：

$$r_{adv} = \epsilon g / ||g||_2$$

整个对抗训练的过程如下，伪代码如下：

1. 计算x的前向loss、反向传播得到梯度；
2. 根据embedding矩阵的梯度计算出r，并加到当前embedding上，相当于x+r；
3. 计算x+r的前向loss，反向传播得到对抗的梯度，累加到(1)的梯度上；
4. 将embedding恢复为(1)时的值；
5. 根据(3)的梯度对参数进行更新。

```
loss = model(**batch_data)[0]
loss.backward()
fgm.attack()
loss_adv = model(**batch_data)[0]
loss_adv.backward()
fgm.restore()
optimizer.step()
```

```
1 class FGM:
```

```
    def __init__(self, model: nn.Module, eps=1.):
        self.model = (
            model.module if hasattr(model, "module") else model
        )
        self.eps = eps
        self.backup = {}

    # only attack word embedding
    def attack(self, emb_name='word_embeddings'):
        for name, param in self.model.named_parameters():
            if param.requires_grad and emb_name in name:
                self.backup[name] = param.data.clone()
                norm = torch.norm(param.grad)
                if norm and not torch.isnan(norm):
                    r_at = self.eps * param.grad / norm
                    param.data.add_(r_at)

    def restore(self, emb_name='word_embeddings'):
        for name, para in self.model.named_parameters():
            if para.requires_grad and emb_name in name:
                assert name in self.backup
                para.data = self.backup[name]

        self.backup = {}

```

## PGD (Projected Gradient Descent)

FGM直接通过epsilon参数一下子算出了对抗扰动，这样得到的可能不是最优的。因此PGD进行了改进，多迭代几次，慢慢找到最优的扰动。

$$r_{adv|t+1} = \alpha g_t / ||g_t||_2$$

$$||r||_2 \leq \epsilon$$

pgd整个对抗训练的过程如下，伪代码如下：

1.计算x的前向loss、反向传播得到梯度并备份；

2.对于每步t：

a.根据embedding矩阵的梯度计算出r，并加到当前embedding上，相当于x+r(超出范围则投影回epsilon内)；

b.if t 不是最后一步，则进行b步骤：将模型梯度归0，根据a的x+r计算前后向并得到梯度,继续a步骤；if t 是最后一步,则进行c步骤.

c.恢复(1)的梯度，根据a的x+r计算前后向得到梯度并将梯度累加到(1)的梯度上，跳出循环；
3.将embedding恢复为(1)时的值；

4.根据2c的梯度对参数进行更新。

可以看到，在循环中r是逐渐累加的，要注意的是最后更新参数只使用最后一个x+r算出来的梯度。

```
loss = model(**batch_data)[0]
loss.backward()
pgd.backup_grad()
for _t in range(pgd_k):
        pgd.attack(is_first_attack=(_t == 0))
        if _t != pgd_k - 1:
                model.zero_grad()
        else:
                pgd.restore_grad()
        loss_adv = model(**batch_data)[0]
        loss_adv.backward()
pgd.restore()
optimizer.step()
```

## 领域知识融入

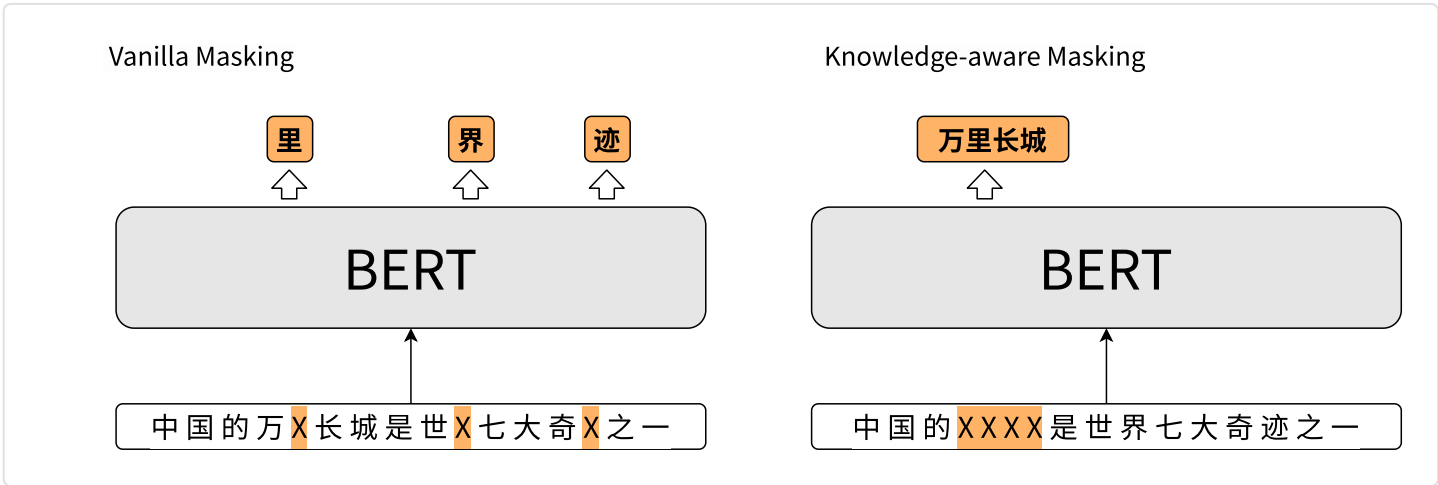
开源的中文BERT模型是基于中文维基百科数据训练得到，属于通用领域预训练语言模型。但是下游业务场景中有海量的业务数据，为了充分发挥领域数据的优势，在中文BERT模型上加入领域数据继续训练进行领域自适应（Domain Adaptation），使得模型更加匹配下游的业务场景，实践证明，这种Domain-aware Continual Training方式，有效地改进了BERT模型在下游任务中的表现。

| Benchmark | Metric | Google BERT | MT-BERT |
| --- | --- | --- | --- |
| MSRA-NER | F1 | 95.76% | 95.89% |
| LCQMC | Accuracy | 86.06% | 86.74% |
| ChnSentiCorp | Accuracy | 92.25% | 95.00% |
| NLPCC-DBQA | MRR | 93.55% | 94.07% |
| XNLI | Accuracy | 77.47% | 78.10% |
| 细粒度情感分析 | Macro-F1 | 71.63% | 72.04% |
| Query意图分类 | F1 | 92.68% | 93.27% |
| Query成分分析（NER） | F1 | 90.66% | 91.46% |

## Knowledge-aware Pretrain

在BERT预训练过程中融入知识图谱信息。知识图谱可以组织现实世界中的知识，描述客观概念、实体、关系。这种基于符号语义的计算模型，可以为BERT提供先验知识，使其具备一定的常识和推理能力。

BERT在进行语义建模时，主要聚焦最原始的单字信息，却很少对实体进行建模。具体地，BERT为了训练深层双向的语言表征，采用了Masked 训练策略。该策略类似于传统的完形填空任务，即在输入端，随机地“遮蔽”掉部分单字，在输出端，让模型预测出这些被“遮蔽”的单字。




小象学院