

JioJio 一下，搞定 CRF 算法

冬日新雨（崔诚煜）

2022 年 3 月 28 日星期一

前言

在自然语言处理（NLP）领域，**条件随机场**（Conditional Random Field - CRF）算法可以算得上是一个经典的处理序列标注的算法。该算法最早是在 2001 年被提出，但是我最早接触该算法还是在 2011 年读研的时候，那个时候的 NLP 还没有被神经网络和深度学习碾压，NLP 的研究进展依然处于一个比较缓慢的阶段。当时在中文领域（当然也包括日文、韩文等无空格分割的语种领域），为数不多的能够落地的一些应用就是**分词**，这个任务基本都需要采用 CRF 算法来处理。

可以说，在深度学习杀入 NLP 领域之前，CRF 算法称霸了序列标注领域。

在深度学习杀入 NLP 领域以前，NLP 的发展是个什么状态呢？那时我还在读大学，而且也并未真正接触 NLP，我只能简单凭借自己曾经的经验来简单说一下。

大约在 2009 年，我曾经做过一段时间的兼职英语笔译，当时中国的对外贸易比较火热，有很多的小工厂、小公司需要和国外进行邮件、文件、法律合同等信息的往来，我就通过翻译公司，接到了一些法律合同、会议发言稿等资料，完成中译英、或者英译中。那时，以我的英语四、六级水平要真正流畅完成稿件翻译是困难的，我会把这些稿件，一句一句，粘贴在百度翻译网站中，然而百度的机器翻译实际上还是基于短语的概率统计模型，而非基于神经网络的 Seq2seq 模式，导致机器翻译的结果完全是破碎的，我需要把机器翻译得到的结果，再根据英语语序和语感，一句一句重新拼接起来。可以说，那时的百度机器翻译完全就是个加强版的电子词典的作用。当然，现在做英语笔译的门槛已经大大降低，一来英语越来越普及，二来，只要把文本粘贴进 Google 和百度翻译网站里，得到的结果八九不离十，稍微改一改很少的词汇就可以了。那时，还有一个非常有名的计算机辅助翻译软件 Trados，作用类似于如今的搜狗输入法，具有记忆和联想功能，用于辅助翻译，随着机器翻译的兴起，这款软件也已经慢慢落寞了。

以上是题外话，类似于曾经的一些基于短语概率模型的机器翻译，在当前的环境下，CRF 算法在处理一些序列标注问题上，确实已经算不上一个最优解，而且其短板也逐渐暴露无疑，例如，很难并行化处理，模型参数量指数级爆炸等等。但是我依然认为，这个算法在工程中依然有存在的意义，这个之后再讲。

CRF 算法是一个比较难的概率图算法，学习门槛也比较高，这也是我写这篇文章的初衷。当我们翻开宗成庆老师的《统计自然语言处理》、李航老师的《统计学习方法》以及其它一些机器学习书籍的时候，基本都可以看到，CRF 基本都放在末尾的章节。对于我和我的研究生同学来说，第一次看这个算法是挺难理解

的，一来，是这个算法本身就比较抽象，二来是这些教材也不会细讲，书中往往列出几个关键建模公式，就到此为止了。

后来，又从网上找到了由 Charles Sutton 和 Andrew McCallum 共同编写的《An Introduction to Conditional Random Fields for Relational Learning》这是一篇很经典的文章，读了这篇文章，才算真正看懂了 CRF 算法。

然而，看懂和真正用其来实现算法模型，深刻领会这个算法里里外外，又是进一层理解。于是，我又用代码编写了一遍 CRF 算法，并仔细地调参，希望能够实现一个真正好用的分词工具，这个工具在 Github 上开源，命名为 jiojio，地址链接为 <https://github.com/dongrixinyu/jiojio>，安装方式为

pip install jiojio

因此，本文将采用最易理解的方式来介绍 CRF 算法，希望这篇文章能够对读者（包括机器学习算法学习者，以及非机器学习算法领域的读者）理解 CRF 算法与其在分词上的应用有一个透彻的理解，对学习、面试、都有所帮助。

序列标注问题

文本是序列化的，是一个字紧接一个字的，先后顺序不可以变换，例如“皇马对奥尼尔不感兴趣”，这句话变成“马尔对皇感奥尼兴不趣”，自然是无法理解。此时有人会杠“研表究明，汉字的序顺影不响的人理解。”那这只能说，人脑的思维结构和模式还需要我们进一步地探索。

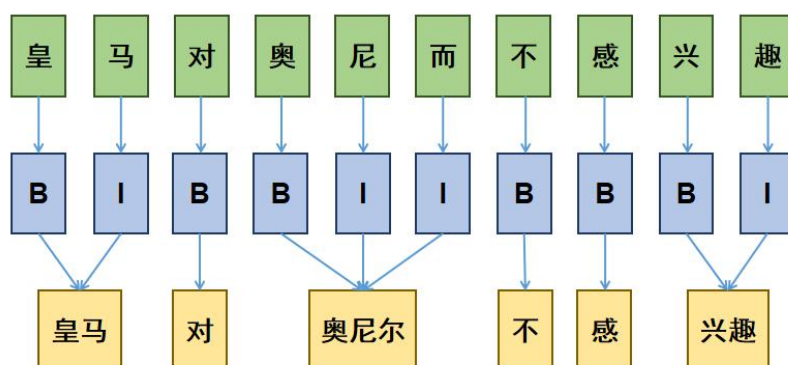
若我们想找出上述文本中的球队名“皇马”，最直观的建模方式，就是为其标识出球队名的偏移量。如下图。



我们在文本中，为想要找出的信息（球队名）打上位置标记（Team 标签），就是序列标注的基本建模方式，以上 NLP 任务被称为**实体识别**。

最常见的，我们一般将序列标注用于建模 NLP 中的分词、词性标注、实体识别等任务；此外，还包括语义角色标注、语义槽识别和填充、观点抽取、要素抽取等任务，而后者这些模型往往是很难用 CRF 模型来解决的，原因后续解释。

CRF 就是针对序列标注任务的一种解决方法，这里，我们一般均指线性链 CRF，即其所解决的问题可以抽象为一条线性链，前后顺序依次固定，但相互之间有依赖关系。以分词为例，其建模方式如下：



图中，B 标签表示字符是一个词汇的起始，I 标签表示一个词汇的中间或者结尾。当我们采用 CRF 算法计算出每一个汉字的标签后，也就相当于计算出了分词之后的词汇结果。CRF 的目的，就是从绿色的中文汉字，计算出其对应的蓝色标签。

CRF 的建模公式

CRF 模型可以看作一个黑盒，输入是汉字序列，输出是一串标签序列。以上述分词任务为例，从上面的图中，我们可以看到，我们就是希望找到“皇马”、“对”、“奥尼尔”、“不”、“感”、“兴趣”，这样的分词结果是正确的。因此，我们希望 CRF 模型能够正确地输出这样的标签序列，而非其它错误标签序列。

从图中可以看到，实际上，标签只有两种，B 和 I。总共有 10 个字，那么，可能的输出标签序列实际上是 $2^{10} = 1024$ 种。当然，这是一个指数级的结果，假设我们的文字序列是无限长的话，这个结果的数量是不可计数的。我们希望从这无数的结果中，找到上述正确的标签。

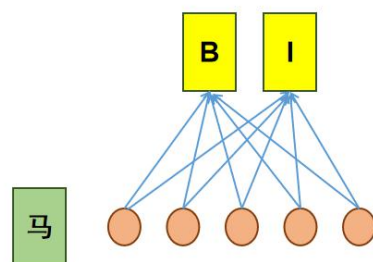
如果用 x 表示输入的文字序列， y 表示输出的标签序列，那么，采用概率模型建模，我们希望在已知 x 的情况下，使上述的 $y_{correct}$ 的概率最大，则建模公式就可以表示成

$$\operatorname{argmax} P(y_{correct}|x)$$

找到最大概率的 $y_{correct}$ 标签也就建模完成的第一步，想要建模，根据机器

学习的基本概念，此时我们需要设定模型的参数了，毕竟，只有有了参数，才能够进行建模。这就是建模的第二步。

我们单独抽出例子中的“马”字，可以看到，实际上，我们需要设定一组参数，来判断“马”字到底属于 **B**，还是 **I**（在别的任务中，标签数量更多）。这实际上就是一个二分类（多分类）的问题。如果接触过神经网络和逻辑回归的话，在上例中，我们完全可以按照一个神经网络与一个 **softmax** 层的方式来为“马”字建立一个模型，如下图



在这个图中，橙色的圆圈表示，我们为“马”字设定了一系列的参数特征，然后这些参数特征又分别表征了“马”字属于“**B**”和“**I**”标签。把所有的参数值全部相加，就得到了“马”字属于“**B**”和“**I**”的总权重值。考虑最简单的情况，哪个值大，我们就认定“马”字应当属于哪个标签。

可以仔细想一下，这和神经网络与逻辑回归简直就是一模一样有木有？其实 **CRF** 作者当时对概率图模型进行参数化引进的时候，就是完全按照这种方式来建模的，因为这是最简单、最有效的建模方式。

我们用公式来描述一下， x_i 表示一条序列中的第 i 个字符， y_i 表示其对应的标签 t ，那么，我们可以为其设定一系列参数 $w_{i,t}$ ，注意，这是一个 $i \times t$ 大小的矩阵，也就是图中橙色的圆圈映射到两个标签的参数矩阵，在上图中，这个矩阵大小为 5×2 。这样，我们就可以得到了一个描述“马”字的参数矩阵。

$$P(y_i|x) = \text{softmax}(x_i^T w_{i,t}) = \frac{\exp(y_t)}{\sum_{y_i} \exp(x_i^T w_{i,t})}$$

上述公式中，对于输入的数据 x_i ，经过一个矩阵运算得到标签的权重向量，经过一个 **softmax** 计算，即可得到该位置的标签概率分布。

随着深度学习以及各种花里胡哨的神经网络的引入，对 **CRF** 模型中的参数化建模已经远远不止于仅仅插入一层神经网络这么简单，逐渐演化出了 **Bi-LSTM-CRF**、**Bi-GRU-CRF**、**Bert-CRF** 等等更加精细化的模型、更加契合人类自然语言特点的参数模型。他们一切都来源于上述这个简单的单层矩阵变换。

当然，除了“马”字，文本序列中的每一个汉字都应当这样分配一套参数权重矩阵，那么，我们最终得到的序列的参数建模就变成了如下形式：

$$P(y|x) = \frac{\exp(w^T x)}{\sum_y \exp(w^T x)}$$

我们应当注意到，上式中，笼统地表达了，将整个输出的标签序列 y 当作一个整体，为其分配一套矩阵参数，然后通过 **softmax** 方式计算得到标签权重值，这样一种计算方式。

当然，这里是没有细细展开参数的具体表达的。我们接下来，对 **CRF** 中的参数做一个更加仔细的分析。

CRF 的特征选择

了解了 **CRF** 模型的建模公式后，有一个疑问，即上图中，橙色的圈圈到底是什么呢？

这得继续从“马”字说起。

在汉语中，“马”字的含义非常多，一种是动物，例如“斑马”、“矮脚马”等等；还有一种是姓氏，例如“马化腾”、“马保国”（我劝你耗子尾汁）等等；再或者，如上例，“皇马”是作为一个单独的球队名字进行命名的，它属于没什么规律的，单纯的命名，因此需要单独考虑。

那么我们可以为“马”字构建如下特征：

特征 1：当“马”字后面紧跟了一些人名常用字，如“刚、强、丽、腾、国、艳”等字时，马字的标签，应当为“**B**”（原因是人名当中，“马”字应当在词首）；

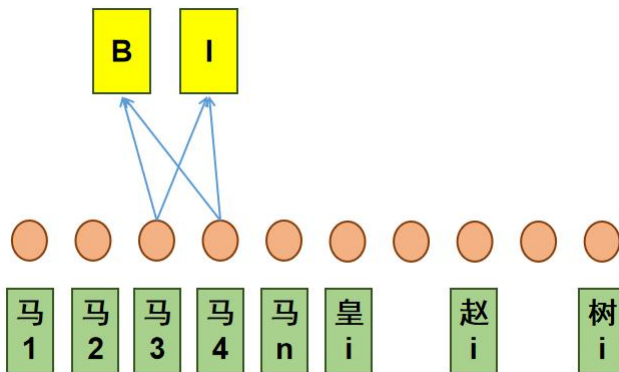
特征 2：当“马”字表示动物名时，如“斑马”、“矮脚马”，其标签应当为“**I**”；（表示这里指示一种动物，因为在这些动物命名中，“马”字在末尾）；

特征 3：当“马”字的前一个字为“皇”字时，其标签应当为“**I**”，专门针对本例中的“皇马”而设定；

特征 4：“马”字单独本身也具有一个特征，从语料中统计了 10000 个带“马”字的词语，我们知道，它更倾向于被标为“**B**”。

当然，特征还可以定义很多，此处仅举若干。若把每一个汉字的所有特征列

个七七八八，实际上也就对汉语中所有的汉字和其词汇做了一个特征大汇总，那这就是一本大型的汉语字典。放在模型中，这样的特征数量往往在百万级起步。如下图所示：



从这张图中，可以看到，橙色圆圈标识了汉语中所有的特征，绿色就是对该特征的说明，即指明该特征是用来描述哪个汉字的。

其中，“马3”和“马4”就对应了上例中的“特征3”和“特征4”。针对“皇马”这个例子，其关联到的特征就是“马3”和“马4”，剩余的“马1”和“马2”根本对于“皇马”的判断是无用的。也就是，模型中剩余所有的百万计的特征在此时都是无用的，在这里可以用一个特征匹配函数来过滤一下。

$$f(y_i, x) = \begin{cases} 1 & \text{当 } i \text{ 位置特征被 } x \text{ 匹配时} \\ 0 & \text{当 } i \text{ 位置特征未被 } x \text{ 匹配时} \end{cases}$$

实际上，也就是当遇到“皇马”这样的文本时，需要在百万的特征上覆盖一层类似于 Bert 模型中的 Mask 蒙板，用来屏蔽掉那些对文本，也就是 x ，无用的参数和特征。

接下来，我们来分别计算一下这些特征对应的标签的权重值。

假设，我们训练好了整个的 CRF 模型，特征3 包含两个参数：

参数1：当“马”字的前一个字为“皇”字时，其标签应当为“l”。参数值为 1.3（这里的参数值非概率值，而是模型的参数权重）；

参数2：当“马”字的前一个字为“皇”字时，其标签应当为“B”。参数值为 -0.2。

此外，我们需要考虑，“马”字本身就是一个特征，也就是特征4，它也包含两个参数：

参数1：“马”字其标签应当为“l”。参数值为 0.2；

参数2：“马”字其标签应当为“B”。参数值为 0.3。

将两个特征相加，就可以得到针对标签的权重值：

$$w(B|\text{马, 皇马}, x) = -0.2 + 0.3 = 0.1$$

$$w(I|\text{马, 皇马}, x) = 1.3 + 0.2 = 1.5$$

然后，我们套用 softmax 公式可以得到如下的概率化结果：

$$P(B|\text{马, 皇马}, x) = \frac{e^{0.1}}{e^{0.1} + e^{1.5}} \approx 0.197$$

$$P(I|\text{马, 皇马}, x) = \frac{e^{1.5}}{e^{0.1} + e^{1.5}} \approx 0.803$$

由此，我们大致可以得出结论，在“皇马对奥尼尔不感兴趣”这句话中，“马”字大概率应该被标为“l”。

由此，我们基本熟悉了 CRF 从构造特征、到模型建模、再到标签计算的全流程。

以上只是一个例子，如果从公式做严谨表达，那么建模公式就可以细化为：

$$\begin{aligned} P(y|x) &= \frac{\prod_{i=1}^L \exp(\sum_{k=1}^K w_k f_k(y_i, x))}{Z(x)} \\ &= \frac{\exp(\sum_{i=1}^L \sum_{k=1}^K w_k f_k(y_i, x))}{Z(x)} \end{aligned}$$

在上式中， L 表示文本中有多少个汉字，在例子中，长度为 10； K 表示每一个汉字对应了多少个特征，在例子中，“马”字仅匹配了两个特征； w_k 是模型的一整套特征参数，其下标在这里仅仅表示被匹配的那一部分， $f_k(y_i, x)$ 表示了那个特征匹配函数；而 $Z(x)$ 则是一个归一化因子，它是所有可能的标签序列的权重值的加和。

$$Z(x) = \sum_y \exp(\sum_{i=1}^L \sum_{k=1}^K w_k f_k(y_i, x))$$

CRF 的转移特征

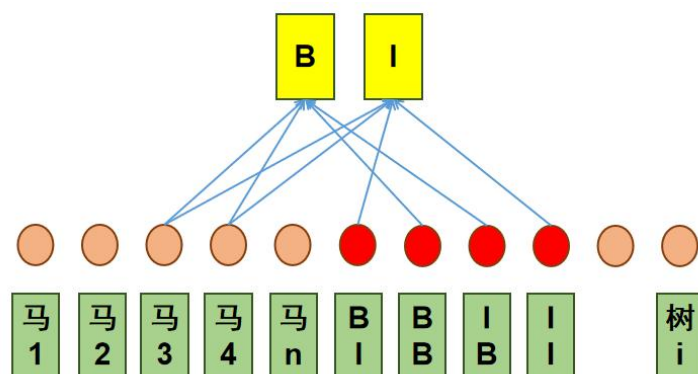
需要注意的是，我们的例子里，仅仅计算了“马”字对应的一些特征，就判断“马”字应当被标为“l”标签。那么，标签和标签直接似乎也应该有一些关

联关系吧。

例如，在词性标注任务中，往往词性之间是有一定的关联关系的，“美丽的海岛”，“神奇之国”，它一定是“形容词->助词->名词”这样的语序来描述事物，这属于汉语的一种固定语法表达结构，而词性本身就是一种输出标签。

为了描述上述标签之间的关系，我们需要再为 CRF 模型添加一类参数特征——**标签转移矩阵特征**。

虽然这类参数听起来似乎有点特殊，但实际上也非常简单。如下图



图中增加了四个特征，用红色圆圈表示，分别表示从 $y_{i-1} \rightarrow y_i$ 的标签转移。在例子中，它也就表示了“皇马”中，“皇”字的标签和“马”字的标签的关系。也就是，具有四种情形：

情形 1：“皇”字“B”标签，“马”字“I”标签；这种是正确的；

情形 2：“皇”字“B”标签，“马”字“B”标签；

情形 3：“皇”字“I”标签，“马”字“I”标签；

情形 4：“皇”字“I”标签，“马”字“B”标签。

在此种情形下，假设训练好的模型的转移矩阵为：

$$w_{y_{i-1}y_i} = \begin{pmatrix} 0.2 & 0.3 \\ 0.1 & 0.4 \end{pmatrix}$$

由上可知，我们计算得到了“马”字的各标签权重

$$w(B|马, 皇马, x) = 0.1$$

$$w(I|马, 皇马, x) = 1.5$$

现在假设我们也计算出了“皇”字的各标签权重

$$w(B|皇, 皇马, x) = 0.7$$

$$w(I|皇, 皇马, x) = -0.1$$

根据建模公式，我们的计算目标是为了使整条序列的标签最优化，那么我们就需要考虑各个标签，尽量使其更加符合特征的参数值。

$$w_{i-1}^T w_{y_{i-1}y_i} + w_i = \begin{pmatrix} 0.7 \\ -0.1 \end{pmatrix} \begin{pmatrix} 0.2 & 0.3 \\ 0.1 & 0.4 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 1.5 \end{pmatrix} = \begin{pmatrix} 0.23 \\ 1.67 \end{pmatrix}$$

矩阵计算是利用“皇”字的权重，来计算“马”字的权重。再将两者参数相加，就可以得到“马”字的最终标签权重应当如上式所示。

我们再仔细观察一下上式，它的本质就是在计算“马”字的特征时，又添加了一个参数而已，同样也是加法，与普通参数的特征值相加毫无区别。

由此，我们的建模公式又可以更近一步：

$$P(y|x) = \frac{\exp(\sum_{i=1}^L \sum_{k=1}^K w_k f_k(y_i, y_{i-1}, x))}{Z(x)}$$

其中，添加了转移矩阵，考虑了前后两标签的位置关系。

需要注意的是，我们添加相邻两位置的转移特征只能采用上述方式吗？其实不是，在实际的 NLP 任务中，有时候转移特征非常有用，直接决定了标签的所属，有时又比较冗余，可能不要这个参数也完全不影响标签的输出结果。之所以采用上述方式，还有一个重要的原因，那就是与 $Z(x)$ 有关。

CRF 的模型训练

有了非常类似于神经网络的特征参数，那么就需要进行模型的训练。现在假设，我们有 N 条样本数据，建模公式可以直接求导：

$$L(\theta) = \prod_{n=1}^N P(y|x) = \prod_{n=1}^N \frac{\exp(\sum_{i=1}^L \sum_{k=1}^K w_k f_k(y_i, y_{i-1}, x))}{Z(x)}$$

对其取对数：

$$\begin{aligned} \log(L(\theta)) &= \log\left(\prod_{n=1}^N P(y|x)\right) = \sum_{n=1}^N \log(P(y|x)) \\ &= \sum_{n=1}^N \left(\sum_{i=1}^L \sum_{k=1}^K w_k f_k(y_i, y_{i-1}, x) - \log(Z(x)) \right) \end{aligned}$$

对其求导：

$$\frac{\partial(\log(L(\theta)))}{\partial w_k} = \sum_{n=1}^N \left(\sum_{i=1}^L f_k(y_i, y_{i-1}, x) - \frac{1}{Z(x)} \frac{\partial Z(x)}{\partial w_k} \right)$$

其中，

$$\frac{\partial Z(x)}{\partial w_k} = \sum_y \exp(\sum_{n=1}^N \sum_{i=1}^L w_k f_k(y_i, y_{i-1}, x)) \sum_{i=1}^L f_k(y_i, y_{i-1}, x)$$

然后，

$$\frac{1}{Z(x)} \frac{\partial Z(x)}{\partial w_k} = \sum_y (p(y|x) \sum_{i=1}^L f_k(y_i, y_{i-1}, x))$$

那么，最终求导结果就是：

$$\frac{\partial(\log(L(\theta)))}{\partial w_k} = \sum_{n=1}^N (\sum_{i=1}^L f_k(y_i, y_{i-1}, x) - \sum_y (p(y|x) \sum_{i=1}^L f_k(y_i, y_{i-1}, x)))$$

相信上面的公式阅读起来是非常恶心人了，如果是做 NLP 任务，还是建议从头推导一遍公式；若是仅仅参阅理解一下，最终得到了这么一个求导后的导数结果，我们来观察一下导数的物理意义。

首先最外侧的 $\sum_{n=1}^N$ 符号标识了该求导是针对每一条样本而计算的，并将结果加和。直观的举例，在上述的“皇马对奥尼尔不感兴趣”这条样本中，出现了一次“皇马”的特征，现假设我们还有另外一条样本“欧洲球队里，皇马还不错”，这里再一次出现了“皇马”，此时在模型训练中，应当将这两次的样本的导数进行加和，导数加和，也就意味着，相同特征出现的次数越多，对该特征参数的学习就应该越激进。

式子里左侧是一个 $\sum_{i=1}^L f_k(y_i, y_{i-1}, x)$ ，它是指，对一条样本中的所有被匹配到的特征，要进行一个加和。例如，“皇马终究还是皇马，皇马永远都是皇马”，这个样本中出现了四次“皇马”，那么此时应该对“皇马”的特征参数做 4 倍的学习，使模型更快地收敛。

到此为止，我们目前的参数学习都是非常顺其自然的，首先在样本中匹配到一个特征，然后根据这个梯度值，向正确的y匹配的参数方向做一个参数权重学习。

接下里，导数右侧有 $-\sum_y (p(y|x) \sum_{i=1}^L f_k(y_i, y_{i-1}, x))$ 这样的式子，首先，我们可以直接认为， $f_k(y_i, y_{i-1}, x)$ 就是等于 1 的，原因如其定义。那么，这部分内容的含义是在前述基础上，对所有的匹配参数做一个逆向的变化，换句话说，这里是所有的输出标签序列的大集合，我们应当对这些被匹配到的标签，全部做一

个逆向的更新，使得下一次更加不易输出这样的结果。

还需要注意的是， \sum_y 是针对所有的样本而言，也包括了那一条正确的标注样本，理论上，这里其实不包括也是可以的。

综上所述，CRF 的求导公式，就是对匹配正确的特征参数进行加强，对未匹配的特征参数进行以概率作为权重的减弱。

CRF 的求导转化

问题来了， $Z(x)$ 是一个指数级计算难度的式子，想要遍历一遍这样的计算难于登天。

此时需要将上式中的指数级的概率计算，转换成一个线性时间的参数计算，这样才有计算的可行性。

在模型中，我们前述定义了一个转移特征参数，并且论述了其存在的物理意义，它仅仅描述了前后相邻的两个标签之间的关系。

当然，如果你要问，前后相隔一个的两标签 y_{i-1}, y_{i+1} 之间有没有关系呢？多少也是有的，只是这不利于转化和计算。

那么，利用前后相邻标签的转移，对右侧式子做一个变形：

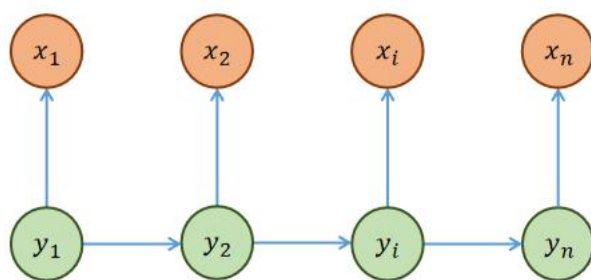
$$\sum_y (p(y|x) \sum_{i=1}^L f_k(y_i, y_{i-1}, x)) = \sum_{i=1}^L \sum_{y_{i-1}} \sum_{y_i} p(y_{i-1}, y_i | x) f_k(y_i, y_{i-1}, x)$$

为避免复杂的公式，这里具体讲一下变形左右两边的具体含义，左边是在每一条输出标签序列基础上，对其匹配到的特征都做一个特征相加计算，其权重也就是 $p(y|x)$ 。

右侧则将一个全局的标签输出序列，转化为了相邻两个标签才具有关联关系的概率模型，其中， $p(y_{i-1}, y_i | x)$ 标识了在所有的序列中，符合 y_{i-1}, y_i 的序列的概率。

这样的方法实际上是借用了 HMM（隐含马尔科夫链模型）算法的思想。基于这样的假设，右侧的物理含义就是成功地将指数计算，线性片段化了。

根据 HMM 算法所述，概率转移图如下图所示。



在这个图中，实际上是由隐藏状态 y_i 来决定了显式状态 x_i ，而隐藏状态也是层级转移得到的。用公式表达形式就是：

$$P(x_1, y_1, \dots, x_i, y_i, x_n, y_n) = P(y_1)P(x_1|y_1) \prod_{i=2}^n P(y_i|y_{i-1}) P(x_i|y_i)$$

其实可以看到，当我们想要计算最后一个节点的状态概率时，我们需要从第一个节点开始计算。通过一个线性复杂度的链式计算来得到最终的结果。

同样的，针对 CRF 算法，我们也可以采用这种方法进行计算，当然了，在 CRF 算法中，参数不是标准概率，而是对数化的概率，只需要处理这个对数化概率，就可以得到状态转移的计算方式，如前例子，“皇马”所示：

$$w_{i-1}^T w_{y_{i-1}y_i} + w_i = \begin{pmatrix} 0.7 \\ -0.1 \end{pmatrix} \begin{pmatrix} 0.2 & 0.3 \\ 0.1 & 0.4 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 1.5 \end{pmatrix} = \begin{pmatrix} 0.23 \\ 1.67 \end{pmatrix}$$

这就是通过前一个状态，经过状态转移而得到的结果。

我们得到了 CRF 算法的求导公式后，就可以迭代进行模型训练了。到此为止，模型介绍完毕，如果还是有不清楚和不明白的，可以详见 Github 项目 jiojio (<https://github.com/dongrixinyu/jiojio>)，里面有详细的模型实现代码。

CRF 的评价指标 F1 值

CRF 模型的评价指标主要采用 F1 值。即所谓查准率和查全率，这里就不列出公式，仅语言描述一下其含义。

以前我们参加劳动的时候会去农田里拔草，农田里有秧苗，还有杂草。马化疼和码云比赛在田里拔草。这个画面想想就很美有没有？

马化疼一个小时里，拔了 10 颗杂草，没有破坏秧苗，则他拔草的准确率，

即查准率是 100%，而码云把田里的 100 颗杂草全拔出干净了，但是他连带着拔了 20 棵秧苗，这说明了他拔草的召回率，即查全率为 100%。

其实我们最希望的是，又准又全，对应在分词等 CRF 任务里，就是希望模型能够把所有的标注语料里的词汇正确的输出出来。将查准率和查全率作平均值计算，得到的就是 F1 值。

F1 值越逼近 1，说明模型效果越好。

CRF 的模型与过拟合

对于一般的神经网络模型，很常见的一个问题就是过拟合。例如，对于一个输入样本，神经网络会调取网络中的所有参数 $w_1, w_2, \dots, w_i, \dots, w_n$ 来参与计算该样本。这是神经网络经常导致过拟合的根源，为了解决该问题，常见的方式就是采用 dropout 策略，其本质就是过滤掉一部分参数，仅仅使用模型中的随机的一小部分参数 w_1, w_2, \dots, w_i 来进行模型训练。

在 CRF 算法，假设我们的模型参数有百万级，而对于某一个汉字，例如“马”字时，实际上仅仅有极少的若干个特征与该字相关。其本质，也就是每一次模型的推断，都仅仅使用了整个模型参数的极小部分，这和 dropout 是非常类似的。

换句话说，CRF 模型天然抗过拟合。所以在训练 CRF 模型时，可以尽情训练，不必担心过拟合问题。

CRF 的模型与欠拟合

CRF 的模型与欠拟合之间的关系，非常好理解，例如处理“马”字时，实际上仅仅有极少的若干个特征与该字相关。这种情况是非常普遍的，在模型推断阶段，参与计算的特征参数在时间上总是非常稀疏的。

这和神经网络非常不同，因为神经网络的推断每次都动用了模型中所有参数进行计算，因此对于神经网络，十分有必要采用 GPU 等设备加速。

而 CRF 则无此必要，相反，CRF 的特征往往过于稀疏，导致特征拟合不足。

自然语言文本具有长尾效应，10%的词汇，例如“我们”、“工作”、“热爱”等等，组成了 50%的语言表达，而最稀疏的 30%的罕见词汇，例如“桎梏”、

“臻懿”、“糠醛”等等，仅仅构成了文本的约 2% 的语言表达。我们想要对这极其罕见的 2% 进行拟合何其困难，需要制造出巨量的参数来满足这 2% 的提升。

然而，考虑到实际的计算机内存，我们往往得缩小参数数量，来提高模型的可用性。这些牺牲掉的特征，就是 CRF 的欠拟合。

CRF 的模型与参数稀疏

在 NLP 的神经网络模型中，有 word2vec 这样的操作，它将每个词汇都用一个固定长度的向量进行表示，利用向量的余弦距离，可以把“遗骸”、“遗骨”这样的近义词用非常相近的词向量进行表达。这实际上是减少了特征表达的稀疏，增强了拟合能力。

但是这个操作对于最原始的 CRF 模型则是无效的。例如，中文中的词汇表达常用数量就在几十万以上，一个 CRF 模型想要满足这样的数量级，则需要巨量的模型参数，在模型中，“遗骸”和“遗骨”实际对应了不同的参数，这增加了模型的参数量，从某种角度上是冗余。

当然，这并不是说，CRF 就不能用词向量进行模型参数的精简。这种工作，经过神经网络一代代的更迭，最终体现在了经典的 Bi-LSTM-CRF 模型上。

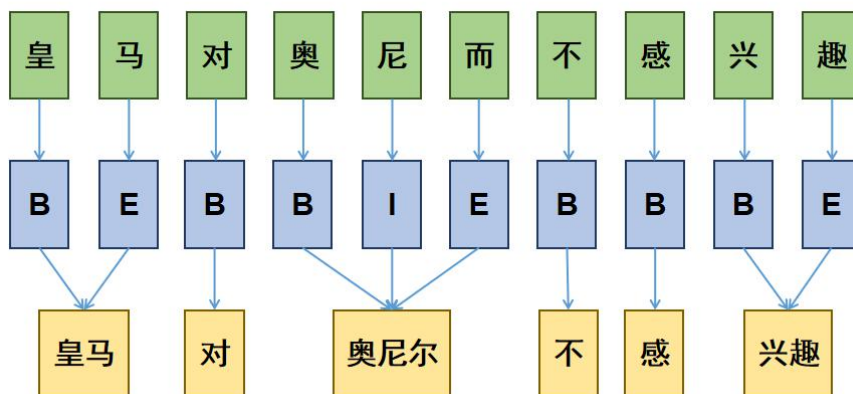
BTW，这里就需要再补充一下，如果说，采用 LSTM 模型来加入 CRF 模型上，叫做 Bi-LSTM-CRF 模型，那么最原始的 CRF 模型，其实可以被称作 Logistic-CRF，也即逻辑回归 CRF 模型。

CRF 的转移参数的必要性讨论

相比于其它的字、词特征，动辄数量上百万，CRF 中的转移参数往往数量只是 n^2 个，其中 n 是标签类别的数量，这样一个数量级对模型拟合结果的效用体现是非常小的。

若像上例中为分词标注 B 和 I 标签，则讲真，有无转移参数，对分词效果提升不大。

但是，除了上述 BI 分词标注标准，还有一些别的标准，比如 BIE 标准。其含义如图所示。



在图中，E 标签表示词汇的结尾，I 标签表示词汇的中部，B 标签表示词汇的起始。这样一来，就存在一种转移参数会趋近无限小，即 $E \rightarrow I$ ，原因在于，真正准确的分词模型不可能从 E 标签直接跳转到 I 标签，这样一来，转移参数就发挥了作用。

这也就是转移参数应当起到的作用。

Viterbi 算法

CRF 算法的常规推断手段就是 Viterbi 算法，它通过动态规划的方式，把指数级的最佳标签转移路径通过线性级的计算量找出来。

假设当前位置的各标签权重为 $\varphi_i(y_i)$ ，则经过一次转移，找出一条最佳的下一个标签权重为：

$$\varphi_{i+1}(y_{i+1}) = \max(\varphi_i(y_i) \cdot w_{y_i y_{i+1}})$$

每一次迭代都会获得一个当前位置的最佳标签序列。到最后一个位置迭代完，就会获得整条序列的最佳标签序列。

结合前述转移参数的效用，其实可以发现，Viterbi 算法的核心作用在于控制一些强依赖标签转移的特征和字符，即，设定 BIE 标签标准， $E \rightarrow I$ 的转移是违规的，这会由转移参数做控制，体现在具体的对数化的参数中，该参数将会是一个参数极小值。

除此之外，转移参数往往不具有显著效果。因此，为了算法模型推断加速，当我们使用 BI 标签标准时，把 Viterbi 算法忽略掉其实也是可以的，这当然会降低一些 F1 值效果，但减少的量级在万分位，这就非常划算了，那么我们可以直

接对标签权重求取结果了， $\operatorname{argmax}(\varphi_i(y_i))$ 。

如果我们使用 BERT-CRF 等类似的拟合能力非常强的算法模型来做一个分词模型，那么，CRF 转移参数层就更加可有可无了，原因在于 BERT 已经可以把结果标签拟合得非常好了，不需要 CRF 转移参数再去纠偏标签的错误输出。

CRF 与分词

CRF 算法与分词任务可以说是非常契合的，在深度神经网络模型席卷各个常用 NLP 任务榜单以前，分词任务可以说在理论上可以利用 CRF 算法取得非常好的效果。本篇文章也是在利用分词任务作为例子进行介绍。

之所以可以取得效果，在中文领域，词汇绝大多数集中在 1 字词，例如“我”，2 字词，例如“科技”，三字词，例如“民政部”，四字词，例如“情不自禁”。4 字长度以下的词汇数量占比达到了所有词汇总量的大约 97%。至于 5 个字长的，则往往已经成了句子，需要进一步切分了，例如“宫廷玉液酒，一百八一杯”，切分成“宫廷”、“玉液酒”、“一百八”、“一杯”，是一种相对较恰当的切分方式。在这种较短的文字前后上下文联系上，就非常契合了。反之，对前后文的依赖越长，则效果越差。

然而，分词任务实际上依然非常依赖词典和规则。这里触及到一个 NLP 领域里一个非常核心的问题——**语义和认知**。现在社会每年都会出一些新兴词汇，例如“新常态”、“躺平”、“内卷”、“YYDS”等等。实际上这些词汇在 CRF 模型训练之后出现，是无法解决的，因为模型从来没见过“内卷”这个词汇，自然也就无法正确切分。

但是人是能明白的，因为人能从大量的社会经验、阅读、学习过程当中，自然而然地掌握了“内卷”这个词汇。但是 CRF 模型是不行的。不仅仅是 CRF 模型，几乎所有的深度学习模型，诸如 BERT、GPT、T5 等等都无法真正解决这个问题。越精进的模型，它可以获得逼近 99.999% 的 F1 值，但是随着社会的发展，语言的演进，模型的效果会逐渐降低。因为所有的这些都是**统计概率模型**，也就是一种基于数据经验的模型，它们全都不具有**推理、不确定性判断**的能力。从这个角度讲，当前我们能接触到所有模型，都是处于同一代人工智能模型，也就是常说的第二代人工智能模型。

因此，这部分功能一定需要添加一个**新词发现**的功能，能够自觉地将语料中

的一些新词固化下来，作为词典辅助 CRF 模型使用。

分词任务，从技术角度来讲，还是运用模型处理得到的效果较为理想；但是从实际应用角度来讲，它常常被当作一种预处理手段，例如，在将文本输入大型神经网络之前，或者抽取信息之前，首先做分词处理，以得到清晰的语义。

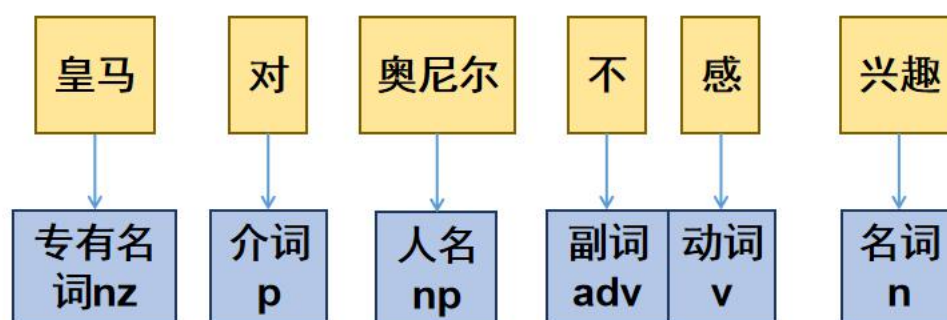
这就造成了一种偏差和拉扯，NLP 开发者往往会感到，常用分词根据速度很慢，或者速度快的词典匹配效果又不理想。

为了解决我自己 coding 过程中的分词不爽症，我开发了一个分词工具 jiojio，Github 链接为（<https://github.com/dongrixinyu/jiojio>），该工具提供了持续的分词模型的数据优化和更新，通过在线提交分词标注数据，就可以为开源模型做数据贡献，模型会定期更新，确保对新数据的敏感性；该工具基于 CPU 和 C 语言加速，增强了分词的计算效率；而且采用 Python 作为调用语言，极大的方便了工具的推广和使用。

在此做一个推广。

CRF 与词性标注

词性标注也就是对一段文本中的每一个词汇赋予一个词性，例如下图



CRF 也可套用在词性标注上，在一般古早的（大约 2006 年前后）一些数据集上，词性标注在 CRF 上取得的 F1 值一般都明显低于分词。其中两个很重要的原因：

一是相对于汉字的数量，中文词汇的数量是非常庞大的，动辄几十万起步，想要在这么庞大的词汇规模上若生成特征，参数量是分词模型的很多倍，导致了巨量的稀疏参数，这个量级很大程度上影响了模型的训练和使用。

二是词性标注实际上已经在某些程度上属于语义深层的任务，例如，“信息

通信”，可以标注为两个名词，如“信息通信技术”，也可以标注为名词+动词，如“信息通信时”。也就是说，词性标注的上下文语义影响更长。

从理论上讲，CRF 当然可以克服上述障碍，毕竟模型可以自定义特征匹配。但是从实际上来说，CRF 模型在处理中文词性标注任务上，受限于资源和训练的数据量，F1 值基本上维持在 90% 上下。

这是一个不好不坏的结果，是一个折中于资源和效果的方案。目前词性标注也在 jiojio 分词工具包中。

当然，序列标注的任务，如前所述，非常多，诸如**语义槽填充**、**要素识别**等等，这些任务都有一个共同点，它们的语义联系非常强，语义上下文依赖也非常强，这些已经不适合最原始的 Logistic-CRF 模型来处理了，需要更强和更大的模型来处理。

尾声

写作这篇文章时间已经是 2022 年了，在自然语言处理（NLP）领域，各种巨量参数超级模型，text-2-text 的模型范式被提出来，巨量的文本被用于预训练。当然，从某种角度，文本真正的语义理解还距离很远。这种遥远体现在 NLP 研究者在除了大模型和贴近文本表达范式之外，还没有找出一条新的语义理解解决路径；也体现在 NLP 工程师还是经常陷于采用规则和显式的代码来解决文本处理中遇到的问题，而无法真正教会模型来处理某些问题。

而从这种角度，这些模型都还属于**统计自然语言处理**，而非智能自然语言处理。所以，CRF 作为一个经典的解决序列标注建模问题的算法，依然具有很高的学习价值。故在此重新详述一遍。