注：本文重度参考文末参考文献[1]！

深度学习在中文分词和词性标注中的应用

peghoty　整理

peghoty@163.com

目　录

本文主要探讨利用深度学习 (DL, Deep Learning) 来做中文分词 (CWS, Chinese word segmentation) 和词性标注 (POS tagging) 的可行性. 我们采用深层神经网络来发现与任务相关的特征, 从而避免依赖于具体任务的特征设计 (task-specific feature engineering). 我们还利用大规模非标注数据 (unlabeled data) 来改善中文字的内在表示 (internal representation), 然后使用这些改善后的表示 (representation) 来提高有监督的分词和词性标注模型的性能. 我们提出的方法在性能上接近于当前最好的 (state-of-the-art) 算法, 但计算开销更小. 作为 maximum-likelihood 方法的一个替代, 文中提出的 perceptron-style 算法不仅可以加速训练过程, 而且实现起来也更容易.

§1 引言

分词一直是中文 NLP 社区的一个挑战, 在过去的二十多年中得到了人们的持续关注. 之前的研究表明: 与流水线系统 (pipelined system) 相比, 联合解决方案 (joint solutions) 通过利用 POS 信息帮助分词以及避免误差传播, 通常可以提高精度. 然而, 传统的联合多任务方法涉及到大量的特征, 从而导致以下四个问题:

1. 得到的模型太大, 内存和计算开销使其不能用于实际问题;
2. 参数太多, 使得训练得到的模型在训练数据上易于过拟合 (overfit);
3. 训练时间长;
4. 由于搜索空间过大, 通过动态规划技术来解码很难.

注 1.1 *这里提到的 pipelined system 和 joint solutions 分别指单任务和多任务, 前者每次只考虑一个任务, 而后者同时考虑多个任务, 如同时执行中文分词和词性标注任务.*

这些系统成功的一个关键因素是特征的选取. 大部分先进 (state-of-the-art) 系统都是首先针对任务精心优化得到特征, 然后在其上应用线性统计模型. 这种方法很有效, 原因是研究人员可以往模型里融入大量的语言知识 (linguistic knowledge). 但是, 这种方法不能很好地扩展到更加复杂的 joint 任务, 如分词、POS 标注、解析和语义角色标注的 joint 任务. 这样的 joint 模型的挑战在于合并后的大搜索空间使得与任务相关特征的设计以及参数的学习变得非常困难. 在本文中, 我们将采用多层神经网络从输入的句子中发现有用的特征.

本文的主要贡献在于:

(1) 给出一种训练神经网络的 perceptron-style 算法, 在几乎不损失性能的情况下, 它不仅可以加速训练过程, 且更容易实现;

(2) 证明了 Deep Learning 可以有效地用于中文分词和词性标注这样的 NLP 任务. 通过将在大规模非标注数据上无监督训练得到的字向量作为我们监督算法的初始值, 可以进一步改善算法的性能.

第二节介绍神经网络的一般构架以及我们的训练算法; 第三节介绍如何利用大规模非标注数据来获取更有用的字表示 (character embeddings), 并给出实验结果; 第四节对相关工作做一个简要介绍; 第五节总结全文.

§2 神经网络构架

中文分词和词性标注的本质是对输入句子中的字进行标注. 传统标注方法的性能强烈地依赖于特征的选取, 例如, 条件随机场 (CRFs) 就需要一组特征模板 (feature templates). 正因为此, 系统设计的重点就转移到特征的设计上去了, 特征设计虽然重要但耗费人力 (labor-intensive), 而且主要基于人类的聪明才智 (human ingenuity) 和语言的直觉 (linguistic intuition).

2003 年, Bengio 等人建立概率语言模型时提出了一种经典的神经网络 ([2]), 2011 年 Collobert 等人基于这个神经网络提出了新的方法 ([3]). 为了减少学习算法对特征设计的依赖性, 本文使用这个神经网络的一个变种, 它将句子作为输入, 对其做多层的特征提取, 从而得到更抽象的特征表示. 图 1 给出了这种神经网络的构架.

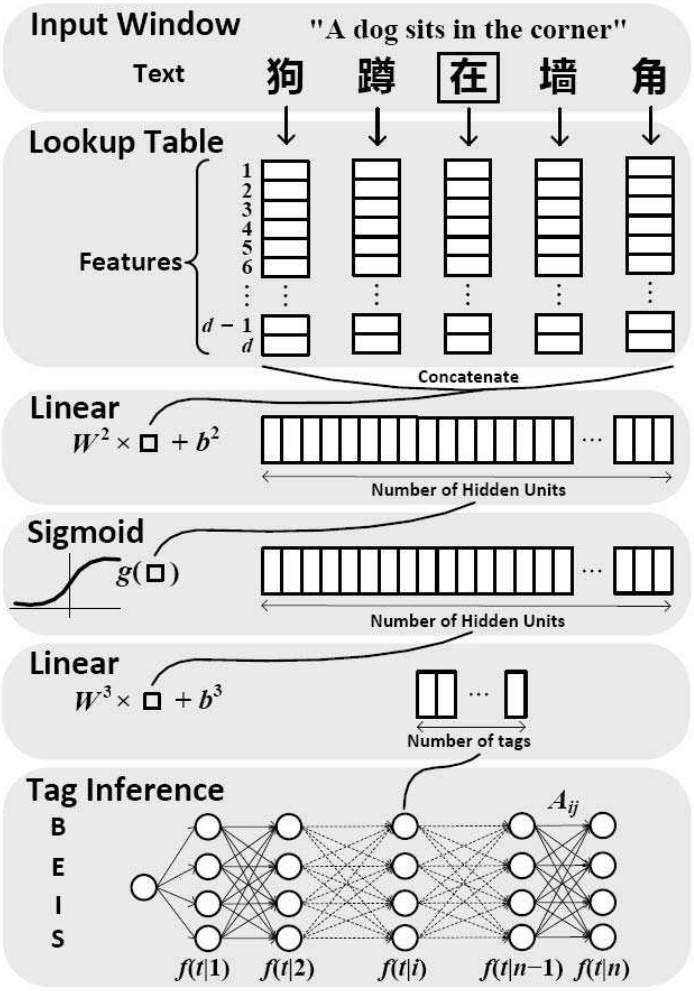

图 1 The neural network architecture

网络的第一层得到输入句子中每个字的特征; 第二层通过将一个固定长度的字序列的特征首尾相连得到一个新的特征; 接下来的层为标准的神经网络层; 网络的输出为一个图 (graph), 利用 Viterbi 算法可以完成标注推断 (tag inference).

§2.1 Mapping Characters into Feature Vectors

字以索引指标的方式被送入网络, 这些指标通过一个查询操作可转换为其特征向量. 考虑一个固定大小的字典 $\mathcal{D}$, 字的向量表示存放在矩阵 $\mathcal{M} \in \mathbb{R}^{d\times|\mathcal{D}|}$ 中, 其中 d 为特征向量空间的维度 (为可调参数), $|\mathcal{D}|$ 为字典的大小.

注 2.1 若不特别说明, 本文提到的字典均是从训练集得到的. 未知的字将被映射到一个未使用过的特殊符号.

给定句子 $c_{[1:n]}$, 它包含 n 个字 c_i, $1 \le i \le n$. 其中 c_i 的特征向量为矩阵 $\mathcal{M}$ 的第 k_i 列, 数学上可表示为

$$\mathcal{Z}_{\mathcal{D}}(c_i) = \mathcal{M}e_{k_i} \in \mathbb{R}^d, \tag{2.1}$$

其中 $e_{k_i} \in \mathbb{R}^{|\mathcal{D}|}$ 为 $\mathbb{R}^{|\mathcal{D}|}$ 中的第 k_i 个单位向量 (除第 k_i 个分量为 1 外其他分量为 0). $\mathcal{Z}_{\mathcal{D}}$ 对应一个查询表层 (lookup table layer), 也可视为一个简单的投影层 (projection layer). 字的特征向量 (即矩阵 $\mathcal{M}$) 首先随机初始化, 然后通过 BP 来自动训练, 使其变得与任务相关.

在实际应用中, 我们还经常会引入一些有用的额外特征, 例如, 进行命名实体识别 (name entity recognition) 任务时, 可以提供一个这样的特征: 指明一个字是否为常见中国姓氏. 另外, 也可以引入一些基于统计的量, 例如 boundary entropy, accessor variety 等就常用于非监督型的中文分词模型. 对于新添加的特征, 我们也为其关联相应的查询表, 最后, 词特征向量就变成所有查询表相应特征列的联接了.

注 2.2 *关于 boundary entropy 和 accessor variety*

1948 年, **香农**提出了 "信息熵" 的概念. 在信息论中, 信息熵可以理解为特定信息的出现概率, 若某个特定词串边界处的 "信息熵" 越高, 那么这个词串可与其他词组合的不确定性也就越高, 这个词串也就越有可能是一个短语. 在文本处理中, 称之为 **"边界熵"** (Boundary entropy, BE). 边界熵在分词系统中有很好的应用, 其定义为

$$BE(w_1w_2\cdots w_k) = -\sum_{w\in C} p(w|w_1w_2\cdots w_k)\log p(w|w_1w_2\cdots w_k), \tag{2.2}$$

上面的公式中 w 是一个中文汉字, $p(w|w_1w_2\cdots w_k)$ 是 w 邻接于汉字串 $w_1w_2\cdots w_k$ 的概率. 因此, 有两种边界熵的定义, 左边界熵 BEl 和右边界熵 BEr.

边界多样性 (Accessor veriety, AV) 非常类似于边界熵, 都以边界的不确定性作为衡量短语好坏的标准. 其定义为

$$AV(w_1w_2\cdots w_k) = \log RL_{av}(w_1w_2\cdots w_k), \tag{2.3}$$

其中 $RL_{av}(w_1w_2\cdots w_k)$ 表示形式上可能的邻接于汉字串 $w_1w_2\cdots w_k$ 的不同的汉字的个数. 同样地, 我们可以定义左边界多样性 AVl 和右边界多样性 AVr.

§2.2 Tag Scoring

一个神经网络可以看做是以 θ 为参数的函数 $f_\theta(\cdot)$, 对于一个 L 层前馈神经网络, f_θ 可以写成以下复合函数的形式

$$f_\theta(\cdot) = f_\theta^L(f_\theta^{L-1}(\cdots f_\theta^1(\cdot)\cdots)). \tag{2.4}$$

对于输入句子中的每一个字, 我们的神经网络架构将为其可能的每一个 tag 进行评分. 为解决不同句子对应的字序列长短不一的问题, 我们采用**窗口方法** (window approach), 窗口方法假定一个字的 tag 主要依赖于与其相邻的字. 更确切地说, 给定一个输入句子 $c_{[1:n]}$, 考虑其所有大小为 w 的窗口 (窗口从 c_1 开始一直滑到 c_n). 对于字 c_i, 得到的向量可表示为

$$f_\theta^1(c_i) = \begin{bmatrix} \mathcal{Z}_\mathcal{D}(c_{i-\frac{w}{2}}) \\ \vdots \\ \mathcal{Z}_\mathcal{D}(c_i) \\ \vdots \\ \mathcal{Z}_\mathcal{D}(c_{i+\frac{w}{2}}) \end{bmatrix}. \tag{2.5}$$

越界下标对应的字将被映射成两个特殊的符号, 分别对应 start 和 stop.

注 2.3 窗口以字 c_i 为中心, 左右两侧取相同的字数, 总长度为 w, 因此这里的 w 应该为奇数. (2.5) 中的 $\frac{w}{2}$ 作了向下取整 (类似于程序语言中的表达).

得到定长的 $f_\theta^1 \in \mathbb{R}^{wd}$ 之后, 将其送入两个标准的线性层, 其中还包含一个非线性的激活函数 g 用来提取高度非线性特征. 设 $\mathcal{T}$ 为所考虑问题的标注集, 则经过这两个线性层后, 得到的向量可以表示为

$$f_\theta(c_i) = f_\theta^3(g(f_\theta^2(f_\theta^1(c_i)))) \ = W^3 g(W^2 f_\theta^1(c_i) + b^2) + b^3 \in \mathbb{R}^{|\mathcal{T}|}, \tag{2.6}$$

其中, **权值矩阵** $W^2 \in \mathbb{R}^{H\times wd}$, $W^3 \in \mathbb{R}^{|\mathcal{T}|\times H}$, 以及**偏置向量** $b^2 \in \mathbf{R}^H$, $b^3 \in \mathbf{R}^{|\mathcal{T}|}$ 均为需要训练的参数, H 表示隐藏层的隐单元个数 (为可调参数), 激活函数 g 选用 sigmoid 函数

$$g(x) = \frac{1}{1+e^{-x}}. \tag{2.7}$$

$f_\theta(c_i)$ 的第 j 个分量表示 c_i 的标注为 $\mathcal{T}$ 中第 j 个元素的得分 (score).

注 2.4 文 [3] 中用的激活函数是 HardTanh 函数, 其定义为

$$HardTanh(x) \ = \begin{cases} -1, & x < -1; \\ x, & -1 \le x \le 1; \\ 1, & x > 1. \end{cases} \tag{2.8}$$

在我们的实验中, 发现 sigmoid 函数比 HardTanh 函数表现要好一点点.

§2.3 Tag Inference

对于分词和词性标注这样的 NLP 任务来说, 一个句子中各个字的 tag 之间存在很强的依赖性. 当连续两个字的 tag 由 $\mathcal{T}$ 中的第 i 个变为第 j 个时, 我们引入相应的转换分数 (transition score) A_{ij}, 特别地, 还引入 A_{0i} 表示句首第一个字的 tag 为 $\mathcal{T}$ 中的第 i 个时的转换分数. 我们希望有效的 tag path 得到激励, 而其他 tag path 得到抑制.

给我们的神经网络输入一个句子 $c_{[1:n]}$, 将得到一个评分矩阵 $f_\theta(c_{[1:n]})$. 记 $f_\theta(t|i)$ 表示输入句子中第 i 个字 c_i 的 tag 为 $\mathcal{T}$ 中的第 t 个时的分数. 给定一条 tag path $t_{[1:n]}$, 则其相应的得分可定义为

$$s(c_{[1:n]}, t_{[1:n]}, \theta) = \sum_{i=1}^{n} (A_{t_{i-1}t_i} + f_\theta(t_i|i)). \tag{2.9}$$

为更好地理解公式 (2.9), 我们给各简单的例子.

例 2.1 设标注集 $\mathcal{T} = \{S, B, I, E\}$, 考虑句子“我是门外汉”, 其标注为

$$\text{我}/S \quad \text{是}/S \quad \text{门}/B \quad \text{外}/I \quad \text{汉}/E$$

则有 $c_{[1:5]} = \{\text{我}, \text{是}, \text{门}, \text{外}, \text{汉}\}$, $t_{[1:5]} = \{1, 1, 2, 3, 4\}$, 并令 $t_0 = 0$.

按照公式 (2.9), 例句的得分为

$$\begin{aligned} s(c_{[1:5]}, t_{[1:5]}, \theta) \quad = \quad & (A_{01} + f_\theta(1|1)) + (A_{11} + f_\theta(1|2)) + \\ & (A_{12} + f_\theta(2|3)) + (A_{23} + f_\theta(3|4)) + (A_{34} + f_\theta(4|5)) \,. \end{aligned}$$

注意, $f_\theta(c_i)$ 为一个长度为 $|\mathcal{T}|$ 的列向量, 若记 $B = (B_{ij}) \in \mathbb{R}^{|\mathcal{T}| \times n}$ 表示将 $f_\theta(c_1)$, $f_\theta(c_2)$, $\cdots$, $f_\theta(c_n)$ 从左至右拼成的矩阵, 亦即上文提到的 $f_\theta(c_{[1:n]})$, 则有 $f_\theta(i|j) = B_{ij}$.

对于给定的输入 $c_{[1:n]}$, 通过最大化由 (2.9) 定义的得分公式, 就可以找到一条最优 tag path $t^*_{[1:n]}$, 即

$$t^*_{[1:n]} = \arg\max_{\tilde{t}_{[1:n]}} s(c_{[1:n]}, \tilde{t}_{[1:n]}, \theta), \tag{2.10}$$

(2.10) 的求解可以利用 **Viterbi 算法**.

注 2.5 由排列组合基本公式可知, 这条最优 tag path 是从 $|\mathcal{T}|^n$ 条 path 中挑出来的. 因此, 句子越长 (即 n 越大), 则搜索空间越大.

接下来, 我们介绍如何使用端对端 (end-to-end) 的方式来训练参数.

§2.4 Training

训练过程将利用训练数据确定神经网络中的参数 $\theta = (\mathcal{M}, W^2, b^2, W^3, b^3, A)$. 通常采用的训练方法是最大化如下 log-likelihood

$$\sum_{\forall\ (c,t)\in\mathcal{R}} \log p(t|c,\theta), \tag{2.11}$$

其中 c 表示一个句子及其相关的特征, t 表示相应的 tag 序列. 为简化记号, 我们不再使用下标 $[1:n]$. 概率 $p(\cdot)$ 通过网络的输出计算得到, 具体计算方法在 §2.4.1 中再介绍.

(2.11) 的最大化可以利用**随机梯度下降法**, 它是一个反复迭代的过程, 每轮迭代依次从训练数据中随机选取一个样本 (c,t), 并执行一次修正

$$\theta \leftarrow \theta + \lambda \frac{\partial \log p(t|c,\theta)}{\partial \theta}, \tag{2.12}$$

其中 λ 为学习率 (为可调参数). (2.12) 中的梯度可以通过 BP 算法来计算.

注 2.6 文 [3] 采用的是随机梯度下降法, 但我们采用梯度下降法. 梯度下降法计算成本函数 (cost function) 的梯度时需要用到所有样本, 因为本文的训练集是有限的, 因此这不会引起问题.

§2.4.1 Sentence-Level Log-Likelihood

由 (2.9) 定义的得分可以通过以下方式转为条件概率:

$$p(t|c,\theta) = \frac{e^{s(c,t,\theta)}}{\sum\limits_{\tilde{t}} e^{s(c,\tilde{t},\theta)}} \in (0,1), \tag{2.13}$$

两边再取 log (以 e 为底), 可得

$$\log p(t|c,\theta) = s(c,t,\theta) - \log \sum_{\tilde{t}} e^{s(c,\tilde{t},\theta)}. \tag{2.14}$$

(2.14) 中求和号 $\sum$ 中的项数 $(= |\mathcal{T}|^n)$ 随着输入句子的长度呈指数增长. 计算 $\log p(t|c,\theta)$ 及其关于 $f_\theta(t|i)$, A_{ij} 的偏导数的计算开销非常大.

注 2.7 给定参数 θ 后, 句子 $c_{[1:n]}$ 的标注有 $|\mathcal{T}|^n$ 条可能的 path, 对于 path t, 设其得分为 $s(c,t,\theta)$, 为保证其值为正, 将它映射成 $e^{s(c,t,\theta)}$, 显然, path t 对应的概率就应该是 $e^{s(c,t,\theta)}$ 在 $\sum\limits_{\tilde{t}} e^{s(c,\tilde{t},\theta)}$ 中所占的份额, 这就是公式 (2.13) 的涵义.

下一节将介绍我们的训练算法, 它求梯度的计算开销要小得多.

§2.5 A New Training Method

与最大似然方法 (maximum-likelihood method) 不同, 我们受到文 [4] 的启发, 提出以下一种新的训练方法.

给定一个训练样本 (c,t), 我们的神经网络基于当前参数 θ 输出一个矩阵 $f_\theta(c)$, 进一步, 利用 Viterbi 算法可以找到句子 c 的一个得分最高的 tag 序列, 将其记为 t'. 对比 t 和 t', 作以下运算:

1. 若 $t_i \neq t'_i$, 则令

$$\frac{\partial L_\theta(t,t'|c)}{\partial f_\theta(t_i|i)}++; \quad \frac{\partial L_\theta(t,t'|c)}{\partial f_\theta(t'_i|i)}--. \tag{2.15}$$

2. 若 $t_{i-1} \neq t'_{i-1}$ 或者 $t_i \neq t'_i$, 则令

$$\frac{\partial L_\theta(t,t'|c)}{\partial A_{t_{i-1}t_i}}++; \quad \frac{\partial L_\theta(t,t'|c)}{\partial A_{t'_{i-1}t'_i}}--. \tag{2.16}$$

这里的 $++, --$ 分别表示自加和自减操作 (和 C 语言中的自加自减操作时一致的). $L_\theta(t,t'|c)$ 是我们想要在所有训练样本 (c,t) 上最大化的函数, 它可以视为 correct path 的得分与 incorrect path (网络在当前参数下得到的具有最高得分的 path) 的得分之间的差异.

注 2.8 关于 A_{0t} 是如何处理的呢? 上述运算中第二条 (2.16), 应该包含了以下情形: 当 $t_1 \neq t'_1$ 时, 做如下运算

$$\frac{\partial L_\theta(t,t'|c)}{\partial A_{0t_1}}++; \quad \frac{\partial L_\theta(t,t'|c)}{\partial A_{0t'_1}}--. \tag{2.17}$$

例 2.2 考虑句子“狗蹲在墙角”, 其正确标注序列为

狗/S 蹲/S 在/S 墙/B 角/E

在当前参数下, 最高得分对应的标注序列为

狗/S 蹲/B 在/E 墙/B 角/E

在本例中, $t = t_{[1:5]} = [SSSBE]$, $t' = t'_{[1:5]} = [SBEBE]$. 按照 (2.15) 和 (2.16) 给出的规则, 我们将所有运算记录如下

1. 当 $i=2$ 时, 由于 $t_2 \neq t'_2$, 执行

$$\frac{\partial L_\theta}{\partial f_\theta(S|蹲)}++, \quad \frac{\partial L_\theta}{\partial f_\theta(B|蹲)}--; \quad \frac{\partial L_\theta}{\partial A_{SS}}++, \quad \frac{\partial L_\theta}{\partial A_{SB}}--.$$

2. 当 $i=3$ 时, 由于 $t_3 \neq t'_3$, 执行

$$\frac{\partial L_\theta}{\partial f_\theta(S|在)}++, \quad \frac{\partial L_\theta}{\partial f_\theta(E|在)}--; \quad \frac{\partial L_\theta}{\partial A_{SS}}++, \quad \frac{\partial L_\theta}{\partial A_{BE}}--.$$

3. 当 $i=4$ 时, 由于 $t_3 \neq t'_3$, 执行

$$\frac{\partial L_\theta}{\partial A_{SB}}++, \quad \frac{\partial L_\theta}{\partial A_{EB}}--.$$

综上, 若这些偏导数初始均为零, 则经过这一轮运算后, 结果如下

$$\begin{aligned}
\frac{\partial L_\theta}{\partial A_{SS}} &= 2, \\
\frac{\partial L_\theta}{\partial f_\theta(S|蹲)} = \frac{\partial L_\theta}{\partial f_\theta(S|在)} &= 1, \\
\frac{\partial L_\theta}{\partial A_{SB}} &= 0, \\
\frac{\partial L_\theta}{\partial f_\theta(B|蹲)} = \frac{\partial L_\theta}{\partial f_\theta(E|在)} = \frac{\partial L_\theta}{\partial A_{BE}} &= \frac{\partial L_\theta}{\partial A_{EB}} = -1.
\end{aligned}$$

直观地看, 这些自加自减操作具有更新参数值的作用, 在某种意义上可以增大正确 tag 序列的得分, 且减小不正确 tag 序列 (网络基于当前参数的输出) 的得分. 极端情况是, 若网络生成的 tag 序列是正确的, 则参数不会发生改变.

接下来, 我们给出一个训练算法的完整描述.

- **输入**

  1. $\mathcal{R}$: 训练集
  2. N: 最大迭代次数
  3. E: 一个指定的 tagging 精度

- **输出**: 训练后的参数集 θ

- **算法描述**

算法 2.1 (训练算法)

Step 1 初始步

1.1 将参数 $\theta = (\mathcal{M}, W^2, b^2, W^3, b^3, A)$ 初始化为**小随机值** *(small random values).*

1.2 令 $k=0$ *(*用来记录迭代次数的变量*).*

Step 2 迭代步

2.1 利用所有样本, 对参数作一次更新, 具体步骤为

FOR $\forall\ (c,t) \in \mathcal{R}$

{

(1) 利用当前参数 θ 以及公式 *(2.6)*, 生成网络的输出矩阵 $f_\theta(c)$.

(2) 利用 $f_\theta(c)$ 及 A, 并调用 *Viterbi* 算法获得具有最高得分的 *tag path* t'.

(3) 对比 t' 和 t, 若 $t \neq t'$, 则做以下三步:

(3.1) 按照公式 *(2.15)* 和 *(2.16)* 计算梯度 $\frac{\partial L_\theta}{\partial f_\theta(c)}$ 和 $\frac{\partial L_\theta}{\partial A}$.

(3.2) 利用 $\frac{\partial L_\theta}{\partial f_\theta(c)}$ 及链式法则, 计算梯度 $\frac{\partial L_\theta}{\partial \mathcal{M}}$, $\frac{\partial L_\theta}{\partial b^3}$, $\frac{\partial L_\theta}{\partial W^2}$, $\frac{\partial L_\theta}{\partial b^2}$, $\frac{\partial L_\theta}{\partial W^3}$.

(3.3) 刷新参数: $\theta := \theta + \lambda \frac{\partial L_\theta}{\partial \theta}$.

}

2.2 令 $k := k + 1$ *(累加迭代次数)*.

2.3 *(终止条件判断)* 若已到达精度 E 或者 $k > N$, 则返回参数集 θ, 算法结束; 否则, 转至步 *2.1*, 继续迭代.

由于篇幅限制, 我们没有给出关于上述训练算法的收敛性理论, 但它可以通过结合文 [4] 中的收敛理论和 BP 算法的收敛结果得到.

§3 实验

- 考察算法
  1. SLL: §2.4.1 中介绍的基于 Sentence-Level Log-Likelihood 的训练算法.
  2. PSA: 本文提出的 Perceptron-Style 训练算法.
- NLP 任务
  1. CWS (实验报告中简记为 SEG)
  2. CWS + POS (实验报告中简记为 JWP)
- 实验方案
  1. 第一组实验

     用来测试各个可调参数 (hyper-parameters) 对算法性能的影响.

     实验数据采用 Chinese Treebank 4 (CTB-4) 的部分数据. 从中随机抽取 1529 个句子作为 training set, 其余的 168 个句子作为 development set.

     具体来说, 实验数据来自 treebank 的 Sections 1-43, 144-169, 900-931, 其中包含 78023 个字, 45135 个词和 1697 个句子. 这些文件是双重标注 (double-annotaed) 的, 是很好的标准文件.
  2. 第二组实验

     实验数据采用 Chinese Treebank (CTB) data sets from Bakeoff-3. 其中包含用于有监督分词和词性标注任务的训练和测试语料.

     实验中没有使用任何额外的知识 (即 the closed test), 实验结果与文献中的其他模型的结果具有可比性.
  3. 第三组实验

     研究大规模非标注数据如何用来改善 PSA 的性能.

     我们采用了文 [3] 的方法, 即首先利用大规模非标注数据来获取更包含更丰富语法和语义信息的字向量 (character embeddingss), 然后利用这些字向量来初始化矩阵 $\mathcal{M}$ (来替代算法 2.1 中对 $\mathcal{M}$ 的简单随机初始化).

     测试的语料来自 Sina news, 它包含 325MB 的数据.
- 测试环境
  - 算法代码: 用 Java 实现的 SLL 和 PSA 两个算法.
  - 机器配置和运行环境: Intel Core i3, 2.13GHz, 2GB RAM; Linux + Java Development Kit 1.6.
- 性能评判标准: 标准的 F-score (即精度和召回率的调和平均值).

§3.1 Tagging Scheme

对于中分分词, 我们为句子中的每一个字标注如下四种 tag 之一:

1. B: (Beginning) 词的第一个字.
2. I: (Inside) 词的中间字.
3. E: (Endding) 词的最后一个字.
4. S: (Single) 单字词 (即由单个字构成的词).

对于联合中文分词和词性标注的任务, 我们采用了文 [5] 中的标注方式: 即扩展字的标注, 使其包含词性标注信息. 例如, 我们使用如下四种 tag 来描述一个动词短语 (verb phrase):

1. B_VP: 动词词组的第一个字.
2. I_VP: 动词词组的中间字.
3. E_VP: 动词词组的最后一个字.
4. S_VP: 只包含一个字的动词词组.

事实上, 这里我们采用的是 "IOBES" (inside, other, beginning, end, and single) 标注格式, 只是 "O" 不适合于中文的分词和词性标注任务.

§3.2 The Choice of Hyper-parameters

本小节重点考察两个可调参数: 滑动窗口大小 (window size) w 和隐单元个数 H. 采用的数据为上小节介绍的 development set. 由于算法对参数采用的是随机初始化, 因此每一种参数配置下的方案都执行 5 次, 然后取 F-score 的**平均值**. 此外, 我们将分词 (word segmentation), 未登录词 (out-of-vocabulary) 和词性标注 (POS tagging) 对应的 F-score 分别记为 F_{word}, F_{oov}, F_{pos}.

图 2 给出了平均 F-score 随着参数 w 的变化曲线图. 由图可见:

当 $w > 3$ 时, 性能平滑地下降 (drop smmothly). 特别对于 F_{oov}, 当 $w > 5$ 时, 性能下降相对更快. 这说明过多的参数 (w 变大时参数增多) 导致了训练数据上的**过拟合** (overfitted).

对这种现象的一个解释是, 绝大部分的中文词少于 3 个字, 因此, 对于分词任务而言, 当 $w > 5$ 时, 大小为 5 的窗口以外的字就变成 "噪音" (noise) 了.

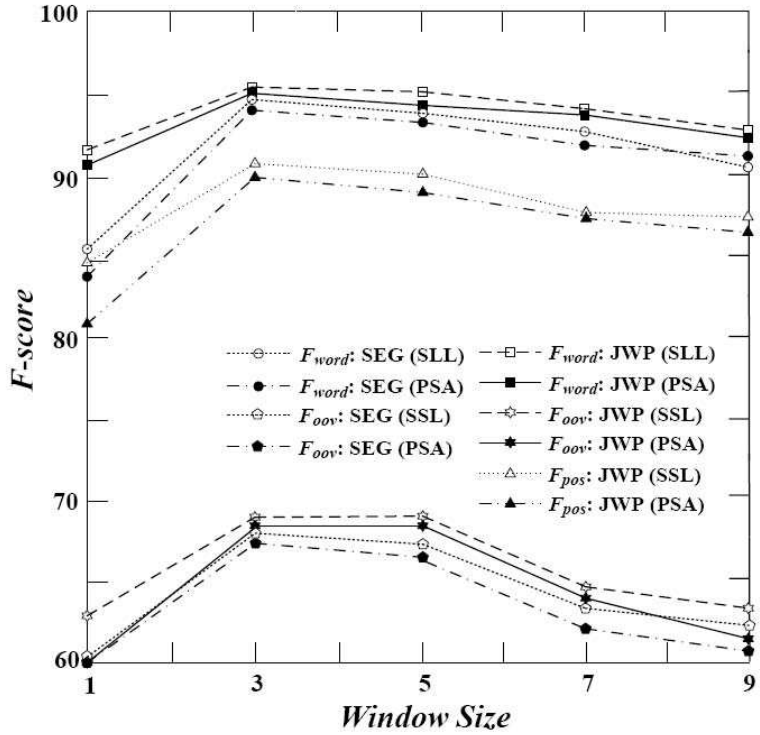

图 2 平均 F-score 随着窗口大小 w 的变化曲线图

图 3 给出了平均 F-score 随着参数 H 的变化曲线图. 通常来讲, 当 H 足够大时, 它对性能的影响很有限 (这和文 [3] 中对英语的测试结果是一致的).

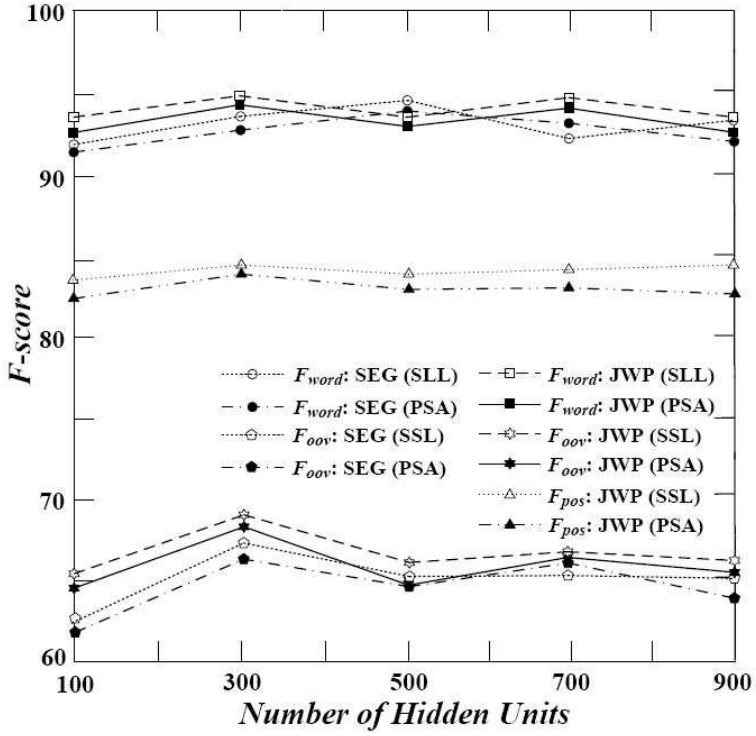

图 3 平均 F-score 随着隐单元个数 H 的变化曲线图

图 4 中的表格给出了以下实验中各参数的取值. 注意, 由图 2 可知当 $w = 3$ 时性能最好 (top performance), 但我们在下面的实验中取的是 $w = 5$, 这是因为接下来的实验中用的是更大的训练语料, 稀疏性问题将得到缓解. 此外, 在第三组实验中, 我们希望通过利用大规模非标注数据来获取更多的语法和语义信息, 更大的窗口可以更好地达到这个目的.

Hyper-parameter	**Value**
Window size	5
Number of hidden units	300
Character feature dimension	50
Learning rate	0.02

图 4 各参数的取值

图 5 中的表格给出了 SLL 和 PSA 在 development set 上执行分词任务时前五次迭代对应的 F-score 值. 由表可见: 当迭代次数超过 5 时, 两个算法的 F-score 就没什么差别了. 但对于每一次迭代, PSA 的训练时间至少比 SLL 要少 10%. 对于像 POS 标注和语义角色标注 (semantic role labeling, SRL) 等更复杂的任务 (它们对应的标注集更大) 来说, PSA 的训练时间将更有优势.

Iteration		**1**	**2**	**3**	**4**	**5**
SSL	F_{word}	49.89	69.56	88.91	90.19	91.24
	F_{oov}	15.92	27.54	54.37	55.89	59.74
	Time (s)	209	398	586	737	886
PSA	F_{word}	49.04	68.54	87.79	89.07	91.19
	F_{oov}	13.61	25.79	52.30	55.15	60.49
	Time (s)	184	343	497	610	754

图 5 SLL 和 PSA 执行分词任务时前五次迭代的 F-score 值

在 SLL 训练算法中, 需要进行大量形如 $\sum_i e^{x_i}$ 的求和运算, 而且这个和值很可能超出了编程语言中定义的双精度或浮点数的范围. 而我们的算法 (PSA) 避免了这样的问题, 不仅提高了效率, 也更容易实现.

§3.3 Closed Test on the SIGHAN Bakeoff

我们在数据集 Chinese Treebank (CTB) data sets from Bakeoff-3 上训练 SLL 和 PSA 两个算法, 并执行 SEG 和 JWP 任务. 图 6 中的表给出了相应的对比结果 (参数选取见图 4 中的表格). 由表可见: PSA 在 F-score 上比最好的算法要差一点, 但考虑到它完全没有使用其他任何额外的信息, 其表现相对来说还是不错的. 其他几个系统大多采用了一些额外的 heuristics 或者 resources 来提高性能.

值得注意的是, 对于 JWP 任务, 由于不同的算法用的是不同的 CTB 数据版本, 因此其比较不那么直接.

Approach		F_{word}	R_{oov}	F_{pos}
SEG	(Zhao et al., 2006)	93.30	70.70	–
	(Wang et al., 2006)	93.00	68.30	–
	(Zhu et al., 2006)	92.70	63.40	–
	(Zhang et al., 2006)	92.60	61.70	–
	(Feng et al., 2006)	91.70	68.00	–
	PSA	92.59	64.24	–
	PSA + LM	94.57	70.12	–
JWP	(Ng and Lou, 2004)	95.20	–	–
	(Zhang and Clark, 2008)	95.90	–	91.34
	(Jiang et al., 2008)	97.30	–	92.50
	(Kruengkrai et al., 2009)	96.11	–	90.85
	PSA	93.83	68.21	90.79
	PSA + LM	95.23	72.38	91.82

图 6 几个算法的比较

类似于文 [2] 和文 [3], 我们希望语法或语义上具有相似性的字在字向量空间 (the embedding space) 距离相近. 这样, 如果我们知道 '狗' 和 '猫' , 以及 '蹲' 和 '躺' 在语义上具有相似性, 那么, 由句子 '狗蹲在墙角' 就可以得到 '猫蹲在墙角' 或 '猫躺在墙角' 这样的句子. 下一小节, 我们介绍利用大规模非标注数据得到这种字向量的方法.

§3.4 Combined Approach

我们采用 Sina news 语料来获取具有更丰富语法和语义信息的字向量 (character embedding), 这需要训练一个语言模型, 它对字串的可接受性 (the acceptability of a piece of text) 进行评估. 这个语言模型仍然利用 PSA 来训练. 按照文 [3], 我们最小化

$$\sum_{h\in\mathcal{H}}\sum_{c'\in\mathcal{D}}\max\{0, 1 - f_\theta(c|h) + f_\theta(c'|h)\}, \tag{3.18}$$

其中

- $\mathcal{H}$: 所有可能的 text window (即字串).
- $\mathcal{D}$: 字典.
- $c|h$: 以字 c 为中心的窗口 h.
- $c'|h$: 将窗口 h 的中心字换为 c' 后得到的新窗口.
- $f_\theta(c|h)$: 以 $c|h$ 为输入得到的输出.
- $f_\theta(c'|h)$: 以 $c'|h$ 为输入得到的输出.

我们使用的字典来自训练语料, 即整个 Bakeoff-3 数据集, 它包含约 8000 个字. 整个无监督训练过程将近花了两周的时间. 通过这种方法得到的 character embeddings 将用来初始化算法 2.1 中的矩阵 $\mathcal{M}$, 且初始化以后, $\mathcal{M}$ 在训练过程中就不再发生变化.

图 6 的表中同时给出了 Combined Approach (表中的 PSA + LM) 的测试结果, 易知, 对于 SEG 和 JWP 任务, 这种方法都改进了原有的 PSA. 对于 JWP 中的 POS 任务, PSA + LM 的 F-score 为 91.82, 已经很接近 92.5 了.

图 7 中的表格对比了我们的分词系统和其他两个基于 CRF 的分词系统在 test data from CTB-3 上的解码速度. 忽略实现上的差异, 我们的系统的优势显而易见, 而且, 与其他两个系统相比, 我们的系统所需的内存更少.

System	Number of parameters	Time (s)
(Tsai et al., 2006)	3.1×10^6	1669
(Zhao et al., 2006)	3.8×10^6	2382
Neural network	4.7×10^5	138

图 7 解码速度的比较

§4 总结

本文提出了一种训练神经网络的 perceptron-style 算法 (PSA), 与 maximum-likelihood 算法相比, 它不仅更容易实现, 而且在速度上更有优势 (尽管在 F-score 指标上稍微差一点). 我们将该算法用于中文分词和词性标注任务, 通过利用对大规模非标注数据学习得到的字向量表示, 算法的表现已经很接近目前最好的算法.

虽然本文主要研究在不使用额外与具体任务相关的特征的前提下 PSA 在中文分词和词性标注任务中的表现. 但是至少还有三个方面值得进一步研究:

1. 引入特定的语言特征 (如地名词典);

2. 结合某些技巧, 如 cascading, voting 和 ensemble;

3. 为具体任务量身定制特定的网络架构.

参考文献

[1] Xiaoqing Zheng, Hanyang Chen, Tianyu Xu. Deep Learning for Chinese Word Segmentation and POS tagging. Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 647-657.

[2] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic langugage model. Journal of Machine Learning Research, 3: 1137 ┼ 1155.

[3] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. Journal of Machine Learning Research, 12: 2493-2537.

[4] Michael Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In Proceedings of the International Conference on Empirical Methods in Natural Language Processing (EMNLP'02).

[5] Hwee Tou Ng, Jin Kiat Low. Chinese Part-of-Speech Tagging: One-at-a-Time or All-at-Once? Word-Based or Character-Based? In Proceedings of the International Conference on Empirical Methods in Natural Language Processing (EMNLP'04).