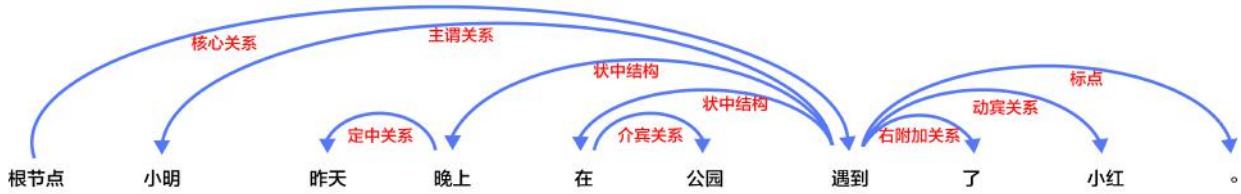


图1. 依存句法分析句法树示例

然而，完全句法分析需要确定句子所包含的全部句法信息，并确定句子各成分之间的关系，是一个非常困难的任务，目前技术下的句法分析准确率并不高，句法分析的细微错误都会导致SRL的错误。为了降低问题的复杂度，同时获得一定的句法结构信息，“浅层句法分析”的思想应运而生。浅层句法分析也称为部分句法分析（partial parsing）或语块划分（chunking）。和完全句法分析得到一颗完整的句法树不同，浅层句法分析只需要识别句子中某些结构相对简单的独立成分，例如：动词短语，这些被识别出来的结构称为语块。为了回避 “无法获得准确率较高的句法树” 所带来的困难，一些研究[1]也提出了基于语块（chunk）的SRL方法。基于语块的SRL方法将SRL作为一个序列标注问题来解决。序列标注任务一般都会采用BIO表示方式来定义序列标注的标签集，我们先来介绍这种表示方法。在BIO表示法中，B代表语块的开始，I代表语块的中间，O代表语块结束。通过B、I、O 三种标记将不同的语块赋予不同的标签，例如：对于一个角色为A的论元，将它所包含的第一个语块赋予标签B-A，将它所包含的其它语块赋予标签I-A，不属于任何论元的语块赋予标签O。

我们继续以上面的这句话为例，图1展示了BIO表示方法。

| 输入序列 | 小明 | 昨天 | 晚上 | 在 | 公园 | 遇到 | 了 | 小红 | 。 |
|---|---|---|---|---|---|---|---|---|---|
| 语块 | B-NP | B-NP | I-NP | B-PP | B-NP | B-VP | | B-NP | |
| 标注序列 | B-Agent | B-Time | I-Time | O | B-Location | B-Predicate | O | B-Patient | O |
| 角色 | Agent | Time | Time | | Location | Predicate | O | Patient | |

图2. BIO标注方法示例

从上面的例子可以看到，根据序列标注结果可以直接得到论元的语义角色标注结果，是一个相对简单的过程。这种简单性体现在：（1）依赖浅层句法分析，降低了句法分析的要求和难度；（2）没有了候选论元剪除这一步骤；（3）论元的识别和论元标注是同时实现的。这种一体化处理论元识别和论元标注的方法，简化了流程，降低了错误累积的风险，往往能够取得更好的结果。

与基于语块的SRL方法类似，在本教程中我们也将SRL看作一个序列标注问题，不同的是，我

们只依赖输入文本序列，不依赖任何额外的语法解析结果或是复杂的人造特征，利用深度神经网络构建一个端到端学习的SRL系统。我们以CoNLL-2004 and CoNLL-2005 Shared Tasks任务中SRL任务的公开数据集为例，实践下面的任务：给定一句话和这句话里的一个谓词，通过序列标注的方式，从句子中找到谓词对应的论元，同时标注它们的语义角色。

# 模型概览

循环神经网络（Recurrent Neural Network）是一种对序列建模的重要模型，在自然语言处理任务中有着广泛地应用。不同于前馈神经网络（Feed-forward Neural Network），RNN能够处理输入之间前后关联的问题。LSTM是RNN的一种重要变种，常用来学习长序列中蕴含的长程依赖关系，我们在情感分析一篇中已经介绍过，这一篇中我们依然利用LSTM来解决SRL问题。

## 栈式循环神经网络（Stacked Recurrent Neural Network）

深层网络有助于形成层次化特征，网络上层在下层已经学习到的初级特征基础上，形成更复杂的高级特征。尽管LSTM沿时间轴展开后等价于一个非常“深”的前馈网络，但由于LSTM各个时间步参数共享，$t-1$时刻状态到$t$时刻的映射，始终只经过了一次非线性映射，也就是说单层LSTM对状态转移的建模是 “浅” 的。堆叠多个LSTM单元，令前一个LSTM$t$时刻的输出，成为下一个LSTM单元$t$时刻的输入，帮助我们构建起一个深层网络，我们把它称为第一个版本的栈式循环神经网络。深层网络提高了模型拟合复杂模式的能力，能够更好地建模跨不同时间步的模式[2]。

然而，训练一个深层LSTM网络并非易事。纵向堆叠多个LSTM单元可能遇到梯度在纵向深度上传播受阻的问题。通常，堆叠4层LSTM单元可以正常训练，当层数达到4~8层时，会出现性能衰减，这时必须考虑一些新的结构以保证梯度纵向顺畅传播，这是训练深层LSTM网络必须解决的问题。我们可以借鉴LSTM解决 “梯度消失梯度爆炸” 问题的智慧之一：在记忆单元（Memory Cell）这条信息传播的路线上没有非线性映射，当梯度反向传播时既不会衰减、也不会爆炸。因此，深层LSTM模型也可以在纵向上添加一条保证梯度顺畅传播的路径。

一个LSTM单元完成的运算可以被分为三部分：（1）输入到隐层的映射（input-to-hidden）：每个时间步输入信息$x$会首先经过一个矩阵映射，再作为遗忘门，输入门，记忆单元，输出门的输入，注意，这一次映射没有引入非线性激活；（2）隐层到隐层的映射（hidden-to-

hidden）：这一步是LSTM计算的主体，包括遗忘门，输入门，记忆单元更新，输出门的计算；（3）隐层到输出的映射（hidden-to-output）：通常是简单的对隐层向量进行激活。我们在第一个版本的栈式网络的基础上，加入一条新的路径：除上一层LSTM输出之外，将前层LSTM的输入到隐层的映射作为的一个新的输入，同时加入一个线性映射去学习一个新的变换。

图3是最终得到的栈式循环神经网络结构示意图。

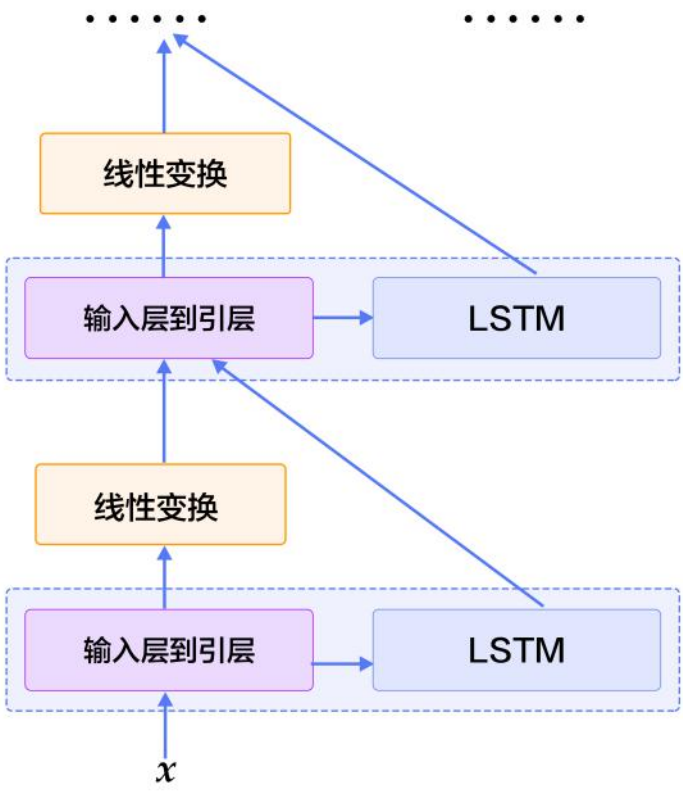


图3. 基于LSTM的栈式循环神经网络结构示意图

## 双向循环神经网络（Bidirectional Recurrent Neural Network）

在LSTM中，$t$时刻的隐藏层向量编码了到$t$时刻为止所有输入的信息，但$t$时刻的LSTM可以看到历史，却无法看到未来。在绝大多数自然语言处理任务中，我们几乎总是能拿到整个句子。这种情况下，如果能够像获取历史信息一样，得到未来的信息，对序列学习任务会有很大的帮助。

为了克服这一缺陷，我们可以设计一种双向循环网络单元，它的思想简单且直接：对上一节的

栈式循环神经网络进行一个小小的修改，堆叠多个LSTM单元，让每一层LSTM单元分别以：正向、反向、正向 …… 的顺序学习上一层的输出序列。于是，从第2层开始，$t$时刻我们的LSTM单元便总是可以看到历史和未来的信息。图4是基于LSTM的双向循环神经网络结构示意图。

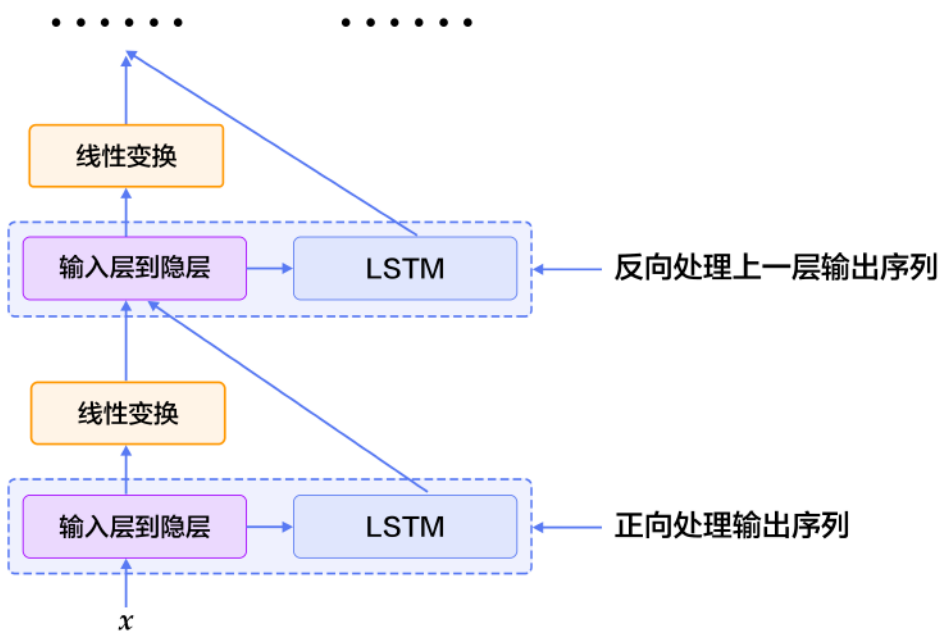


图4. 基于LSTM的双向循环神经网络结构示意图

需要说明的是，这种双向RNN结构和Bengio等人在机器翻译任务中使用的双向RNN结构[3, 4]并不相同，我们会在后续机器翻译任务中，介绍另一种双向循环神经网络。

## 条件随机场 (Conditional Random Field)

使用神经网络模型解决问题的思路通常是：前层网络学习输入的特征表示，网络的最后一层在特征基础上完成最终的任务。在SRL任务中，深层LSTM网络学习输入的特征表示，条件随机场（Conditional Random Filed， CRF）在特征的基础上完成序列标注，处于整个网络的末端。

CRF是一种概率化结构模型，可以看作是一个概率无向图模型，结点表示随机变量，边表示随机变量之间的概率依赖关系。简单来讲，CRF学习条件概率$P(X|Y)$，其中$X=(x_1,x_2,\ldots,x_n)$是输入序列，$Y=(y_1,y_2,\ldots,y_n)$是标记序列；解码过程是给定$X$序列求解令$P(Y|X)$最大的$Y$序列，即$Y^*=\arg\max_Y P(Y|X)$。

序列标注任务只需要考虑输入和输出都是一个线性序列，并且由于我们只是将输入序列作为条

件，不做任何条件独立假设，因此输入序列的元素之间并不存在图结构。综上，在序列标注任务中使用的是如图5所示的定义在链式图上的CRF，称之为线性链条件随机场（Linear Chain Conditional Random Field）。

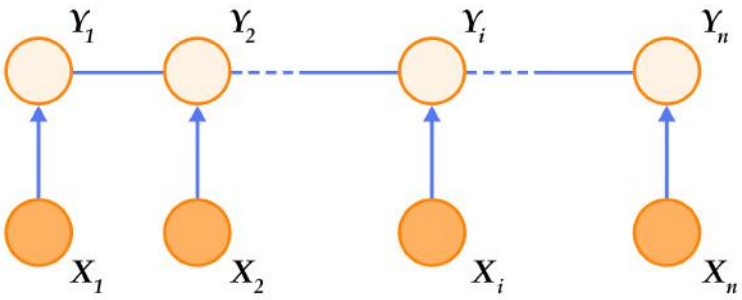


图5. 序列标注任务中使用的线性链条件随机场

根据线性链条件随机场上的因子分解定理[5]，在给定观测序列$X$时，一个特定标记序列$Y$的概率可以定义为：

$$p(Y|X) = \frac{1}{Z(X)} \exp\left(\sum_{i=1}^{n}\left(\sum_{j}\lambda_j t_j(y_{i-1}, y_i, X, i) + \sum_{k}\mu_k s_k(y_i, X, i)\right)\right)$$

其中$Z(X)$是归一化因子，$t_j$ 是定义在边上的特征函数，依赖于当前和前一个位置，称为转移特征，表示对于输入序列$X$及其标注序列在 $i$及$i-1$位置上标记的转移概率。$s_k$是定义在结点上的特征函数，称为状态特征，依赖于当前位置，表示对于观察序列$X$及其$i$位置的标记概率。$\lambda_j$ 和 $\mu_k$ 分别是转移特征函数和状态特征函数对应的权值。实际上，$t$和$s$可以用相同的数学形式表示，再对转移特征和状态特在各个位置$i$求和有：
$f_k(Y, X) = \sum_{i=1}^{n} f_k(y_{i-1}, y_i, X, i)$，把$f$统称为特征函数，于是$P(Y|X)$可表示为：

$$p(Y|X, W) = \frac{1}{Z(X)} \exp\sum_{k}\omega_k f_k(Y, X)$$

$\omega$是特征函数对应的权值，是CRF模型要学习的参数。训练时，对于给定的输入序列和对应的标记序列集合$D = [(X_1, Y_1), (X_2, Y_2), \ldots, (X_N, Y_N)]$，通过正则化的极大似然估计，求解如下优化目标：

$$L(\lambda, D) = -\log\left(\prod_{m=1}^{N} p(Y_m|X_m, W)\right) + C\frac{1}{2}\|W\|^2$$

这个优化目标可以通过反向传播算法和整个神经网络一起求解。解码时，对于给定的输入序列$\boldsymbol{X}$，通过解码算法（通常有：维特比算法、Beam Search）求令出条件概率$\bar{P}(\boldsymbol{Y}|\boldsymbol{X})$最大的输出序列$\bar{\boldsymbol{Y}}$。

## 深度双向LSTM（DB-LSTM）SRL模型

在SRL任务中，输入是 “谓词” 和 “一句话”，目标是从这句话中找到谓词的论元，并标注论元的语义角色。如果一个句子含有$\boldsymbol{n}$个谓词，这个句子会被处理$\boldsymbol{n}$次。一个最为直接的模型是下面这样：

1. 构造输入；
    - 输入1是谓词，输入2是句子
    - 将输入1扩展成和输入2一样长的序列，用one-hot方式表示；
2. one-hot方式的谓词序列和句子序列通过词表，转换为实向量表示的词向量序列；
3. 将步骤2中的2个词向量序列作为双向LSTM的输入，学习输入序列的特征表示；
4. CRF以步骤3中模型学习到的特征为输入，以标记序列为监督信号，实现序列标注；

大家可以尝试上面这种方法。这里，我们提出一些改进，引入两个简单但对提高系统性能非常有效的特征：

- 谓词上下文：上面的方法中，只用到了谓词的词向量表达谓词相关的所有信息，这种方法始终是非常弱的，特别是如果谓词在句子中出现多次，有可能引起一定的歧义。从经验出发，谓词前后若干个词的一个小片段，能够提供更丰富的信息，帮助消解歧义。于是，我们把这样的经验也添加到模型中，为每个谓词同时抽取一个“谓词上下文” 片段，也就是从这个谓词前后各取$\boldsymbol{n}$个词构成的一个窗口片段；
- 谓词上下文区域标记：为句子中的每一个词引入一个0-1二值变量，表示它们是否在“谓词上下文”片段中；

修改后的模型如下（图6是一个深度为4的模型结构示意图）：

1. 构造输入
    - 输入1是句子序列，输入2是谓词序列，输入3是谓词上下文，从句子中抽取这个谓词前后各$\boldsymbol{n}$个词，构成谓词上下文，用one-hot方式表示，输入4是谓词上下文区域标记，标记了句子中每一个词是否在谓词上下文中；
    - 将输入2~3均扩展为和输入1一样长的序列；

2. 输入1~4均通过词表取词向量转换为实向量表示的词向量序列；其中输入1、3共享同一个词表，输入2和4各自独有词表；
3. 第2步的4个词向量序列作为双向LSTM模型的输入；LSTM模型学习输入序列的特征表示，得到新的特性表示序列；
4. CRF以第3步中LSTM学习到的特征为输入，以标记序列为监督信号，完成序列标注；

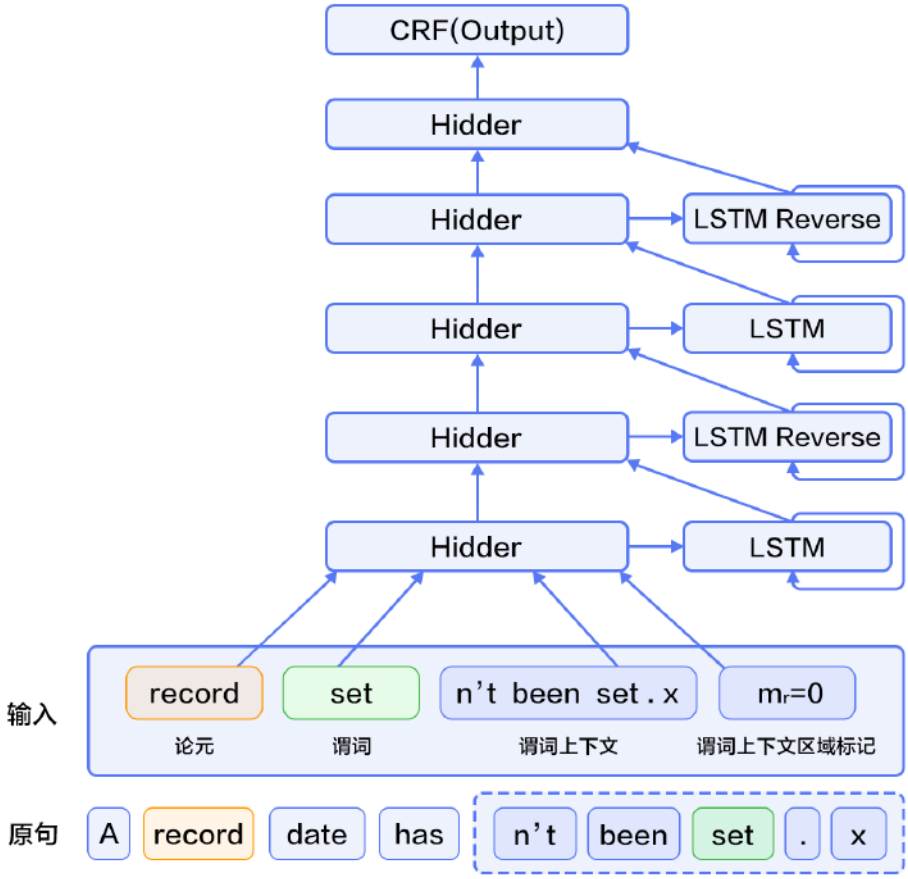


图6. SRL任务上的深层双向LSTM模型

# 数据准备

## 数据介绍与下载

在此教程中，我们选用CoNLL 2005SRL任务开放出的数据集作为示例。运行 `sh ./get_data.sh` 会自动从官方网站上下载原始数据。需要特别说明的是，CoNLL 2005 SRL任务的训练数集和开发集在比赛之后并非免费进行公开，目前，能够获取到的只有测试集，包括Wall Street Journal的23节和Brown语料集中的3节。在本教程中，我们以测试集中

的WSJ数据为训练集来讲解模型。但是，由于测试集中样本的数量远远不够，如果希望训练一个可用的神经网络SRL系统，请考虑付费获取全量数据。

原始数据中同时包括了词性标注、命名实体识别、语法解析树等多种信息。本教程中，我们使用test.wsj文件夹中的数据进行训练和测试，并只会用到words文件夹（文本序列）和props文件夹（标注结果）下的数据。本教程使用的数据目录如下：

```
conll05st-release/
└── test.wsj
    ├── props   # 标注结果
    └── words   # 输入文本序列
```

标注信息源自Penn TreeBank[7]和PropBank[8]的标注结果。PropBank标注结果的标签和我们在文章一开始示例中使用的标注结果标签不同，但原理是相同的，关于标注结果标签含义的说明，请参考论文[9]。

除数据之外，`get_data.sh`同时下载了以下资源：

| 文件名称 | 说明 |
| --- | --- |
| word_dict | 输入句子的词典，共计44068个词 |
| label_dict | 标记的词典，共计106个标记 |
| predicate_dict | 谓词的词典，共计3162个词 |
| emb | 一个训练好的词表，32维 |

我们在英文维基百科上训练语言模型得到了一份词向量用来初始化SRL模型。在SRL模型训练过程中，词向量不再被更新。关于语言模型和词向量可以参考词向量 这篇教程。我们训练语言模型的语料共有995,000,000个token，词典大小控制为4900,000词。CoNLL 2005训练语料中有5%的词不在这4900,000个词中，我们将它们全部看作未登录词，用`<unk>`表示。

## 数据预处理

脚本在下载数据之后，又调用了`extract_pair.py`和`extract_dict_feature.py`两个子脚本进行数据预处理，前者完成了下面的第1步，后者完成了下面的2~4步：

1. 将文本序列和标记序列其合并到一条记录中；
2. 一个句子如果含有$n$个谓词，这个句子会被处理$n$次，变成$n$条独立的训练样本，每个样本一个不同的谓词；
3. 抽取谓词上下文和构造谓词上下文区域标记；
4. 构造以BIO法表示的标记；

`data/feature`文件是处理好的模型输入，一行是一条训练样本，以"\t"分隔，共9列，分别是：句子序列、谓词、谓词上下文（占 5 列）、谓词上下区域标志、标注序列。下表是一条训练样本的示例。

| 句子序列 | 谓词 | 谓词上下文（窗口 = 5） | 谓词上下文区域标记 | 标注序列 |
|---|---|---|---|---|
| A | set | n't been set . × | 0 | B-A1 |
| record | set | n't been set . × | 0 | I-A1 |
| date | set | n't been set . × | 0 | I-A1 |
| has | set | n't been set . × | 0 | O |
| n't | set | n't been set . × | 1 | B-AM-NEG |
| been | set | n't been set . × | 1 | O |
| set | set | n't been set . × | 1 | B-V |
| . | set | n't been set . × | 1 | O |

## 提供数据给 PaddlePaddle

1. 使用hook函数进行PaddlePaddle输入字段的格式定义。

```
def hook(settings, word_dict, label_dict, predicate_dict,
**kwargs):
    settings.word_dict = word_dict    # 获取句子序列的字典
    settings.label_dict = label_dict  # 获取标记序列的字典
    settings.predicate_dict = predicate_dict  # 获取谓词的字典

    #  所有输入特征都是使用one-hot表示序列，在PaddlePaddle中是
interger_value_sequence类型
    #  input_types是一个字典，字典中每个元素对应着配置中的一个
data_layer, key恰好就是data_layer的名字
```

```
    settings.input_types = {
            'word_data': integer_value_sequence(len(word_dict)),
# 句子序列
            'ctx_n2_data': integer_value_sequence(len(word_dict)),
# 谓词上下文中的第1个词
            'ctx_n1_data': integer_value_sequence(len(word_dict)),
# 谓词上下文中的第2个词
            'ctx_0_data': integer_value_sequence(len(word_dict)),
# 谓词上下文中的第3个词
            'ctx_p1_data': integer_value_sequence(len(word_dict)),
# 谓词上下文中的第4个词
            'ctx_p2_data': integer_value_sequence(len(word_dict)),
# 谓词上下文中的第5个词
            'verb_data': integer_value_sequence(len(predicate_dict))
,  # 谓词
            'mark_data': integer_value_sequence(2),  # 谓词上下文区域标
记
            'target': integer_value_sequence(len(label_dict))  # 标记
序列
    }
```

2. 使用process将数据逐一提供给PaddlePaddle，只需要考虑如何从原始数据文件中返回一条训练样本。

```
def process(settings, file_name):
    with open(file_name, 'r') as fdata:
        for line in fdata:
            sentence, predicate, ctx_n2, ctx_n1, ctx_0, ctx_p1, ctx_
p2,  mark, label = \
                line.strip().split('\t')

            # 句子文本
            words = sentence.split()
            sen_len = len(words)
            word_slot = [settings.word_dict.get(w, UNK_IDX) for w in
words]

            # 一个谓词，这里将谓词扩展成一个和句子一样长的序列
            predicate_slot = [settings.predicate_dict.get(predicate)
] * sen_len

            # 在教程中，我们使用一个窗口为 5 的谓词上下文窗口：谓词和这个谓词
前后隔两个词
```

```
            # 这里会将窗口中的每一个词，扩展成和输入句子一样长的序列
            ctx_n2_slot = [settings.word_dict.get(ctx_n2, UNK_IDX)]
* sen_len
            ctx_n1_slot = [settings.word_dict.get(ctx_n1, UNK_IDX)]
* sen_len
            ctx_0_slot = [settings.word_dict.get(ctx_0, UNK_IDX)] *
sen_len
            ctx_p1_slot = [settings.word_dict.get(ctx_p1, UNK_IDX)]
* sen_len
            ctx_p2_slot = [settings.word_dict.get(ctx_p2, UNK_IDX)]
* sen_len

            # 谓词上下文区域标记，是一个二值特征
            marks = mark.split()
            mark_slot = [int(w) for w in marks]

            label_list = label.split()
            label_slot = [settings.label_dict.get(w) for w in
label_list]
            yield {
                'word_data': word_slot,
                'ctx_n2_data': ctx_n2_slot,
                'ctx_n1_data': ctx_n1_slot,
                'ctx_0_data': ctx_0_slot,
                'ctx_p1_data': ctx_p1_slot,
                'ctx_p2_data': ctx_p2_slot,
                'verb_data': predicate_slot,
                'mark_data': mark_slot,
                'target': label_slot
            }
```

# 模型配置说明

## 数据定义

首先通过 define_py_data_sources2 从dataprovider中读入数据。配置文件中会读取三个字典：输入文本序列的字典、标记的字典、谓词的字典，并传给data provider，data provider会利用这三个字典，将相应的文本输入转换成one-hot序列。

```
define_py_data_sources2(
```

```
        train_list=train_list_file,
        test_list=test_list_file,
        module='dataprovider',
        obj='process',
        args={
            'word_dict': word_dict,    # 输入文本序列的字典
            'label_dict': label_dict, # 标记的字典
            'predicate_dict': predicate_dict   # 谓词的词典
        }
)
```

## 算法配置

在这里，我们指定了模型的训练参数，选择了$L_2$正则、学习率和batch size，并使用带Momentum的随机梯度下降法作为优化算法。

```
settings(
    batch_size=150,
    learning_method=MomentumOptimizer(momentum=0),
    learning_rate=2e-2,
    regularization=L2Regularization(8e-4),
    model_average=ModelAverage(average_window=0.5, max_average_window=10000)
)
```

## 模型结构

1. 定义输入数据维度及模型超参数。

```
mark_dict_len = 2     # 谓上下文区域标志的维度，是一个0-1 2值特征，因此维度为2
word_dim = 32         # 词向量维度
mark_dim = 5          # 谓词上下文区域通过词表被映射为一个实向量，这个是相邻的维度
hidden_dim = 512      # LSTM隐层向量的维度 ： 512 / 4
depth = 8             # 栈式LSTM的深度

word = data_layer(name='word_data', size=word_dict_len)
predicate = data_layer(name='verb_data', size=pred_len)

ctx_n2 = data_layer(name='ctx_n2_data', size=word_dict_len)
```

```
ctx_n1 = data_layer(name='ctx_n1_data', size=word_dict_len)
ctx_0 = data_layer(name='ctx_0_data', size=word_dict_len)
ctx_p1 = data_layer(name='ctx_p1_data', size=word_dict_len)
ctx_p2 = data_layer(name='ctx_p2_data', size=word_dict_len)
mark = data_layer(name='mark_data', size=mark_dict_len)

if not is_predict:
    target = data_layer(name='target', size=label_dict_len)    # 标记序列只在训练和测试流程中定义
```

这里需要特别说明的是hidden_dim = 512指定了LSTM隐层向量的维度为128维，关于这一点请参考PaddlePaddle官方文档中lstmemory的说明。

2. 将句子序列、谓词、谓词上下文、谓词上下文区域标记通过词表，转换为实向量表示的词向量序列。

```


# 在本教程中，我们加载了预训练的词向量，这里设置了：is_static=True


# is_static 为 True 时保证了在训练 SRL 模型过程中，词表不再更新

emb_para = ParameterAttribute(name='emb', initial_std=0., is_static=True)

word_input = [word, ctx_n2, ctx_n1, ctx_0, ctx_p1, ctx_p2]
emb_layers = [
        embedding_layer(
            size=word_dim, input=x, param_attr=emb_para) for x in word_input
]
emb_layers.append(predicate_embedding)
mark_embedding = embedding_layer(
        name='word_ctx-in_embedding', size=mark_dim, input=mark, param_attr=std_0)
emb_layers.append(mark_embedding)
```

3. 8个LSTM单元以“正向/反向”的顺序对所有输入序列进行学习。

```

#  std_0 指定的参数以均值为0的高斯分布初始化，用在LSTM的bias初始化中

```

```
std_0 = ParameterAttribute(initial_std=0.)

hidden_0 = mixed_layer(
    name='hidden0',
    size=hidden_dim,
    bias_attr=std_default,
    input=[
        full_matrix_projection(
            input=emb, param_attr=std_default) for emb in emb_layers
    ])
lstm_0 = lstmemory(
    name='lstm0',
    input=hidden_0,
    act=ReluActivation(),
    gate_act=SigmoidActivation(),
    state_act=SigmoidActivation(),
    bias_attr=std_0,
    param_attr=lstm_para_attr)
    input_tmp = [hidden_0, lstm_0]

for i in range(1, depth):
    mix_hidden = mixed_layer(
        name='hidden' + str(i),
        size=hidden_dim,
        bias_attr=std_default,
        input=[
            full_matrix_projection(
                input=input_tmp[0], param_attr=hidden_para_attr),
            full_matrix_projection(
                input=input_tmp[1], param_attr=lstm_para_attr)
        ])
    lstm = lstmemory(
        name='lstm' + str(i),
        input=mix_hidden,
        act=ReluActivation(),
        gate_act=SigmoidActivation(),
        state_act=SigmoidActivation(),
        reverse=((i % 2) == 1),
        bias_attr=std_0,
        param_attr=lstm_para_attr)

    input_tmp = [mix_hidden, lstm]
```

4. 取最后一个栈式LSTM的输出和这个LSTM单元的输入到隐层映射，经过一个全连接层映射

到标记字典的维度，得到最终的特征向量表示。

```
feature_out = mixed_layer(
    name='output',
    size=label_dict_len,
    bias_attr=std_default,
    input=[
        full_matrix_projection(
            input=input_tmp[0], param_attr=hidden_para_attr),
        full_matrix_projection(
            input=input_tmp[1], param_attr=lstm_para_attr)
    ], )
```

5. CRF层在网络的末端，完成序列标注。

```
crf_l = crf_layer(
        name='crf',
        size=label_dict_len,
        input=feature_out,
        label=target,
        param_attr=ParameterAttribute(
            name='crfw', initial_std=default_std, learning_rate=mix_
hidden_lr))
```

# 训练模型

执行`sh train.sh`进行模型的训练，其中指定了总共需要训练150个pass。

```
paddle train \
  --config=./db_lstm.py \
  --save_dir=./output \
  --trainer_count=1 \
  --dot_period=500 \
  --log_period=10 \
  --num_passes=200 \
  --use_gpu=false \
  --show_parameter_stats_period=10 \
  --test_all_data_in_one_period=1 \
2>&1 | tee 'train.log'
```

训练日志示例如下。

```
I1224 18:11:53.661479  1433 TrainerInternal.cpp:165]  Batch=880 samples=145305 AvgCost=2.11541 CurrentCost=1.8645 Eval: __sum_evaluator_0__=0.607942  CurrentEval: __sum_evaluator_0__=0.59322
I1224 18:11:55.254021  1433 TrainerInternal.cpp:165]  Batch=885 samples=146134 AvgCost=2.11408 CurrentCost=1.88156 Eval: __sum_evaluator_0__=0.607299  CurrentEval: __sum_evaluator_0__=0.494572
I1224 18:11:56.867604  1433 TrainerInternal.cpp:165]  Batch=890 samples=146987 AvgCost=2.11277 CurrentCost=1.88839 Eval: __sum_evaluator_0__=0.607203  CurrentEval: __sum_evaluator_0__=0.590856
I1224 18:11:58.424069  1433 TrainerInternal.cpp:165]  Batch=895 samples=147793 AvgCost=2.11129 CurrentCost=1.84247 Eval: __sum_evaluator_0__=0.607099  CurrentEval: __sum_evaluator_0__=0.588089
I1224 18:12:00.006893  1433 TrainerInternal.cpp:165]  Batch=900 samples=148611 AvgCost=2.11148 CurrentCost=2.14526 Eval: __sum_evaluator_0__=0.607882  CurrentEval: __sum_evaluator_0__=0.749389
I1224 18:12:00.164089  1433 TrainerInternal.cpp:181]  Pass=0 Batch=901 samples=148647 AvgCost=2.11195 Eval: __sum_evaluator_0__=0.60793
```

经过150个 pass 后，得到平均 error 约为 0.0516055。

# 应用模型

训练好的$N$个pass，会得到$N$个模型，我们需要从中选择一个最优模型进行预测。通常做法是在开发集上进行调参，并基于我们关心的某个性能指标选择最优模型。本教程的`predict.sh`脚本简单地选择了测试集上标记错误最少的那个pass（这里是pass-00100）用于预测。

预测时，我们需要将配置中的 `crf_layer` 删掉，替换为 `crf_decoding_layer`，如下所示：

```
crf_dec_l = crf_decoding_layer(
        name='crf_dec_l',
        size=label_dict_len,
        input=feature_out,
        param_attr=ParameterAttribute(name='crfw'))
```

运行`python predict.py`脚本，便可使用指定的模型进行预测。

```
python predict.py
      -c db_lstm.py  # 指定配置文件
      -w output/pass-00100  # 指定预测使用的模型所在的路径
      -l data/targetDict.txt  # 指定标记的字典
      -p data/verbDict.txt  # 指定谓词的词典
      -d data/wordDict.txt # 指定输入文本序列的字典
      -i data/feature  # 指定输入数据的路径
      -o predict.res  # 指定标记结果输出到文件的路径
```

预测结束后，在 - o 参数所指定的标记结果文件中，我们会得到如下格式的输出：每行是一条样本，以 “\t” 分隔的 2 列，第一列是输入文本，第二列是标记的结果。通过BIO标记可以直接得到论元的语义角色标签。

```
The interest-only securities were priced at 35 1\/2 to yield 10.72 % .
B-A0 I-A0 I-A0 O O O O O O B-V B-A1 I-A1 O
```

## 总结

语义角色标注是许多自然语言理解任务的重要中间步骤。这篇教程中我们以语义角色标注任务为例，介绍如何利用PaddlePaddle进行序列标注任务。教程中所介绍的模型来自我们发表的论文[10]。由于 CoNLL 2005 SRL任务的训练数据目前并非完全开放，教程中只使用测试数据作为示例。在这个过程中，我们希望减少对其它自然语言处理工具的依赖，利用神经网络数据驱动、端到端学习的能力，得到一个和传统方法可比、甚至更好的模型。在论文中我们证实了这种可能性。关于模型更多的信息和讨论可以在论文中找到。

## 参考文献


1. Sun W, Sui Z, Wang M, et al. Chinese semantic role labeling with shallow parsing[C]//Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3-Volume 3. Association for Computational Linguistics, 2009: 1475-1483.
2. Pascanu R, Gulcehre C, Cho K, et al. How to construct deep recurrent neural networks[J]. arXiv preprint arXiv:1312.6026, 2013.
3. Cho K, Van Merriënboer B, Gulcehre C, et al. Learning phrase representations using

RNN encoder-decoder for statistical machine translation[J]. arXiv preprint arXiv:1406.1078, 2014.
4. Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate[J]. arXiv preprint arXiv:1409.0473, 2014.
5. Lafferty J, McCallum A, Pereira F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data[C]//Proceedings of the eighteenth international conference on machine learning, ICML. 2001, 1: 282-289.
6. 李航. 统计学习方法[J]. 清华大学出版社, 北京, 2012.
7. Marcus M P, Marcinkiewicz M A, Santorini B. Building a large annotated corpus of English: The Penn Treebank[J]. Computational linguistics, 1993, 19(2): 313-330.
8. Palmer M, Gildea D, Kingsbury P. The proposition bank: An annotated corpus of semantic roles[J]. Computational linguistics, 2005, 31(1): 71-106.
9. Carreras X, Màrquez L. Introduction to the CoNLL-2005 shared task: Semantic role labeling[C]//Proceedings of the Ninth Conference on Computational Natural Language Learning. Association for Computational Linguistics, 2005: 152-164.
10. Zhou J, Xu W. End-to-end learning of semantic role labeling using recurrent neural networks[C]//Proceedings of the Annual Meeting of the Association for Computational Linguistics. 2015.