PyTorch Bi-LSTM+CRF NER标注代码精读


追梦人
AI平台、搜索推荐

前言

首先，本文是对pytorch官方的Bi-LSTM+CRF实现的代码解读，原文地址：

https://pytorch.org/tutorials/beginner/nlp/advanced_tutorial.html#bi-lstm-...
pytorch.org

然后，要搞清楚为什么要用它而不是其它序列模型，如LSTM、Bi-LSTM。

最后，我们对代码的解读分为三部分：概率计算、参数学习、预测问题。其中概率计算是为了引出优化目标；参数学习指Bi-LSTM中的参数与CRF中的转移矩阵；预测问题就是给定一个输入序列预测最可能的输出序列。后续会详细说明。

适用场景

NER序列标注可用方法：HMM、CRF、LSTM/Bi-LSTM、LSTM/Bi-LSTM+CRF。之所以把Bi-LSTM+CRF做为经典解决方案，原因如下：

- HMM是生成模型(精度不如判别模型)，且不同时刻的隐状态相互独立。
- CRF是判别模型且可增加不同时刻隐状态之间的约束，但需要人工设计特征函数。
- LSTM模型输出的隐状态在不同时刻相互独立，它可以适当加深横向(序列长度)纵向(某时刻layer层数)层次提升模型效果。

采用Bi-LSTM+CRF就是结合了Bi-LSTM的特征表达能力与CRF的无向图生成模型的优点，成为经典就是必然。其典型架构如下图：

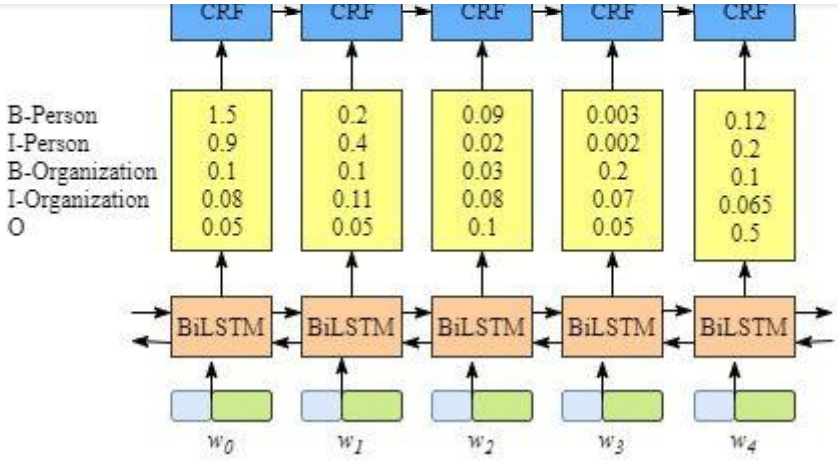

图1 Bi-LSTM+CRF架构图

注：在Bi-LSTM+CRF架构中，CRF最终的计算基于状态转移概率矩阵和发射概率矩阵(均指非归一化概率)。而Bi-LSTM的输出就充当了上述发射概率矩阵的角色。

代码详解

1.概率计算

有了训练数据，对于输入序列x与其对应的标签序列y，可以定义score(x,y)如下式(A为状态转移矩阵，$A_{i,j}$ 代表从 $tag_i \rightarrow tag_j$ 的概率；而B代表发射矩阵，$B_{i,j}$ 代表 $w_i \rightarrow tag_j$ 的概率。前述概率均指非规一化概率)：

$$s(x,y) = \sum_{i=0}^{n} A_{y_i,y_{i+1}} + \sum_{i=1}^{n} B_{i,y_i} \quad \text{(公式1)}$$

基于上式输出序列取y的概率定义如下(Y_x 为输入序列x所有可能的标注序列集合)：

$$p(y|x) = \frac{e^{s(x,y)}}{\sum_{\tilde{y} \in Y_x} e^{s(x,\tilde{y})}} \quad \text{(公式2)}$$

训练过程中我们要把**公式2**最大化，现在把它做一些必要的转化：

$$log(p(y|x)) = log(\frac{e^{s(x,y)}}{\sum_{\tilde{y} \in Y_x} e^{s(x,\tilde{y})}}) = s(x,y) - log(\sum_{\tilde{y} \in Y_x} e^{s(x,\tilde{y})}) \quad \text{(公式3)}$$

由上**公式3**转化成了两部分，s(x,y)容易求得而log_sum_exp(**后面记为LSE**)的求解就需要用到前向算法进行迭代计算，loss可定义为：-log(p(y|x))。

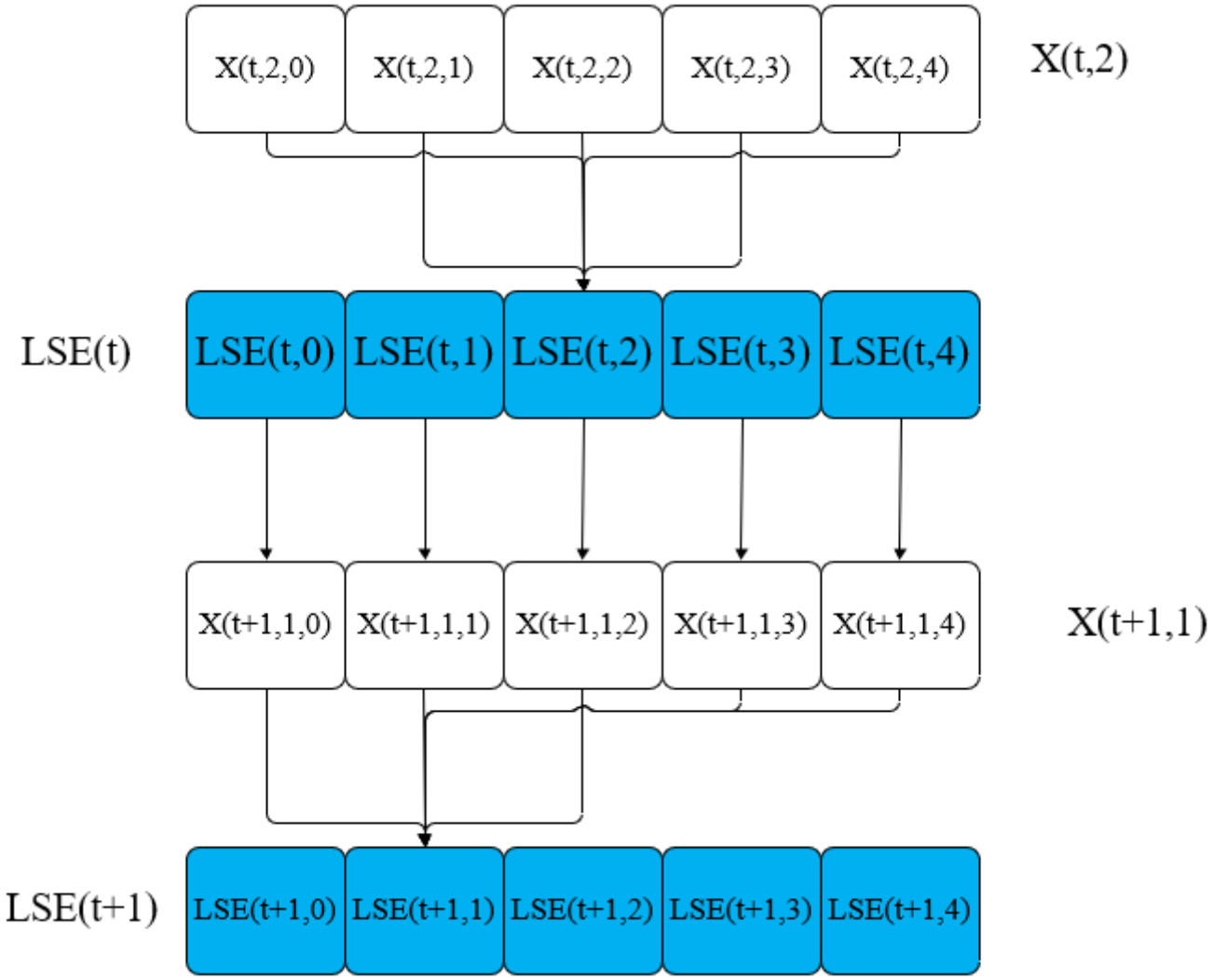

图2 CRF概率计算的前向算法图示

在此强烈建议回头复习下公式1中score的定义。上图中LSE(t,0)代表考虑输入序列前t个分量且timestep=t时取tag0的LSE，LSE(t)就是timestep=t时取所有可能tag对应的向量。而要理解X(t+1,1)就需结合LSE(t)和LSE(t+1)，比如其分量X(t+1,1,0)代表LSE(t,0)+A(0,1)+B(t+1,1)。然后我们来看一下LSE的递推公式(**公式4**)是如何得出的：

$$log\sum_{j=1}^{5}e^{X(t+1,s,j)} = log\sum_{j=1}^{5}\sum_{k=1}^{5}e^{X(t,j,k)+y(t,j,s)}$$

$$= log\sum_{j=1}^{5}[exp((log\sum_{k=1}^{5}e^{X(t,j,k)}) + y(t,j,s))]$$

注意：上式中的y(t,j,s)是个简写，代表LSE(t,j)到X(t+1,s,j)需要的增量(转移+发射)。另外上面我们定义的LSE(t)就是下面代码中的forward_var。X是公式1中的score，X(t+1,s,j)代表ts=t时标注为tagj且ts=t+1时标注为tags对应的非规一化score

终于到了代码部分，好期待(假设大家已熟悉LSTM，不然请移步 追梦人：LSTM原理及实战)：

```
# Compute log sum exp in a numerically stable way for the forward algorithm
# 最后return处应用了计算技巧，目的是防止sum后数据过大越界，实际就是对vec应用log_sum_exp
def log_sum_exp(vec):   # vec size=1*tag_size
    max_score = vec[0, argmax(vec)]
    max_score_broadcast = max_score.view(1, -1).expand(1, vec.size()[1])
    return max_score + \
        torch.log(torch.sum(torch.exp(vec - max_score_broadcast)))

# 此处基于前向算法，计算输入序列x所有可能的标注序列对应的log_sum_exp，同时可参考上述LSE的说明.
def _forward_alg(self, feats):
    # Do the forward algorithm to compute the partition function
    init_alphas = torch.full((1, self.tagset_size), -10000.)
    # START_TAG has all of the score.
    init_alphas[0][self.tag_to_ix[START_TAG]] = 0.

    # Wrap in a variable so that we will get automatic backprop
    # forward_var初值为[-10000.,-10000.,-10000.,0,-10000.] 其数值是非规一化概率或者crf计算
    forward_var = init_alphas

    # Iterate through the sentence，如下遍历一个句子的各个word或者step
    for feat in feats:
        alphas_t = []  # The forward tensors at this timestep，alphas_t即是上述定义的LSE
        for next_tag in range(self.tagset_size):  # 此处遍历step=t时所有可能的tag
            # broadcast the emission score: it is the same regardless of
            # the previous tag
            # 建议先看trans_score再看emit_score,对所有tag转移到next_tag而言其对应发射概率相
            emit_score = feat[next_tag].view(
                1, -1).expand(1, self.tagset_size)
            # the ith entry of trans_score is the score of transitioning to
            # next_tag from i
            # 从每一个tag转移到next_tag对应的转移score
            trans_score = self.transitions[next_tag].view(1, -1)  # trans_score's size
            # The ith entry of next_tag_var is the value for the
            # edge (i -> next_tag) before we do log-sum-exp
            # 如下加法是三个向量相加，相同槽位的数值直接相加即可
            next_tag_var = forward_var + trans_score + emit_score
            # The forward variable for this tag is log-sum-exp of all the
            # scores.
            alphas_t.append(log_sum_exp(next_tag_var).view(1))  # 此处相当于LSE(t,next_
        forward_var = torch.cat(alphas_t).view(1, -1)  # 此处生成完整的LSE(t),其size=1*t
```

2.参数学习

本文着重从实践层面进行解读，在pytorch中只要定义好loss后就可实现自动求导，相关代码如下(参数学习的理论后续再议)：

```
# 极大取相反数作为loss
def neg_log_likelihood(self, sentence, tags):
    feats = self._get_lstm_features(sentence)
    forward_score = self._forward_alg(feats)
    gold_score = self._score_sentence(feats, tags)
    return forward_score - gold_score

# Make sure prepare_sequence from earlier in the LSTM section is loaded
for epoch in range(300):  # again, normally you would NOT do 300 epochs, it is toy dat
    for sentence, tags in training_data:
        # Step 1. Remember that Pytorch accumulates gradients.
        # We need to clear them out before each instance
        model.zero_grad()

        # Step 2. Get our inputs ready for the network, that is,
        # turn them into Tensors of word indices.
        sentence_in = prepare_sequence(sentence, word_to_ix)
        targets = torch.tensor([tag_to_ix[t] for t in tags], dtype=torch.long)

        # Step 3. Run our forward pass.
        loss = model.neg_log_likelihood(sentence_in, targets)

        # Step 4. Compute the loss, gradients, and update the parameters by
        # calling optimizer.step()
        loss.backward()    # 反向求导
        optimizer.step()   # 参数更新
```

3.序列预测

如果理解了前述有关概率计算部分，那这部分就如砍瓜切菜般容易。只需记住如下几点：

- 采用viterbi算法给输入序列寻找最可能的标注序列。
- 计算过程中，一方面保留最优子序列的非规一化概率，同时还要保留走过的路径。

```
def _viterbi_decode(self, feats):
    backpointers = []

    # Initialize the viterbi variables in log space
    init_vvars = torch.full((1, self.tagset_size), -10000.)
    init_vvars[0][self.tag_to_ix[START_TAG]] = 0

    # forward_var at step i holds the viterbi variables for step i-1
    forward_var = init_vvars
    for feat in feats:
        bptrs_t = []  # holds the backpointers for this step, timestamp=t时每个tag对应的
        viterbivars_t = []  # holds the viterbi variables for this step, timestamp=t时

        for next_tag in range(self.tagset_size):
            # next_tag_var[i] holds the viterbi variable for tag i at the
            # previous step, plus the score of transitioning
            # from tag i to next_tag.
            # We don't include the emission scores here because the max
            # does not depend on them (we add them in below)
            next_tag_var = forward_var + self.transitions[next_tag]
            best_tag_id = argmax(next_tag_var)
            bptrs_t.append(best_tag_id)
            viterbivars_t.append(next_tag_var[0][best_tag_id].view(1))
        # Now add in the emission scores, and assign forward_var to the set
        # of viterbi variables we just computed
        # 此处两个1*tag_size的向量相加，这样就得到timestamp=t时每个tag对应的完整最大score
        forward_var = (torch.cat(viterbivars_t) + feat).view(1, -1)
        backpointers.append(bptrs_t)

    # Transition to STOP_TAG
    terminal_var = forward_var + self.transitions[self.tag_to_ix[STOP_TAG]]
    best_tag_id = argmax(terminal_var)
    path_score = terminal_var[0][best_tag_id]

    # Follow the back pointers to decode the best path.
    best_path = [best_tag_id]
    # 对backpointers的遍历顺序变为从T->1,而对每个t来说各tag对应的前一步最优节点数组(bptrs_t)
    for bptrs_t in reversed(backpointers):
        best_tag_id = bptrs_t[best_tag_id]
        best_path.append(best_tag_id)
    # Pop off the start tag (we dont want to return that to the caller)
    start = best_path.pop()
```

说明：如果各位老板对本文有疑问或者有错误的地方欢迎一起交流讨论。

参考资料

1、PyTorch官方代码：pytorch.org/tutorials/b...。

2、Pytorch BiLSTM：zhuanlan.zhihu.com/p/59...。

3、Bi-LSTM+CRF paper：arxiv.org/pdf/1508.0199...。

4、BiLSTM模型中CRF层的运行原理：BiLSTM模型中CRF层的运行原理-1。

编辑于 2020-05-16

PyTorch LSTM CRF

文章被以下专栏收录

NLP大揭秘

推荐阅读

LSTM+CRF 解析（代码篇）

最近在搞信息抽取任务，用到了LSTM+CRF模型，之前没有深入了解过，就趁这次好好总结一下。把所有的代码，文章看完一遍后发现，这个LSTM+CRF和一般的CRF还是有点区别的，以及具体的代...

lstm学习笔记及使用代码

写在前面：终于碰，总算对楚，想写点后自己复习正。 本篇bl

6 条评论

切换为时间排序

写下你的评论...

椰子树 03-25

写的很好

赞

Katkit Wu 02-23

you save my day

赞

123 2020-06-13

请教下，状态转移矩阵在代码里是随机初始化的且固定不变的?

赞

建国 回复 123 2020-12-07

刚开始是随机初始化的，之后肯定要训练优化参数而改变

赞

行之 2020-06-09

你好，请问对于数据集的bieo标注是手动进行依次标注吗?

赞

追梦人 (作者) 回复 行之 2020-06-10

序列标注问题一般是有监督模型效果更好，所以肯定是需要手动标注的。不过手动标注未必完全从raw数据开始，你可以用无监督模型生成一版结果再人工review，造成以后生成用于有监督训练的标注样本。

赞