模型先计算 loss，求 negative log likelihood，调用函数 neg_log_likelihood

在此函数中，先用 lstm 获得 emission probability，调用函数 get_lstm_features

get_lstm_features 详解

```
def _get_lstm_features(self, sentence):
    self.hidden = self.init_hidden()
    embeds = self.word_embeds(sentence).view(len(sentence), 1, -1)
    lstm_out, self.hidden = self.lstm(embeds, self.hidden)
    lstm_out = lstm_out.view(len(sentence), self.hidden_dim)
    lstm_feats = self.hidden2tag(lstm_out)
    return lstm_feats
```

假设句子长度为 10，word embedding 为 30，

embeds 为 (10, 1, 30)，lstm 需要输入为 3 维，

lstm_out 为 (10, hidden_dim)

lstm_feats 为 (10, tags)，此为由 lstm 获得的特征

lstm_feats：→ 每一个 tag 到此 word 的 emission probability，注意：与 HMM 不同，此值为广义范围，具体说为 tags 与当前 word 的关系。

10 × tags

return feats (10 × tags)

将 feats 传入 forward_alg 中，计算所有 exp 的和

forward_alg 详解

```
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 = init_alphas

    # Iterate through the sentence
    for feat in feats:
        alphas_t = []  # The forward tensors at this timestep
        for next_tag in range(self.tagset_size):
            # broadcast the emission score: it is the same regardless of
            # the previous tag
            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
            trans_score = self.transitions[next_tag].view(1, -1)
            # The ith entry of next_tag_var is the value for the
            # edge (i -> next_tag) before we do log-sum-exp
            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))
        forward_var = torch.cat(alphas_t).view(1, -1)
    terminal_var = forward_var + self.transitions[self.tag_to_ix[STOP_TAG]]
    alpha = log_sum_exp(terminal_var)
    return alpha
```

init_alphas：初始 tag 概率 [-10000, -10000, 0, -10000 ...] → START

设为 -10000，因为初始只能为 START

forward → init_alphas 不懂为什么要指向

for feat in feats：循环每个单词 [ ] 1 × tags

alphas_t = [ ]

for next_tag in tags：循环每个 tag，接着对

emit_score = feat[某tag] → [同样数值] 1 × tags

tag ↔ word → 这个 tag 和这个 word 的关系

trans_score = [ ] 1 × tags

任意 tags → 此 tag 的概率

next_tag_var = forward + trans_score + emit_score.

[-10000 -10000 1 -10000 -10000]

tags → tag [100 200 3 1 5]

tag → word [100 100 100 100 100]

alphas_t = [ ] ← log_sum 1 × tags

因为 tag → tag 怎么配对，都要加 emission score.

```
def log_sum_exp(...):
    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)))
```

插入 log_sum 函数，讲解一下

先上公式：$P(y|x) = \frac{P(x,y)}{\sum_y P(x,y)}$ 取 log

$\log P(y|x) = \log P(x,y) - \log \sum_y P(x,y)$

vec中是上面相加结果，第一次 alphas_t 只有 START 权重最高，减去最大值，防止溢出。

$P(x,y) = \exp(w \cdot \phi(x,y))$

因为后期我们要用 negative -log -likelihood，故取负号

$-\log P(y|x) = (\log \sum_{y} P(x,y)) - \log P(x,y)$

把所有可能的 tag 循环完后，我们得到对于每个 tag 可能的结果，是 tags 个分数

alphas_t = [ ] 1x tags

↓ 到此tag的所有可能的 log_sum

跳出循环：forward 更新为 alphas_t，即保留之前的 path，画个图实例下

最初的 forward [t1 t2 START]

next_tag_var [t1 t2 START]

alphas_t [t1 t2 START] ← 新的forward

算下个单词，还是那样，但是forward变了

最后，加上STOP，返回分数

将 feats, tags 传入 _score_sentence 中

```
def _score_sentence(self, feats, tags):
    # Gives the score of a provided tag sequence
    score = torch.zeros(1)
    tags = torch.cat([torch.tensor([self.tag_to_ix[START_TAG]], dtype=torch.long), tags])
    for i, feat in enumerate(feats):
        score = score + \
            self.transitions[tags[i + 1], tags[i]] + feat[tags[i + 1]]
    score = score + self.transitions[self.tag_to_ix[STOP_TAG], tags[-1]]
    return score
```

i+1为此tag，i为上个tag，feat[i+1]为当前tag与word关系

这里的score为什么不取log

需要解答 → 因为 log(exp) 约掉了，上面 log_sum_exp 函数，其实也能写，但是为了直观，重写出来

训练完成后，我们采用 viterbi 算法解码

```
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
        viterbivars_t = []  # holds the viterbi variables for this step

        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
        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]
    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()
    assert start == self.tag_to_ix[START_TAG]  # Sanity check
    best_path.reverse()
    return path_score, best_path
```

next_tag_var = [forward]

t1 t2 t3

[t1] transition

bptrs_t best_tag_id 哪个tag会传到此tag最大分数

viterbivars_t

到t1最大值

[t1 t2 t3] viterbivars_t } forward

[t1 t2 t3] feat

↓ t1到此word的emission

backpointers [[ ]]

到t1最好tag ↘ 到t1最好tag

transiton to STOP_TAG

得到最后一个tag到 STOP_TAG 的值，求最大，就为最后一个tag

[[ ], [ ], [ ]]

↓ 到最后一个tag的最好之前tag

最后最好tag为 t3

取出来到 t3 的最好tag，eg. t2

取到 t2 的最好 tag

written by Jiaxin Zhang
Please talk with me if you have any question or find any problem