LSTM+CRF 整体流程与相关变量：

变量：

X_train:训练集句子：EU rejects German call to boycott British lamb .

Y_train:训练集句子单词的实体类型：B-ORG O B-MISC O O O B-MISC O O

X_test:测试集句子：SOCCER - JAPAN GET LUCKY WIN , CHINA IN SURPRISE DEFEAT .

Y_test:测试集句子单词的实体类型：O O B-LOC O O O O B-PER O O O O

Word2idx:训练集+测试集单词数组：{'Boat': 0, 'Standings': 1, ... , '<pad>': 27316}

Vocab_size:训练集+测试集单词个数：27317

Tag2idx:训练集+测试集实体类型数组：{'I-ORG': 0, 'I-MISC': 1, ... , '<pad>': 11}}

Max_length:句子最长的长度：设置为 124

Dataset:

Inputs:句子单词在 word2idx 中的编号：[386,10193,24516,14669,24332,17873,9648,21165,11724, 27316,27316,27316, ... , 27316]:一组里面有 max_length 个数值，用<pad>的编号填充

Targets:句子单词的实体类型在 tag2idx 中的编号：[1,2,7,2,2,2,7,2,2,11,11,11, ... ,11]:一组里面有 max_length 个数值，用<pad>的编号填充

Length_list:每个句子的真实长度：[9, 2, 2, 30, ... , 4]

Dataloder:以 batch_size 为一组分割 Dataset

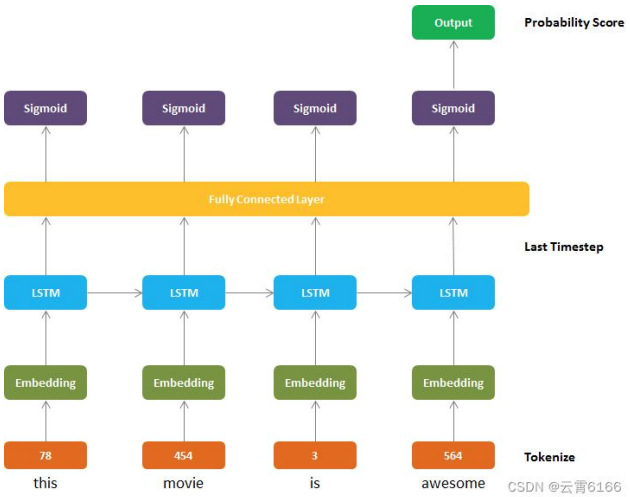
batch_size:一次训练所抓取的数据样本数量：论文里设置为 100

embedding_size:特征向量的大小:论文里设置为 50

hidden_size:隐藏层：论文里设置为 300

Epochs:循环次数：论文里设置为 10

整体流程：



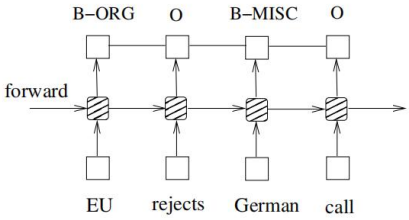


Figure 6: A LSTM-CRF model.

1. 构造 dataloder：

Dataloder 每一块为 100 行(=batch size)和 124 列(=sequence length)，共有 141 块（根据具体数据计算）.

Dataloder 作为嵌入层的输入.

2. Embedding Layer：nn.Embedding

self.embedding = nn.Embedding(vocab_size(=词汇表大小), embedding_size(=嵌入维数，把具体数值改成 embedding_size 维向量)).

embeds = self.embedding(sentences)

从嵌入层的输出可以看出，它作为嵌入权值的结果创建了一个三维张量。现在它每块有 100 行，124 列和 50 个嵌入维，也就是说，在我们的审查中，我们为每个标记化的单词添加了嵌入维。该数据现在将进入 LSTM 层.

torch.nn.Embedding(

num_embeddings, – 词典的大小尺寸，比如总共出现 5000 个词，那就输入 5000。此时 index 为（0-4999）

embedding_dim,– 嵌入向量的维度，即用多少维来表示一个符号。

padding_idx=None,– 填充 id，比如，输入长度为 100，但是每次的句子长度并不一样，后面就需要用统一的数字填充，而这里就是指定这个数字，这样，网络在遇到填充 id 时，就不会计算其与其它符号的相关性。（初始化为 0）

max_norm=None, – 最大范数，如果嵌入向量的范数超过了这个界限，就要进行再归一化。

norm_type=2.0, – 指定利用什么范数计算，并用于对比 max_norm，默认为 2 范数。

scale_grad_by_freq=False, 根据单词在 mini-batch 中出现的频率，对梯度进行放缩。默认为 False.

sparse=False, – 若为 True,则与权重矩阵相关的梯度转变为稀疏张量。

_weight=None)

3. LSTM Layer：nn.LSTM

self.lstm = nn.LSTM(embedding_size(=嵌入维数), hidden_size(=隐藏层数), bidirectional=False(=选择是否双向))

lstm_out, _ = self.lstm(packed_sentences)

通过查看 LSTM 层的输出，我们可以看到每块现在有 100 行，124 列和 300 个 LSTM 节点。接下来，该数据被提取到全连接层。

torch.nn.LSTM(

input_size 输入数据的特征维数，通常就是 embedding_dim(词向量的维度)

hidden_size LSTM 中隐层的维度

num_layers 循环神经网络的层数

bias 用不用偏置，default=True

batch_first 这个要注意，通常我们输入的数据 shape=(batch_size,seq_length,embedding_dim),而 batch_first 默认是 False,所以我们的输入数据最好送进 LSTM 之前将 batch_size 与 seq_length 这两个维度调换

dropout 默认是 0，代表不用 dropout

bidirectional 默认是 false，代表不用双向 LSTM)

输入数据包括 input,(h_0,c_0):

input 就是 shape=(seq_length,batch_size,input_size)的张量

*h_0 是 shape=(num_layers*num_directions,batch_size,hidden_size)的张量，它包含了在当前这个 batch_size 中每个句子的初始隐藏状态。其中 num_layers 就是 LSTM 的层数。如果 bidirectional=True,num_directions=2,否则就是 1，表示只有一个方向。*

c_0 和 h_0 的形状相同，它包含的是在当前这个 batch_size 中的每个句子的初始细胞状态。h_0,c_0 如果不提供，那么默认是 0。

输出数据包括 output,(h_n,c_n):

*output 的 shape=(seq_length,batch_size,num_directions*hidden_size),*

它包含的是 LSTM 的最后一时间步的输出特征(h_t), t 是 batch_size 中每个句子的长度。

*h_n.shape==(num_directions * num_layers,batch,hidden_size)*

c_n.shape==h_n.shape

h_n 包含的是句子的最后一个单词（也就是最后一个时间步）的隐藏状态，c_n 包含的是句子的最后一个单词的细胞状态，所以它们都与句子的长度 seq_length 无关。

output[-1]与 h_n 是相等的，因为 output[-1]包含的正是 batch_size 个句子中每一个句子的最后一个单词的隐藏状态，注意 LSTM 中的隐藏状态其实就是输出，cell state 细胞状态才是 LSTM 中一直隐藏的，记录着信息

4. Fully Connected Layer:nn.Linear

self.hidden_to_tag = nn.Linear(hidden_size(=隐藏层数), self.target_size(=实体类型个数，为 12))

feature = self.hidden_to_tag(result)

Feature.shape = torch.Size([100, 124, 12])

对于全连通层，输入特征数= LSTM 中隐藏单元数。输出大小= 实体类型个数

torch.nn.Linear(in_feature： int 型, 在 forward 中输入 Tensor 最后一维的通道数,

out_feature： int 型, 在 forward 中输出 Tensor 最后一维的通道数,

bias: bool 型， Linear 线性变换中是否添加 bias 偏置)

5. CRF Layer：CRF

self.crf = CRF(self.target_size(=实体类型个数),batch_first=True)

self.crf(self.LSTM_Layer(sentences, length_list), targets, self.get_mask(length_list))

假设 LSTM 节点的输出是 1.5 (B-Person), 0.9 (I-Person), 0.1 (B-Organization), 0.08 (I-Organization) and 0.05 (O)。这些分数将会是 CRF 层的输入。所有的经 LSTM 层输出的分数将作为 CRF 层的输入，类别序列中分数最高的类别就是我们预测的最终结果。

CRF(num_tags, batch_first=True)

self.crf.decode(self.LSTM_Layer(sentences, length_list), self.get_mask(length_list))

CRF.decode(emissions, mask=None):使用 Viterbi algorithm 找到概率最大的实体类型

Parameters:

emissions (Tensor) – Emission score tensor of size if is False, otherwise.(seq_length, batch_size, num_tags)batch_first(batch_size, seq_length, num_tags)

mask (ByteTensor，0、1 向量) – Mask tensor of size if is False, otherwise.(seq_length, batch_size)batch_first(batch_size, seq_length)

Return type:List[List[int]]

Returns:List of list containing the best tag sequence for each batch(注意：这个 decode 返回的是一个 List，由于 mask 的存在，解码返回的是实际的句子长度的解码结果).

6. 优化器与损失函数

optimizer = torch.optim.Adam(model.parameters(), lr)#构造优化器

model.zero_grad()#梯度初始化为 0

loss = (-1) * model(inputs, length_list, targets)#构造损失函数

loss.backward()#反向传播（计算梯度）

total_norm = torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm)#裁剪梯度

optimizer.step()#更新网络参数

7. 其他小知识点

①pack_padded_sequence()与 pad_packed_sequence()：

数据进入 LSTM 层之前要 pack_padded_sequence:包装数据

数据离开 LSTM 层之前要 pack_packed_sequence:解压数据

https://blog.csdn.net/xinjieyuan/article/details/108562360

②batch_first 的使用

LSTM 默认 batch_first=False，即默认 batch_size 这个维度是在数据维度的中间的那个维度，即喂入的数据为【seq_len, batch_size, hidden_size】这样的格式。此时

lstm_out:【seq_len, batch_size, hidden_size * num_directions】

lstm_hn:【num_directions * num_layers, batch_size, hidden_size】

当设置 batch_first=True 时，喂入的数据就为【batch_size, seq_len, hidden_size】这样的格式。此时

lstm_out:【 batch_size, seq_len, hidden_size * num_directions】

lstm_hn:【num_directions * num_layers, batch_size, hidden_size】

https://blog.csdn.net/qq_52785473/article/details/124368762

③get_mask 的使用

Mask 机制就是我们在使用不等长特征的时候先将其补齐，在训练模型的时候再将这些参与补齐的数去掉，从而实现不等长特征的训练问题。

https://blog.csdn.net/Jeaten/article/details/105011214

8. 引用网址

源代码：https://blog.csdn.net/Raki_J/article/details/122435674

流程解读：https://blog.csdn.net/yinwen1999/article/details/125894923

CRF 层详解：https://zhuanlan.zhihu.com/p/44042528