CS 533: Natural Language Processing **(Due: 03/31/20)**

Assignment 4

Name: Keya Desai, *NetID:* kd706 *Students discussed with:*

Problem 1: HMM

((1 + 6 + 6 = 13 points))

1. Non-zero MLE parameters values of (o, t) are as follows:

   Emission probabilities $o(x|y)$:

$o(x\|y)$	Emission Probability
$o(the\|D)$	1.0
$o(cut\|N)$	0.167
$o(man\|N)$	0.33
$o(saw\|N)$	0.33
$o(the\|N)$	0.167
$o(cut\|V)$	0.5
$o(saw\|V)$	0.5

   Transition probabilities $t(x|y)$:

$t(y^{'}\|y)$	Transition probability
$t(D\|*)$	0.67
$t(N\|*)$	0.33
$t(D\|V)$	1.0
$t(N\|D)$	1.0
$t(V\|N)$	0.33
$t(N\|N)$	0.167
$t(STOP\|N)$	0.5

   Non-zero emission probabilities for **cut** are:

$$\boxed{\begin{array}{l} o(cut|N) = 0.167 \\ o(cut|V) = 0.5 \end{array}}$$

2. Probability under the HMM that the third word is tagged with V conditioning on $x^{(2)}$ is:

$$\begin{aligned} \sum_{(y1,y2,y3,y4,y5)\in Y:y3=V} & p(y1, y2, y3, y4, y5|\text{the saw cut the man}) \\ &= \alpha(3, V) * \beta(3, V) \\ &= 0.03755 * 0.1667 \\ &= 0.00626 \end{aligned}$$

3. Probability that the fifth word is tagged with N conditioning on $x^{(1)}$ is:

$$\begin{aligned} \sum_{(y1,y2,y3,y4,y5)\in Y:y5=N} & p(y1, y2, y3, y4, y5|\text{the man saw the cut}) \\ &= \alpha(5, N) * \beta(5, N) \\ &= 0.00626 * 0.5 \\ &= 0.00313 \end{aligned}$$

Problem 2: PCFG ((1 + 6 + 6 = 13 points))

1. Non-zero MLE parameter values of (u, b) estimated from the given corpus are:
   Unary Rule Probabilities

$u(x\|y)$	Probability
$u(the\|D)$	0.67
$u(a\|D)$	0.33
$u(saw\|V)$	1.0
$u(with\|P)$	1.0
$u(boy\|N)$	0.33
$u(man\|N)$	0.33
$u(telescope\|N)$	0.33

   Binary Rule Probabilities

$b(X \rightarrow YZ)$	Probability
$b(S \rightarrow NP, VP)$	1.0
$b(PP \rightarrow P, NP)$	1.0
$b(NP \rightarrow D, N)$	0.857
$b(NP \rightarrow NP, PP)$	0.143
$b(VP \rightarrow VP, PP)$	0.33
$b(VP \rightarrow V, NP)$	0.67

2. Probability under the PCFG that **NP spans (4, 8)** conditioning on x is:

$$\begin{aligned}\sum_{\tau \in GEN(x): root(\tau,4,8)=NP} & p(\tau|x) \\ &= \alpha(4,8,NP) * \beta(4,8,NP) \\ &= 0.00259 * 0.12698 \\ &= 0.000329\end{aligned}$$

3. Probability under the PCFG that **VP spans (3, 5)** conditioning on x is:

$$\begin{aligned}\sum_{\tau \in GEN(x): root(\tau,3,5)=VP} & p(\tau|x) \\ &= \alpha(3,5,VP) * \beta(3,5,VP) \\ &= 0.12698 * 0.00605 \\ &= 0.000768\end{aligned}$$

Problem 3: CRF ((1 + 1 + 1 + 1 + 1 + 1 + 8 + 8 = 22 points))

1. (a) The sequences are sorted by length so that each batch gets sequences of same length. This eliminates the need for padding of sequences in the same batch.

   (b) No. Setting batch size N ensures that number of sequences in each batch is $\leq$ N. The sequences are sorted by length and sequence of same length are put in a batch. If number of sequence with same length is less than N, then batch size is less than N.

   (c) No. There is no padding at word sequence level. Since each batch has same length words, padding is not required.

   (d) Characters in a batch are stored as a list of LongTensors - *cseqs*. Each LongTensor corresponds to a word's character mapped to index by char2c.

   (e) Yes there is padding at the character sequence level:

   ```
   C = pad_sequence(flattened_cseqs, padding_value=self.PAD_ind,
       batch_first=True).to(self.device)
   ```

2. output['acc'] is calculated as follows:
   Number of correct predictions and total number of predictions is calculated.

   ```
   num_preds += B * T
   num_correct += (preds == Y).sum().item()
   ```

   Then accuracy is calculated by dividing number of correct predictions by total number of predictions:

   ```
   output = {'acc': num_correct / num_preds * 100}
   ```

   output[$'f1\_ < all >'$] is calculated as follows:
   Using predicted and Y tags, gold_boundaries and pred_boundaries are computed. If an entry of gold_boundaries -(s, t, entity) - is present in pred_boundaries then, true positive(tp) is incremented else false negative(fn) is incremented. Entries which are present in pred_boundaries but not in gold_boundaries are counted towards false positives(fp). Using tp, fn and fp, precision, recall and f1 scores are calculated as follows:

$$\begin{aligned} \text{Precision} &= \frac{100 * tp}{tp + fp} \\ \text{Recall} &= \frac{100 * tp}{tp + fn} \\ \text{f1} &= \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \end{aligned} \tag{1}$$

3. The loss for a single data point (x^i, h^i, y^i) is given as follows

$$\text{loss}(x^i, h^i, y^i) = -\log\left(\frac{\exp(h^i[y^i])}{\sum_{j=1}^{L}\exp(h^i[j])}\right)$$

   The loss of each data point in the batch is averaged out:

   ```
   loss = self.avgCE(scores, targets)
   ```

   Hence, Greedy loss is given as follows:

$$\text{Greedy Loss} = \frac{\sum_{i \in Batch}\text{loss}(x^i, h^i, y^i)}{\text{Batch Size}}$$

4. (a) Character-level embedding *self.cemb* is initialised in BiLSTMTagger using *nn.Embedding* which takes the number of unique characters and the character emdedding dimension *dim_char* as parameters. This is sent to *BiLSTMOverCharacters*.

   (b) The character level embedding is concatenated with the word embedding in the forward function. Hence, the final word embedding size is **wdim + 2 * cdim.**

5. (a) The parameters of this layers are: $self.start$, $self.T$ and $self.end$

   (b) The formula for the score is given as:

$$\text{score}(h,\ y) = \sum_{i=1}^{T} h_i[y_i] + \text{self.start}[y_0] + \sum_{i=2}^{T-1} \text{self.T}[y_i, y_{i-1}] + \text{self.end}[y_T]$$

   (c) loss (computed in forward) as a function of the label sequence scores is given as:
   loss = $Mean$(normalizers − target_scores)

6. (a) *compute_normalizers_brutes* generates all possible combination of target sequence and calculates the sequence score for input sequence and this target sequence. The sequence scores are appended to a list which is stacked at the end. The final normalised score is calculated by taking log of the summation of exponential of each of the scores. Dimension of the normalised score = B (no. of batches).

   decode_brute similarly iterates over all the possible target sequences, computes the sequence score but also stores the target sequence. Maximum score is returned from the normalised scores calculated along with the corresponding target sequence.

   (b) **Computational Complexity:**
   Both functions iterate over all possible combinations of target sequences. Number of possible combinations = L^T. In each iteration *score_targets* function is called which takes $O(T)$ time. Hence, overall complexity of both the functions = $O(|L|^T T)$.

7. **Forward algorithm.** Implementation of $compute_normalisers$ with complexity $O(|L|^2 T)$ is:

```
def compute_normalizers(self, scores):
    B, T, L = scores.size()
    scores = scores.transpose(0, 1) # (T x B x L)
    prev = self.start + scores[0]  # (B x L)

    for i in range(1, T):
        # (B x L x 1) + (L x L) + (B x 1 x L) = (B x L x L)
        prev = torch.logsumexp(prev.unsqueeze(2) + self.T.transpose(0, 1)
                + scores[i].unsqueeze(1), dim=1)  # B x L

    prev += self.end
    normalizers = torch.logsumexp(prev, dim=1)  # (B)

    return normalizers
```

8. **Viterbi Algorithm.** Implementation of *decode* with complexity $O(|L|^2 T)$ is:

```
def decode(self, scores):
    B, T, L = scores.size()
    scores = scores.transpose(0, 1) # (T x B x L)
    prev = self.start + scores[0]   # (B x L)
    back = []
    for i in range(1, T):
        prev, indices = (prev.unsqueeze(2) + self.T.transpose(0, 1)
                         + scores[i].unsqueeze(1)).max(dim=1)
        back.append(indices)
    prev += self.end

    max_scores, indices = prev.max(dim=1)
    tape = [indices]
    back = list(reversed(back))
    for i in range(T - 1):
        indices = back[i].gather(1, indices.unsqueeze(1)).squeeze(1)
        tape.append(indices)

    return max_scores, torch.stack(tape[::-1], dim=1)
```

Test is passed in *test_decode* in *test_crf.py* for both the questions:

```
Keyas-MacBook-Pro:code keyadesai$ python test_crf.py
...
----------------------------------------------------------------------
Ran 3 tests in 1.010s

OK
```