Assignment 4

Instructor: Karl Stratos

- 3 problems: total 48 points (13 + 13 + 22)
- No collaboration
- Due by 11:59pm of the due date, no late submission accepted
- Use the provided LaTeX assignment template to write the answers. Upload the code as well.

**Tip.** For Problem 1 and 2, it will be easier to write a small program that computes all probabilities than to manually calculate the probabilities in question.

**Problem 1: HMM**

We consider a bigram HMM \((o, t)\) that has emission probabilities \(o(x|y)\) for all word types \(x \in V\) and tag types \(y \in Y\) and transition probabilities \(t(y'|y)\) for all \(y \in Y \cup \{*\}\) and \(y \in Y \cup \{\text{STOP}\}\) where * and STOP are special tags for the beginning and the end of a sequence. It defines the probability of \((x_1 \ldots x_n, y_1 \ldots y_n) \in V^n \times Y^n\) by (letting \(y_0 = *\))

\[
p(x_1 \ldots x_n, y_1 \ldots y_n) = t(y_1|*) \times \left( \prod_{i=1}^{n} t(y_i|y_{i-1}) \times o(x_i|y_i) \right) \times t(\text{STOP}|y_n)
\]

1. Assume the following data of tagged sequences \((x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)})\)

   - \(x^{(1)} = \text{the man saw the cut}\)
   - \(y^{(1)} = D\ N\ V\ D\ N\)
   - \(x^{(2)} = \text{the saw cut the man}\)
   - \(y^{(2)} = D\ N\ V\ D\ N\)
   - \(x^{(3)} = \text{the saw}\)
   - \(y^{(3)} = N\ N\)

Write all nonzero MLE parameter values of \((o, t)\) estimated from this corpus. In particular, what are the nonzero emission probabilities for the word cut (i.e., possible tag types)?

2. Calculate the probability under the HMM that the third word is tagged with \(V\) conditioning on \(x^{(2)}\) by running the forward and backward algorithms. That is, calculate

\[
\sum_{(y_1, y_2, y_3, y_4, y_5) \in Y^5: y_3 = V} p(y_1, y_2, y_3, y_4, y_5 | \text{the saw cut the man})
\]

as a function of forward and backward probabilities. Round to 5 decimal places if necessary.

3. Similarly, calculate the probability that the fifth word is tagged with \(N\) conditioning on \(x^{(1)}\):

\[
\sum_{(y_1, y_2, y_3, y_4, y_5) \in Y^5: y_5 = N} p(y_1, y_2, y_3, y_4, y_5 | \text{the man saw the cut})
\]

**Problem 2: PCFG**

1 + 6 + 6 = 13 points
We consider a PCFG in CNF \((u,b)\) that has unary rule probabilities \(u(X \rightarrow x|X)\) for all nonterminal types \(X\) and terminal types \(x\) and binary rule probabilities \(b(X \rightarrow Y Z|X)\) for all nonterminal types \(X,Y,Z\). It defines the probability of the parse tree as a product of the probabilities of the rules occurred in the tree. For instance, the first tree in the data described below is assigned the probability

\[
\begin{align*}
u(D \rightarrow \text{the})^2 \times u(D \rightarrow a|D) \times u(N \rightarrow \text{boy}|N) \times u(N \rightarrow \text{man}|N) \times u(N \rightarrow \text{telescope}|N) \times u(V \rightarrow \text{saw}|V) \\
\times u(P \rightarrow \text{with}|P) \times b(S \rightarrow NP \ VP|S) \times b(NP \rightarrow D \ N|NP)^3 \times b(NP \rightarrow NP \ PP|NP) \times b(PP \rightarrow P \ NP|PP) \\
\times b(VP \rightarrow V NP|VP)
\end{align*}
\]

1. Assume the following data of 2 parses for sentence \(x = (\text{the boy saw the man with a telescope})\):

- First parse:
  - S
    - NP
      - D
        - the
      - N
        - boy
    - VP
      - V
        - saw
      - NP
        - D
          - the
        - N
          - man
      - PP
        - P
          - with
        - NP
          - D
            - a
          - N
            - telescope

- Second parse:
  - S
    - NP
      - D
        - the
      - N
        - boy
    - VP
      - V
        - saw
      - NP
        - D
          - the
        - N
          - man
      - PP
        - P
          - with
        - NP
          - D
            - a
          - N
            - telescope

Write all nonzero MLE parameter values of \((u,b)\) estimated from this corpus.

2. Calculate the probability under the PCFG that NP spans \((4,8)\) (i.e., “the man with a telescope”) conditioning on \(x\) by running the inside and outside algorithms. That is, calculate

\[
\sum_{\tau \in \text{GEN}(x): \text{root}(\tau,4,8)=NP} p(\tau|x)
\]

(where \(\text{GEN}(x)\) is the set of all possible parses and \(\text{root}(\tau,i,j)\) is the nonterminal at the root of a subtree spanning \((i,j)\)) as a function of inside and outside probabilities.

3. Similarly, calculate the probability that VP spans \((3,5)\) (i.e., “saw the man”) conditioning on \(x\)

\[
\sum_{\tau \in \text{GEN}(x): \text{root}(\tau,3,5)=VP} p(\tau|x)
\]

**Problem 3: Programming (CRF)**

(1 + 1 + 1 + 1 + 1 + 1 + 8 + 8 = 22 points)

You will implement the CRF inference layer in a tagger based on bi-directional LSTMs (BiLSTMs). As usual, download the starter kit provided and follow the instructions below.

As a warmup, run `python main.py /tmp/pos data/ptb-wsj-snippet --train --loss greedy --nochar`. This trains a BiLSTM greedy tagger without character-level information on a tiny POS tagging dataset in which train/validation/test portions are the same. (When the word labels do not have the BIO format, the code automatically sets the performance metric to be per-position accuracy.) Also run `python main.py /tmp/pos data/conll2003-snippet --train --loss greedy --nochar`. This trains the tagger on a tiny NER tagging dataset in which train/validation/test portions are the same. (When the word labels have the BIO format, the code automatically sets the performance metric to be global F\(_1\).)

\(^1\)Neural Architectures for Named Entity Recognition (Lample et al., 2016)
1. Take a look at tagged sequences in ptb-wsj-snippet/ and conll2003-snippet/ and the class TaggingDataset and explain in your own words how the program organizes labeled sequences into batches. In particular: (1) Why do we sort sequences by length? (2) When you set batch size $N$, does that mean every batch will contain $N$ sequences? (3) Is there any padding at the word sequence level? (4) How are characters organized? (5) Is there any padding at the character sequence level?

2. Take a look at evaluate method in BiLSTMTagger and explain how output[‘acc’] and output[‘f1_<all>’] are computed from ground-truth and predicted labeled sequences.

3. The greedy tagging loss is implemented for you in crf.py. Let $(x^{(1)}, y^{(1)}) \ldots (x^{(N)}, y^{(N)})$ denote $N$ labeled sentences in a batch where $x^{(i)} \in V^T$ and $y^{(i)} \in \{1 \ldots L\}^T$. For $i = 1 \ldots N$, let $h^{(i)} \in \mathbb{R}^{T \times L}$ denote the output of the BiLSTM layer given input $x^{(i)} \in V^T$ representing label scores at $T$ positions (i.e., scores[i]). Give a precise formula for the greedy loss computed on this batch as a function of $x^{(i)}, y^{(i)}, h^{(i)}$.

4. You can make the tagger a function of characters by simply removing --nochar in the command. Explain how character-level information is incorporated. In particular: (1) Given a word $w \in V$, how are its characters $c_1 \ldots c_{|w|} \in C$ ($C$ is the set of all character types) used to produce an embedding? (2) What is the final dimension of a word representation to be fed into the BiLSTM layer which produces the label scores $h \in \mathbb{R}^{T \times L}$ (as a function of $w$dim and $c$dim)?

5. Take a look at CRFLoss in crf.py. (1) What are the parameters of this layer? (2) Give a precise formula for the single scalar score$(h, y)$ computed by the layer in terms of the CRF parameters where $h \in \mathbb{R}^{T \times L}$ is label scores for an input sequence and $y \in \{1 \ldots L\}^T$ is a specific label sequence. This is implemented in score_targets. (3) What is the loss (computed in forward) as a function of the label sequence scores in (2)?

6. What quantities are compute_normalizers_brute and decode_brute computing and how? What is the computational complexity of these methods in terms of label set size $L$ and sequence length $T$?

7. **Forward algorithm.** Implement compute_normalizers with complexity $O(|L|^2T)$ to compute the same quantity computed by compute_normalizers_brute. The tensor $\text{prev} \in \mathbb{R}^{B \times L}$ contains the active part of the conceptual $B \times T \times L$ dynamic programming table, that is

$$\text{prev}[i][y] = \log \left( \sum_{y_1 \ldots y_{t-1}} \exp \left( \text{score}(x_1^{(i)} \ldots x_t^{(i)}, y_1 \ldots y_{t-1}y) \right) \right)$$

where $t = 1 \ldots T$ is implicitly traversed (since we only need the information from $t-1$ to get $t$). You will have to be careful with batch dimensions using commands like unsqueeze and expand. You will have to be careful with details like the directionality of self.T: its value at index $(y', y)$ means score for $y \rightarrow y'$ not $y' \rightarrow y$. Do not introduce any for loops except for the one iterating over positions $1 \ldots T$. Use logsumexp. **You have to pass the test test_compute_normalizer in test_crf.py to get any credits for this problem (run python test_crf.py).** (In the reference implementation, each line is a one-liner.)

8. **Viterbi algorithm.** Implement decode with complexity $O(|L|^2T)$ to compute the same quantity computed by decode_brute. The tensor $\text{prev} \in \mathbb{R}^{B \times L}$ contains the active part of the conceptual $B \times T \times L$ dynamic programming table, that is

$$\text{prev}[i][y] = \max_{y_1 \ldots y_{t-1}} \text{score}(x_1^{(i)} \ldots x_t^{(i)}, y_1 \ldots y_{t-1}y)$$

where $t = 1 \ldots T$ is implicitly traversed (since we only need the information from $t-1$ to get $t$). PyTorch’s max operation will yield indices, which you will store in the list back for backtracking purposes. After the main loop, you will backtrack from the final maximizers through back and store results into tape $\in \mathcal{Y}^{B \times T}$ (whose order will be reversed at return). Again be careful with dimensions/details and do not introduce any for loops. **You have to pass the test test_decode in test_crf.py to get any credits for this problem (run python test_crf.py).** (In the reference implementation, each line is a one-liner.)