3 Linear Structures (5 + 6 + 5 + 7 + 7 = 30 Points)

(a) The formula for estimating n-gram probabilities:

$$P\left(w_i|w_{i-n+1},\ldots,w_{i-1}\right) = \frac{Count\left(w_{i-n+1},\ldots,w_i\right)}{Count(w_{i-n+1},\ldots,w_{i-1})} \textit{ [2 points]}$$

$$P(never) = \frac{2}{35} \textit{ [1 points]},\ P(never|use) = \frac{0}{3} \textit{ [1 points]}$$

Practical motivation for start- and end-of-sentence symbols: to have probability estimations for words that are more likely to occur in sentence-initial or final position. *[1 points]*

(b) The probability of a word sequence is equal to the product of the probability of each word, given the previous $n-1$ words. This follows from the chain rule for calculating joint probabilities. *[1 points]*

$$P\left(w_1 \ldots w_n\right) \approx \prod_i P\left(w_i|w_{i-n+1}\ldots w_{i-1}\right) \textit{ [1 points]}$$

We cannot obtain reliable estimates for long n-grams because of *data sparseness*; given the creativity of human language, we are bound to face the problem of sparsity, i.e. words or n-grams that do not occur in our training corpus. *[1 points]*

Markov assumption simplifies our definition of "context" by introducing a partial independence assumption in which we assume that the probability of word at position i only depends on a fixed number of previous words k. In other words, the Markov assumption estimates a full context by only using the last $n-1$ words. For a bigram ($n=2$), that means just the previous word. For a higher-order (eg $n>2$) model, the probabilities could be conditioned on more words. *[2 points]*

Practical problems: PoS tagging, handwriting recognition, speech recognition. *[1 points]*

(c) The sentence (*I never use the passive*) is impossible because neither the pronoun I nor the bigram *I never* occur in our 'corpus'. *[0.5 points]*

The general idea behind smoothing is to reassign or move some of the probability mass in maximum likelihood estimation from frequent events to less frequent and unseen events. In other words, we assume that all events can in principle happen but those that we don't observe in the training corpus happen with very small probabilities.*[1.5 points]*

$$P_{\mathrm{L}}\left(w_n|w_{n-1}\right) = \frac{C\left(w_{n-1}w_n\right)+1}{C\left(w_{n-1}\right)+V} \textit{ [1 points]}$$

$$P_{\mathrm{L}}\left(I\ never\ use\ the\ passive\right) = \frac{C\left(I\ never\right)+1}{C\left(I\right)+V} \times \frac{C\left(never\ use\right)+1}{C\left(never\right)+V} \ldots$$

$|V| = 26$ (27 including punctuation)

$$P_{\mathrm{L}}\left(I\ never\ use\ the\ passive\right) = \frac{1}{26} \times \frac{3}{28} \times \frac{3}{29} \times \frac{2}{28} \textit{ [2 points]}$$

(d) Sample answer:

HMM vs. LM Hidden Markov Models assume a hidden layer of abstraction (states) that influences the word sequence (observations). The hidden state sequence is modelled as a Markov process, with the probability of each state conditional on the previous state(s). The emission probabilities model the probability of a word, conditional just on the current state. HMM arguably models the hidden layer as an n-gram language model (i.e. the transition probabilities are 'similar' to the n-gram probabilities). *[3 points]*

An HMM models the joint probability of the state and observation sequences:

$$P(S,O) = \prod_{i=1}^{n} P(s_i|s_{i-1})P(o_i|s_i) \text{ [1 points]}$$

HMM as LM: Using HMMs to compute the probability $P(S)$ of a string S is basically using HMMs as language models. Hence, we need to compute the probability of the observation sequence O given all the possible hidden sequences that give rise to this observation sequence, i.e. sum over all possible state sequences the joint probability of the observation sequence O and the state sequence S *[2 points]*.

Instead of using the Viterbi algorithm, however, we use the Forward algorithm which sums over the preceding column for each new column in the trellis (in contrast to Viterbi which maximizes over the preceding column) *[1 points]*.

(e) The Viterbi algorithm achieves dynamic programming algorithm by recording partial calculations for re-use during an iterative process. Each cell in the trellis records the highest probability that the sequence could be in the corresponding state, given the prefix of the input up to this time point. At each successive time point, we use the values calculated in the last iteration in the calculation to fill the next column of cells. *[3 points]*

Calculations:

- First column:
  1 $P(H|\langle S\rangle)P(o_1|H)$ *[1 points]*
  2 $P(C|\langle S\rangle)P(o_1|C)$ *[1 points]*
- Second column:
  1 $max(P(H|H)P(o_2|H), P(H|C)P(o_2|H))$ *[1 points]*
  2 $max(P(C|C)P(o_2|C), P(C|H)P(o_2|C))$ *[1 points]*

Providing the general recursive formula can also give 1 point in case the answer is otherwise incomplete:

$$v_i(x) = \max_{k=1}^{L} [v_{i-1}(k) \cdot P(x|k) \cdot P(o_i|x)]$$

4 Hierarchical Structures (5 + 5 + 4 + 9 + 7 = 30 Points)

(a) drawing four correct trees rooted in $\boxed{4}$ *[3 points]* and two rooted in $\boxed{2}$ *[1 points]*; total count: six *[1 points]*.

(b) *Local ambiguity* arises when there are multiple structurally distinct ways of deriving the same category for a given (sub-)string of input *[2 points]*.

The oval nodes (or equivalence classes of edges) in the parse forest correspond to local ambiguity *[1 points]*.

The CKY and generalized chart parsers 'pack' (or 'factor' or 'bound') local ambiguity, by keeping in the chart only one entry (the 'representative') for each category and each cell *[1 points]*.

Packing prevents cross-multiplication of local ambiguity and allows these parsers to construct the complete forest in polynomial time; naïvely computing all complete trees, on the other hand, would have exponential worst-case complexity *[1 points]*.