Few more hints on HW 1

- $N_0 = |V^n| - \text{number of n-grams}$
- For Good-Turing smoothing, when you get to last bucket how do you smooth?
  - You can assume absolute discounting, say $C^* = C-0.75$
  - You could do something fancy like try to predict $N_{\text{END}}$
  - Whatever you do, be sure to document it
Parts of speech are classes of words that behave similarly to each other. The term word class is sometimes used instead.

The study of parts of speech goes back very early in history:

- Sanskrit Grammarians such as Yaska and Panini studied word classes as early as the 6th century BCE.
- Dionysus of Thrax wrote about 8 word classes in around the 1st/2nd century CE.
- Modern European notions of parts of speech usually can be traced back to Dionysus of Thrax.

Parts of speech are determined linguistically by how they act, rather than by rules like:

- Nouns are people, places, or things.
Evidence for Parts of Speech

- Certain morphemes only attach to certain types of words
- Groups of words are inflected the same way
- The acceptability of replacing words in a sentence, e.g. in the sentence

The big ball bounced.

We can replace *big* with *red*, but not with *ate*

The red ball bounced.

*The ate ball ball bounced.*
Why do we Care about Parts of Speech for NLP?

- Knowing the part of speech can help us disambiguate when a word has multiple meanings
  - Example Here
  - Example Here

- Part of Speech is important feature to know for many downstream tasks including but not limited to:
  - Constituency Parsing
  - Dependency Parsing
  - Relationship Extraction
  - Semantic Lexicons
  - etc
Classes of Parts of Speech

- Part of Speech are traditionally divided into two classes, open class and closed class.
- Open class categories have new words added to them frequently and in English include:
  - Nouns
  - Verbs
  - Adjectives
  - Adverbs
- Closed class categories very rarely have new words added and tend to be function words. In English they include:
  - Pronouns
  - Prepositions
  - Determiners
  - Particles
The Universality of Parts of Speech

- A quick note about how universal parts of speech are
- It is generally accepted that all languages have nouns and verbs
- Beyond that, other parts of speech aren’t guaranteed
  - Most languages do have all the parts of speech we are familiar with
  - If a language doesn’t have a part of speech, that does not mean a concept can’t be expressed in that language, it is just done differently
    - Many languages without adjectives use verb like words to modify nouns
- Marine Carpuat (UMCP) gives this example from Riau Indonesian/ Malay (spoken just outside of Singapore) Ayam (chicken) Makan (eat) can mean:
  - The chicken is eating
  - The chicken ate
  - The chicken will eat
  - The chicken is being eaten
  - Where the chicken is eating
  - How the chicken is eating
  - Somebody is eating the chicken
  - The chicken that is eating
Parts of Speech Tagging, or PoS tagging, is the process of labeling of every word in a sentence with its correct part of speech.

- It is specific example of the general problem of sequence labeling that we will run into a few more times.

- It is generally done as a supervised task, because corpora annotated with parts of speech are relatively cheap to create and readily available.

- The simplest version would be to label each word with the most frequent part of speech it is in a training corpus, which would get an accuracy of about 90%.
A Markov Chain is a Finite State Machine where each arc is a transition probability between states.

○ All the arcs originating from a state must add up to 1

A Markov Chain is a probabilistic graphical model

In the example to the right, \( P(b|a) = 0.15 \)

How does this FSA demonstrate the Markov assumption?
Markov Chain Example

- Given the Markov Chain on the right, find the probabilities of
  - hot, warm, cold
  - cold, hot, cold, hot, hot
- Reminder
  - Combine a sequence of probabilities using the chain rule

Based on Example from SLP
Hidden Markov Models are Markov Models where the states aren’t directly observable.

- For example, we can observe words, but not their parts of speech.

For each state in the model, there is another set of probabilities, giving the probability of generating an observation, given the fact we are in that state.

- Assume the observation probability is only dependent on the state we’re in and not any priors.
- We’ll go into this in detail over the next couple of slides.

Things to do with an HMM:

- Calculate the likelihood of observations.
- Calculate the most likely sequence of states given an Observation.
- Train it (Learn the probabilities).
HMMs have been one of the areas students have traditionally struggled with in NLP courses
  ○ Or maybe instructors have struggled to teach it 😞

Next class we’ll discuss how to do Part of Speech tagging with HMMs, but first will go over an example specifically created by Jason Eisner (JHU) to teach them.

The scenario is: Imagine we are 100s of years in the future and we find Jason’s diary in which he recorded the number of ice creams he ate each day.
  ○ From this, we want to find out what the weather was like the summer Jason was keeping his ice cream diary.
Hidden Markov Model Example

- We make some guesses to get the probabilities, basing it off intuitions
  - A hot day is more likely to follow a hot day and vice versa
  - Jason probably ate more ice cream on hot days
- The following probabilities and HMM show the same thing

|      | p( ... | C ) | p( ... | H ) | p( ... | START ) |
|------|-------|-------|----------|
| p( 1 | ...  ) | 0.7   | 0.1     |           |
| p( 2 | ...  ) | 0.2   | 0.2     |           |
| p( 3 | ...  ) | 0.1   | 0.7     |           |
| p( C | ...  ) | 0.8   | 0.1     | 0.5       |
| p( H | ...  ) | 0.1   | 0.8     | 0.5       |
| P( END | ... ) | 0.1   | 0.1     | 0         |

Eisner, Jason 2002. An Interactive Spreadsheet for Teaching the Forward-Backward Algorithm
So Where is the Ice Cream?

- Each state has a probability table
  - This is sometimes represented as below

|       | p( ... | C )  | p( ... | H )  |
|-------|--------|--------|
| p( 1 | ... ) | 0.7    | 0.1    |
| p( 2 | ... ) | 0.2    | 0.2    |
| p( 3 | ... ) | 0.1    | 0.7    |

Eisner, Jason 2002. *An Interactive Spreadsheet for Teaching the Forward-Backward Algorithm*
Given an HMM, how do we determine the probability of a sequence of diary entries?

The naive way to do this would be to use the law of total probability

- The probability of a sequence is the sum of the probability of that sequence given all possible sequences of observations.
- This becomes intractable for complex HMMs, which are quite common.
  - For an HMM with $N$ states and sequence of length $L$, the number of sequences we’d have to look at is $N^L$.

The naive way would repeat a lot of intermediate calculations

- We can use dynamic programming to calculate the likelihood more efficiently.
We use a device called a trellis which stores previous calculations.

- A trellis is like a matrix of states, each observation is a column, each state a row.
Likelihood Example

- Define a value $\alpha$ for each state in the trellis that represents how likely we are to be in that particular state.

\[
\begin{align*}
\alpha_1(H) &= 0.5 \times 0.7 = 0.35 \\
0.05 &= 0.5 \times 0.1 = \alpha_1(C)
\end{align*}
\]
For a given time step $t$, and state $j$, 

$$\alpha_t(j) = \sum_{i}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t)$$

- $\alpha_1(H) = 0.35$
- $\alpha_1(C) = 0.05$
- $\alpha_2(H) = 0.057 = 0.35 \times 0.8 \times 0.2 + 0.05 \times 0.1 \times 0.2$
- $\alpha_2(C) = 0.015 = 0.35 \times 0.1 \times 0.2 + 0.05 \times 0.8 \times 0.2$
Likelihood Practice

- Calculate the alphas for the rest of the trellis

\[ \alpha_t(j) = \sum_{i}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t) \]
Likelihood Algorithm

- The algorithm we just stepped through can be written as three steps
  - Initialize
    - For each state, q, at time step 1, $\alpha_1(q)$ is the probability of getting to that state from the start times the probability of seeing observation 1 given state q.
    
    $\alpha_1(q) = a_0 q b_q(o_1), \text{ for all states, } 1 \leq q \leq N$

- Recursion
  - For each timestep t, for each state q in that timestep, $\alpha_t(q)$ is the sum of $\alpha_{t-1}(r)$ times the probability of getting to state q from state r times the probability of observation t given state q.

    $\alpha_t(q) = \sum_{i=0}^{N} \alpha_{t-1}(r) a_{rq} b_q(o_t), \text{ for all states and all times, } 1 \leq q \leq N, 1 \leq t \leq T$
The final step in calculating the likelihood is termination. Because we know we only go to the end state at the end of input, and that all states will end up there, the equation is sum of the probabilities of going to the end state from each state multiplied by $\alpha_T$.

$$P(O|\lambda) = \sum_{i}^{N} \alpha_T(q) a_{qF} \text{ for all states, } 1 \leq q \leq N$$
Decoding

- Likelihood is a useful calculation in many applications, but for part-of-speech tagging and others, what we are really interested in is the most likely sequence of states given a sequence of observations.
  - For the ice cream example, this answers our original question of determining the sequence of hot and cold days, given Jason’s diary entries.

- This task is known as decoding and we will use the **Viterbi Algorithm** to accomplish it.
  - The Viterbi algorithm is a dynamic programming solution very similar to the one used to calculate the likelihood.
  - Instead of summing, we take the maximum.

- We also need to keep track of the states we came from in order to return the best sequence of hidden states.
  - We use backpointers to do this.
To avoid confusion with likelihood, we define a value $v$ for each state in the trellis that represents the maximum probability of a particular state.

$v_1(H) = 0.5 \times 0.7 = 0.35$

$P(H|S) \times P(3|H) = 0.05 = 0.5 \times 0.1 = v_1(C)$

$P(C|S) \times P(3|C) = 0.05 = 0.5 \times 0.1 = v_1(C)$
For a given time step $t$, and state $q$, $v_t(q) = \max_r v_{t-1}(r) a_{rq} b_q(o_t)$.

$v_1(H) = 0.35$

$v_1(C) = 0.05$

$v_2(H) = 0.056 = \max(0.35 \times 0.8 \times 0.2, 0.05 \times 0.1 \times 0.2)$

$v_2(C) = 0.008 = \max(0.35 \times 0.1 \times 0.2, 0.05 \times 0.8 \times 0.2)$
Calculate the \( v \)'s and backtraces for the rest of the trellis

\[
v_t(q) = \max_r v_{t-1}(r) a_{rq} b_q(o_t)
\]

\( v_1(H) = 0.35 \)
\( v_1(C) = 0.05 \)
\( v_2(H) = 0.057 \)
\( v_2(C) = 0.015 \)
Viterbi Algorithm

- Like likelihood, Viterbi has 3 steps
  - Each step involves setting the backtrace in addition to the value of v

- Initialize
  - For each state, q, at time step 1, \( \alpha_1(q) \) is the probability of getting to that state from the start times the probability of seeing observation 1 given state q. (same as before)
    \[
v_1(q) = a_{0q} b_q(o_1), \text{ for all states, } 1 \leq q \leq N\]
  - All states at time 1 point back to the start state
    \[
b_{t1}(q) = 0, \text{ for all states } 1 \leq q \leq N\]
Viterbi Algorithm

● Recursion
  ○ For each timestep $t$, for each state $q$ in that timestep, $v_t(q)$ is the max of $v_{t-1}(r)$ times the probability of getting to state $q$ from state $r$ times the probability of observation $t$ given state $q$.

  \[
  v_t(q) = \max_r v_{t-1}(r) a_{rq} b_q(o_t),
  \]
  for all states and all times $1 \leq q \leq N, 1 \leq t \leq T$

  ○ The backtrace is the same, but we take the argmax instead of max

  \[
  b_{t}(q) = \arg \max_r v_{t-1}(r) a_{rq} b_q(o_t),
  \]
  for all states and all times $1 \leq q \leq N, 1 \leq t \leq T
Viterbi Algorithm

- **Termination**
  - To find the best score, we take the max of all states leading into the end state.
    \[
    \nu_T(q_F) = \max_r \nu_{T-1}(r)a_rF
    \]
  - The backtrace is the same, but we take the argmax instead of max
    \[
    bt_T(q_F) = \arg\max_r \nu_{T-1}(r)a_rF
    \]
  - The best sequence of states is gotten by following $bt_T$ and the backpointers from that state until we get to the start state.