HMM Posterior Probability
Friday’s discussion section will be a review session to help you prepare for next Wednesday’s midterm.

We will post practice exam problems and answers today.

We will also post the equation sheet that will be included with your exam (along with the bio glossary that will also be included with your exam).

The exam will be approximately half traditional “problems to solve” (i.e. like the homework questions), and half conceptual questions (i.e. like the in-class exercises).

The midterm does not cover any material from today onwards.

HW 5 will be due the Monday (Nov. 7) after the midterm.
example: last lecture you were asked about how to get “the same hidden state probability at all times $t$".

*the words don’t matter; the concept does:* a number of you worked out the right answer but thought it was “wrong” because you didn’t say “stationary distribution”. Your answers showed you were thinking for yourselves; that is even better than the “right answer”!

another common response: didn’t realize the problem was referring to the “stationary distribution”. Need to think about the meaning, rather than just looking for keywords.
Why am I asking you to answer questions in class that you haven’t already seen the answer to?

This is the only way I can get you to think for yourself.

If I work through an example of the question for you, showing you an “algorithm” for how to answer it “right”, then ask you the question with different numbers or details, then I’m just asking you to remember the right answer rather than think for yourself.

Why do you need to learn to think for yourself?

Because that’s what you need to do to solve problems in the real world. Plug-and-chug and regurgitation only work in the highly artificial world of exam questions that test Plug-and-chug and regurgitation.
Say we wish our model of the Occasionally Dishonest Casino to have the same hidden state probability \( p(X_t = F) \) at all times \( t \). Assuming the transition probabilities are \( \tau_{FL} = p(L|F) = 0.05 \) and \( \tau_{LF} = p(F|L) = 0.1 \), how can we achieve this goal? (If this is not possible with the given data, just say “Insufficient data”).
For Learning, Some Pain Is Good

There are three times you could discover you don’t really understand how to use the concepts:

- out in the real world, when you try to solve real problems... Hey UCLA, I want my money back!
- during the exam... Ouch!
- during class.

“No pain, no gain”? There’s some truth to this. You have to discover the specific ways in which you don’t understand what you’ve been told, in order to really learn it.
Study Suggestions

- Study together -- your grades are not in competition with each other.
- Talk about the things that are confusing you, until you get those concepts nailed down.
- Use the concept tests that confused you to show you where you need to clear up your understanding.
- Ask questions: CourseWeb class forum; Friday’s review session; office hours (Monday).
What About Longer Range Dependencies?

Say a hidden variable $\theta_t$ depends on more information than just $\theta_{t-1}$. Can this be handled in a Markov model?

Two ways of solving this:

- explicitly add more dependency edges, e.g. $p(\theta_t|\theta_{t-2}, \theta_{t-1})$ etc. Increases the computational complexity, but it also requires changes to the structure of your algorithm (think of this as additional “inner loops” in your algorithm).

- You can achieve the same thing without forcing changes to your algorithm, by creating “duplicate” states that represent the same state with different precursors. Then you can feed this state graph to a standard (first-order) HMM algorithm and it will work without requiring any code changes.
A gene consists of a protein coding sequence consisting of three nucleotide “words” (called codons) encoding the 20 amino acids. These trinucleotide frequencies cannot be encoded by a simple pairwise conditional probability \( p(X_{t+1} | X_t) \).
We can solve this by using a different set of states for codon position 1, codon position 2, and codon position 3.

Position 1 states can only transition to position 2 states, $2 \rightarrow 3$ and $3 \rightarrow 1$.

A further complication: genes of higher organisms are encoded by multiple segments ("exons") separated by "intron" segments that are removed (spliced) before translating to protein. An intron can occur anywhere in a coding sequence, e.g. between codon positions 1 and 2, 2 and 3, or 3 and 1. These are called a "phase 1 intron", phase 2 intron, or phase 0 intron.

In this case, a gene prediction HMM must track both splicing signals that indicate a possible exon-intron transition, and the coding sequence phase from the end of one exon to the start of the next.
since there are only three phases, an easy solution is just to duplicate the intron model for each separate phase.

the three intron models are identical (e.g. intron start site, end site and other features).

Having three of them simply tracks which phase our codon ended at, so that we start at the right codon position in the next exon.

This captures very long-range dependencies, e.g. an intron could be 100,000 bp long!

Note that even though this state graph structure will work in a standard (first-order) HMM algorithm, the computational complexity is *still* increased, because we are creating extra states.
Which of the following are valid statements about limitations of the Viterbi algorithm? Select all statement(s) that are true:

1. Since it is based on a recursion rule, it must be computed using a recursive algorithm.
2. It is restricted to systems that obey a first-order Markov property.
3. It may not tell you anything meaningful about the hidden states.
It’s possible to calculate the Viterbi path with a non-recursive algorithm.

The Viterbi algorithm can be applied to higher order Markov chains; the computational complexity goes up proportionally.

The Viterbi path may be largely meaningless, if there is substantial uncertainty about hidden states. The problem is that it does not tell you a posterior confidence for any of its hidden state inferences.

For example, what if many paths had almost as high probability as the Viterbi path?
Instead of reporting a single best path a la Viterbi, compute the posterior probability of each individual hidden state, i.e. \( p(\theta_t | \tilde{X}^n) \).

This turns out to be surprisingly straightforward to compute.

The basic (forward) calculation is just like Viterbi with “max” replaced by “sum” over the possible predecessor states \( \theta_{t-1} \).

Recall that Viterbi has an “implicit” backwards phase, i.e. it must “backtrack” from the maximum value of \( p(\tilde{\theta}^n \star) \).

Posterior calculation has an \textit{explicit} backward component. For computational efficiency we split \( p(\theta_t, \tilde{X}^n) \) into two halves: 
\( p(\theta_t, \tilde{X}^t) \) (forward) and \( p(\tilde{X}^n_{t+1} | \theta_t) \) (backward).

This is called the “forward-backward algorithm”.
To make this completely concrete, let’s just implement this now for a simple problem.

The Occasionally Dishonest Casino is a good example because it’s easy to interpret (when you see a lot of 6s, that’s probably the loaded dice), and only has two hidden states.

For simplicity, compute this in probability (not log-probability) space. This restricts us to fairly short obs sequences.

We’ll use Excel so we can see the calculations in action.