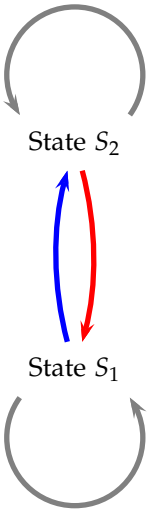
Basics of HMMs

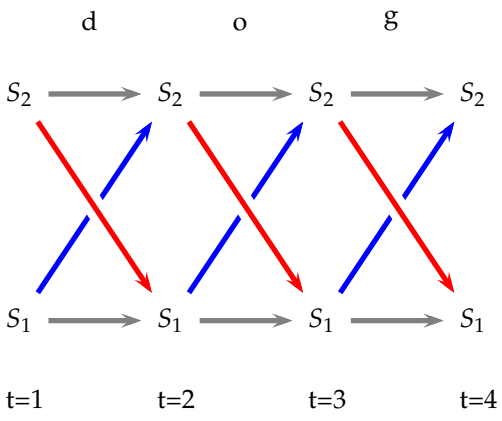
You should be able to take this and fill in the right-hand sides.

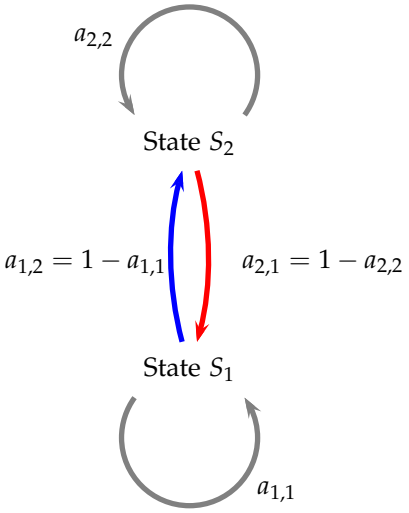
1 The problem

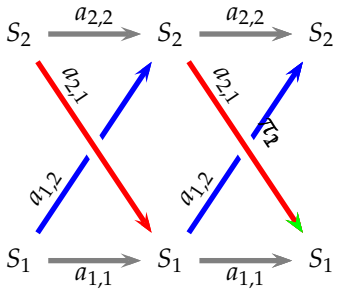
X = sequence of random variables (X_i).	There are N states: $S = S_1 \ldots S_N$. N=2 in these diagrams.
	The random variables taken on the states as their values.
O = $\{o_i\}_{i=1,T}$	Output sequence (letters, e.g.).
T	Number of symbols output—so we care about T+1 states.
Π	Initial probability distribution over the states.
A	Transition probabilities from state to state.
B	Emission probabilities: $b_{x_i o_i}$.

o_i is selected from our alphabet $\mathcal{A}$. For our project, the alphabet is letters, but you could build an HMM where the "alphabet" was words, i.e., the lexicon (vocabulary) of the language.









$$\text{Markov model on states: limited lookback (horizon) : } p(X_{t+1} = s_i | X_1 \dots X_t) = p(X_{t+1} = s_i | X_t) \quad (1)$$

$$\text{Stationary} p(X_{t+1} = s_i | X_t) = p(X_2 = s_j | X_1) \quad (2)$$

$$\text{Transition matrix:} a_{ij} = \quad (3)$$

So for fixed i,

$$\sum_{j=1}^{|S|} a_{ij} =$$

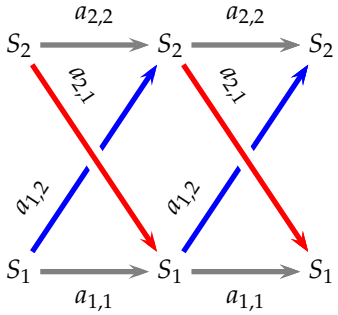
We initialize

$$p(X_1) = \pi_i.$$

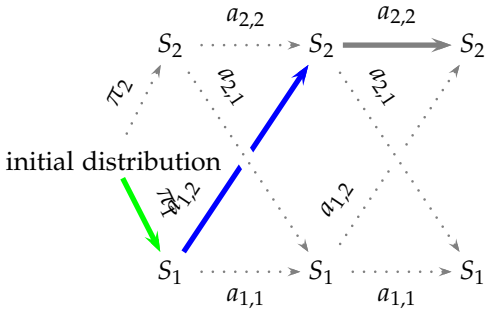
So (what are we summing over?)

$$\sum \pi_i =$$

start



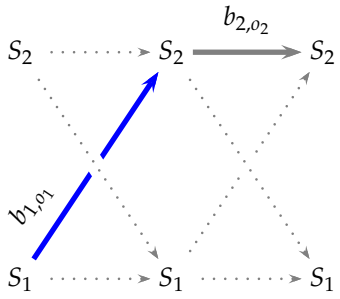
Now, **X** is a *path*, a sequence of states, such as $X_1X_2X_2$.



$$p(\mathbf{X}) = p(X_1 \dots X_T) = \pi_{x_1} \prod_{i=1}^{T} a_{x_i x_{i+1}} \tag{4}$$

The probability of taking a path **X** *and* generating a string O is equal to the product of the probability of the path times the probability of emitting the correct letter at each point on the path.

The probability of emitting the correct letter at each point on the path, *given the path*, is $\prod_{t=1}^{N} b_{x_t o_t}$



2 The Viterbi search for the best path

We often use μ to refer to the family of parameters.

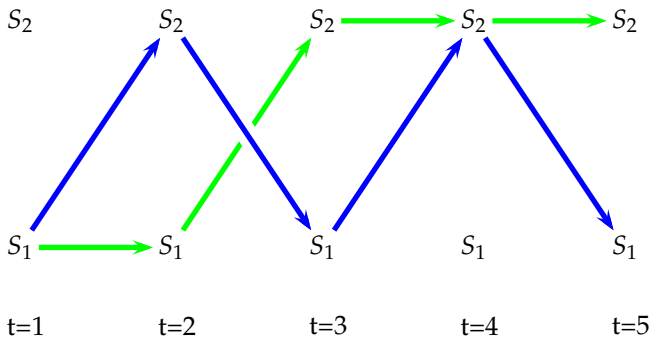
Find **X** to maximize p(**X** | O,μ) or p(**X**,O | μ). We are searching over all paths of length exactly *t-1*, not *t*.

Let's fix our ideas (as they say) by looking at a 2-state HMM, where the initial distribution π is uniform, and the states generate p,t,a,i with the following probabilities:

	1	2
p	.375	.125
t	.375	.125
a	.125	.375
i	.125	.375

From:	To:	1	2
1		.25	.75
2		.75	.25

Here are two paths, the blue and the gray, out of the 32 paths through this lattice:



What is the joint probability of each of those paths and the output *tipa*?

The blue path (disregarding the string emitted) has probability $0.5 \times 0.75 \times 0.75 \times 0.75 \times 0.75 = \frac{3^4}{2^9} = \frac{81}{512}$. Its probability of emitting the sequence *tipa* is $\frac{3^4}{8^4} = \frac{81}{4096}$, so the joint probability is $\frac{6561}{2097152} = .003\,128$.

And how about the green path? It has probability $0.5 \times 0.25^3 \times 0.75 = \frac{3}{512}$. Its probability of emitting the sequence *tipa* is $0.375 \times 0.125^3 = \frac{3}{8^4} = \frac{3}{4096} = 0.000\,732$, so the joint probability is $\frac{9}{2^{21}} = \frac{9}{2097152} = 0.000\,004\,291$.

But we *really* don't want to do all those calculations for each path.

$$\delta_j(t) = argmax_{X_1 \ldots X_{t-1}} P(X_1 \ldots X_{t-1}, o_1 \ldots o_{t-1}, X_t = j|\mu) \tag{5}$$

$$\begin{aligned}
\text{Initialize, for all states } i\text{: } \delta_i(1) &= \pi_i && (6)\\
\text{Induction: } \delta_i(t+1) &= max_j \delta_j(t) a_{ji}\, b_{jo_t} && (7)\\
\text{Store backtrace: } \psi_j(t+1) &= argmax_i \delta_i(t) a_{ij}\, b_{io_t} && (8)\\
\text{Termination: } \hat{X}_{T+1} &= argmax_i\, \delta_i(T+1) && (9)\\
\text{Back trace: } \hat{X}_t &= \psi_{\hat{X}_{t+1}}(t+1) && (10)\\
P(\hat{X}) &= max_i\, \delta_i(T+1) && (11)\\
& && (12)
\end{aligned}$$

3 Probability of a string, given current parameters

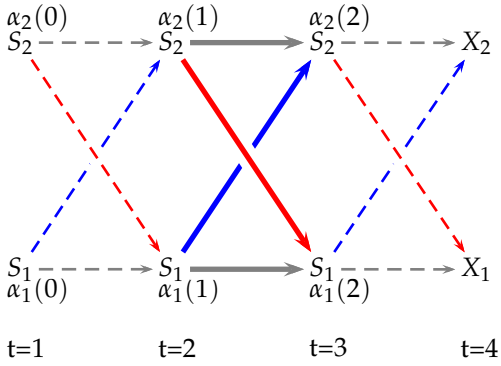
Given π, A, B, calculate $p(O|\mu)$. How? It is the sum over all of the paths, of the probability of emitting O times the probability of that path. We call this a sum of joint path-string probabilities, each of which is $p(X)\prod_{t=1}^{T} b_{x_t o_t}$. This sum, then, is:

$$\sum_{all\ paths\ X} p(X)\prod_{t=1}^{T} b_{x_t o_t} \tag{13}$$

$$\sum_{all\ paths\ X} \pi_{x_1}\prod_{t=1}^{N} a_{a_{x_t x_{t+1}}}\prod_{t=1}^{N} b_{x_t o_t} \tag{14}$$

To paraphrase this: the probability of the string O is equal to the sum of the joint path-string probabilities. And each path-string probability is the product of exactly T transitional probabilities, T emission probabilities, and one initial (π) probability. We will return to this.

$$\alpha_2(2) = \alpha_2(1) \times a_{2,2} \times b_{2,o_1} + \alpha_1(1) \times a_{1,2} \times b_{1,o_1}$$



$$\alpha_1(2) = \alpha_2(1) \times a_{2,1} \times b_{2,o_1} + \alpha_1(1) \times a_{1,1} \times b_{1,o_1}$$

Let's calculate the probability of being at state i at time t, after emitting t-1 letters. This amounts to summing over all the paths only the first part of the path-string probability. That sum of products is the *forward* quantity α. The part that is left over (the sum over all the path-strings to the right of t) will be summarized with the *backward* quantityt β.

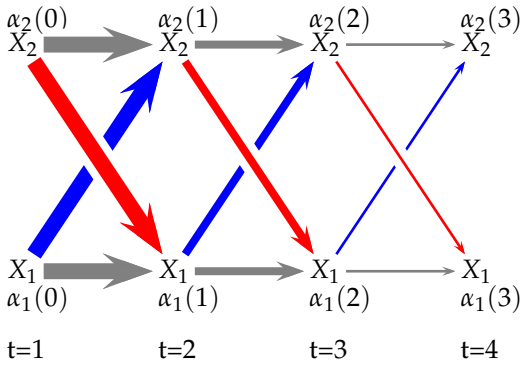
$$\text{Forward: } \alpha_i(t) = p(X_t = i, o_1 o_2 \dots o_{t-1}|\mu) \tag{15}$$

$$\textit{Calculate} \text{ : Initialize, for all states } i\text{: } \alpha_i(1) = \pi_i \tag{16}$$

$$\text{Induction: } \alpha_i(t+1) = \sum_j \alpha_j(t) a_{ji} b_{jo_t} \tag{17}$$

$$\text{End (total): } P(O|\mu) = \sum \alpha_i(T+1) \tag{18}$$

$$\tag{19}$$



Similarly, we calculate the probability of generating the rest of the observed letters, given that we are at state i at time t.

$$\text{Backward: } \beta_i(t) = p(o_t \dots o_T | X_t = i, \mu) \tag{20}$$

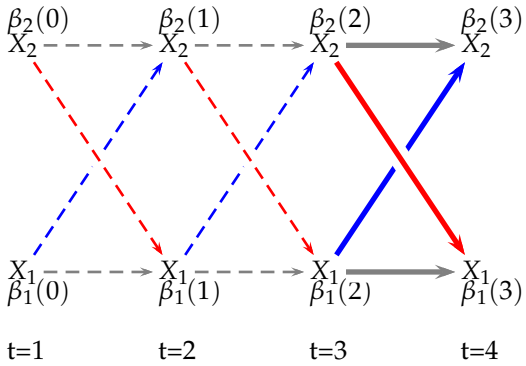
$$\textit{Calculate} \text{ : Initialize, for all states } i\text{: } \beta_i(T+1) = 1 \tag{21}$$

$$\text{Induction: } \beta_i(t) = \sum_j \beta_j(t+1)\, a_{ij}\, b_{io_t} \tag{22}$$

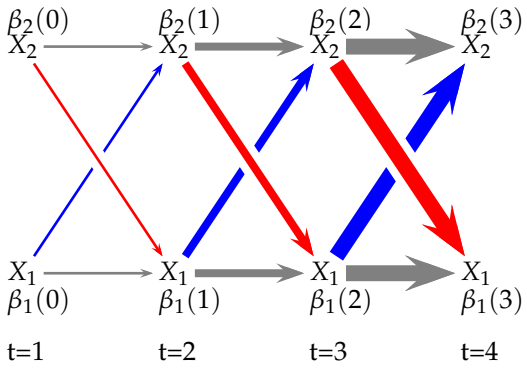
$$\text{End (total): } P(O|\mu) = \sum_i \pi_i \beta_i(1) \tag{23}$$

$$\tag{24}$$

$$\beta_2(2) = \beta_1(3) \times a_{2,2} \times b_{2,o_2} + \beta_2(3) \times a_{2,1} \times b_{2,o_2}$$



$$\beta_1(2) = \beta_1(3) \times a_{1,1} \times b_{1,o_1} + \beta_2(3) \times a_{1,2} \times b_{1,o_1}$$



Mixing α and β:

$$P(O|\mu) = \sum_i \alpha_i(t)\beta_i(t) \tag{25}$$

I think that you understand the whole idea if and only if you see that this equation is true. The basic insight is that the total probability of the string is equal to sum, over all paths through the lattice, of the product of the probability of the path times the probability of generating the string along that path. And that for any time t, we can partition all of those paths by looking at which set of paths goes through each of the states. If that is clear, then you have it.

The probability that the HMM generates our string is equal to the sum of the joint path-string probability.

Finding the best path through the HMM for a given set of data is the main reason we created the HMM. The state that the best path is in when it emits a given symbol is a *label* that the model assigns to that piece of data. In the case we look at, it is Consonant versus Vowel.

The values of a are also very important for us. For a two-state model, there are two independent parameters, which we choose to be a_{11} and a_{22}. If we let the system learn from the data, and the data is linguistic letters, then both of those values should be low, because the system should learn a consonant/vowel distinction, and there is a strong tendency to alternative between C's and V's.

```
def Forward(States,Pi,thisword):
    Alpha= dict()
    for s in range(len(States)):
        Alpha[(s,1)] = Pi[s]
    for t in range(2,len(thisword)):
        for to_state in States:
            Alpha[(to_state,t)] = 0
            for from_state in States:
                Alpha[(to_state,t)] += Alpha[(from_state,t-1)] *
                from_state.m_EmissionProbs[thisword[t]] * from_state.m_TransitionProbs[to_state]
    return Alpha

def Backward( States, thisword):
    Beta = dict()
    last = len(thisword) + 1
    for s in range(len(States)):
        Beta[(s, last)] = 1
    for t in range(len(thisword),1,-1):
    for from_state in States:
    Beta[(from_state,t)] = 0
            for to_state in States:
                Beta[(from_state,t)] += Beta[(to_state,t+1)] * from_state.m_EmissionProbs[thisword[t]] * from_state.m_Tra
    return Beta
```

4 Counting expected (soft) counts

The probability of going from $i \to j$ from time t to time $t+1$ during the generation of string O. Conceptually, this means looking at *all* the path-string pairs, and dividing them into N^2 different sets (based on which state they are in at time t and time $t+1$).

We know that the total probability of the string is the sum of a list of products, each with 2N+1 factors. For each pair i, j, we consider $\alpha_i(t)a_{ij}b_{io_t}\beta_j(t+1)$. What is the meaning of

$$\frac{\alpha_i(t)a_{ij}b_{io_t}\beta_j(t+1)}{p(O)}?$$

It is quite simply the *proportion of the total probability* (of the path-string pair) that goes through state i at t and state j at $t+1$. And that is exactly what we mean by the *expected* count of the transitions between those two states at that time interval. And by construction, those N^2 soft counts add up to 1.0.

$$\begin{aligned} p_t(i,j) &= p(X_t = i, X_{t+1} = j|O,\mu) && (26)\\ &= \frac{p(X_t = i, X_{t+1} = j, O|\mu)}{p(O)} && (27)\\ &= \frac{\alpha_i(t)a_{ij}b_{io_t}\beta_j(t+1)}{p(O)} && (28)\\ & && (29) \end{aligned}$$

If we *fix t*, then summing $p_t(i,j)$ over all i,j must sum to 1.0. And the distribution over all pairs (i,j) is exactly what we call the *soft counts* of generating o_t associated with each of the state-transitions.

Now we perform that computation for each letter in our entire corpus, and we keep track of a table of the following form. We will call this the SoftCount function: SC(l,i,j).[1] Here, SC(a,1,1) = 0.37.

Soft Count table

letter	from state	to state	total soft count
a	1	1	0.37
a	1	2	0.12
a	2	1	0.45
a	2	2	0.08

etc.

And we will set up a very similar table in which we include only those transitions that occur at time t=1 (summing over all of our data). Call this ISC(l,i,j) (for "initial soft count").

These are *expected counts*. Now we perform the *maximization* operations, by which we set the values of our three sets of parameters on the next iteration to the frequencies of the expected counts on the current iteration:

$$\pi_i = \sum_{l \in \mathcal{A}, 1 \leq j \leq N} \frac{\sum ISC(l,i,j)}{Z}$$

where Z is the number of words in our training corpus and $\mathcal{A}$ is our alphabet. Do you see why we divide by Z? The soft counts in this initial position add up to what, anyway?

We now can calculate

$$\frac{\text{expected number of transitions from state i to state j}}{\text{expected number of transitions from state } i}$$

$$a_{ij} = \frac{\sum_{l \in \mathcal{A}} SC(l,i,j)}{\sum_{l \in \mathcal{A}, 1 \leq k \leq N} SC(l,i,k)}$$

So this will be our new value for a_{ij} on the next learning iteration.

In much the same way we define the new values of b:

$$b_{il} = \frac{\sum_{1 \leq j \leq N} SC(l,i,j)}{\sum_{m \in \mathcal{A}, 1 \leq j \leq N} SC(m,i,j)}$$

[1] This is where my presentation departs from that of Manning and Schuetze; I find the way I describe here clearer.