CMSC 723: Computational Linguistics I — Session #5

Hidden Markov Models

Jimmy Lin
The iSchool
University of Maryland

Wednesday, September 30, 2009

Today's Agenda

- The great leap forward in NLP
- Hidden Markov models (HMMs)
  - Forward algorithm
  - Viterbi decoding
  - Supervised training
  - Unsupervised training teaser
- HMMs for POS tagging

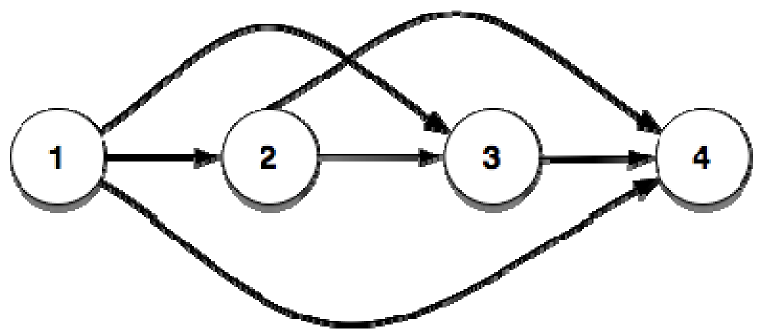
Deterministic to Stochastic

- The single biggest leap forward in NLP:
  - From deterministic to stochastic models
  - What? A *stochastic process* is one whose behavior is non-deterministic in that a system's subsequent state is determined both by the process's predictable actions and by a random element.
- What's the biggest challenge of NLP?
- Why are deterministic models poorly adapted?
- What's the underlying mathematical tool?
- Why can't you do this by hand?

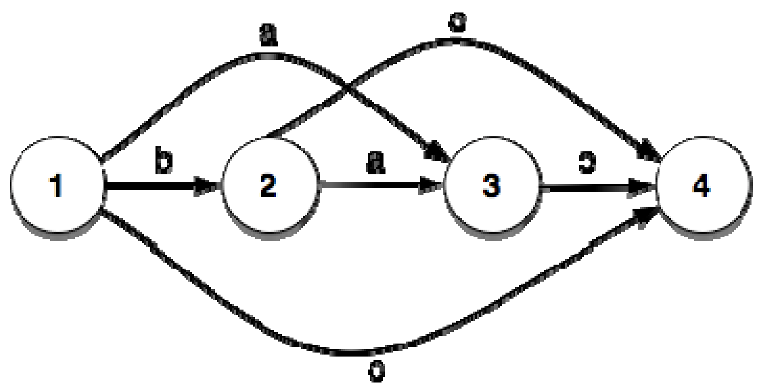
FSM: Formal Specification

- Q: a finite set of N states
  - $Q = \{q_0, q_1, q_2, q_3, \ldots\}$
  - The start state: q_0
  - The set of final states: q_F
- Σ: a finite input alphabet of symbols
- $\delta(q,i)$: transition function
  - Given state q and input symbol i, transition to new state q'

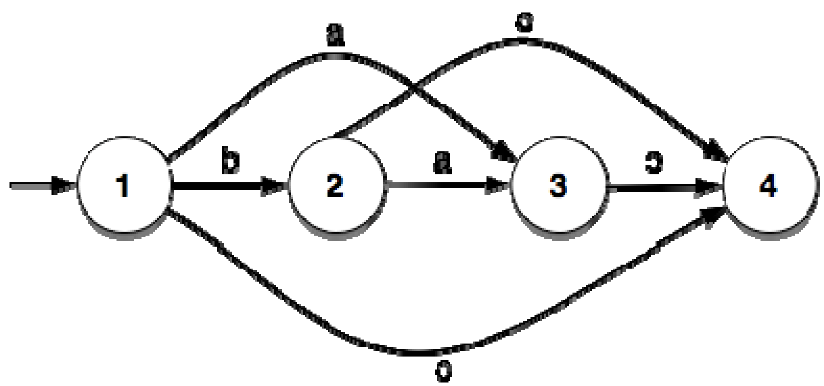
Finite number of states



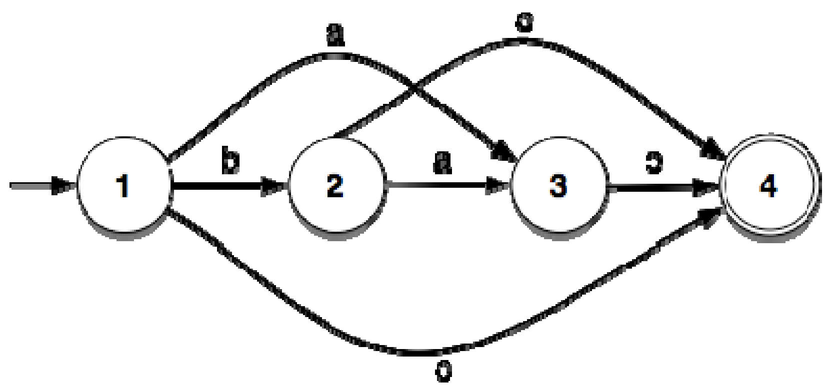
Transitions



Input alphabet



Start state



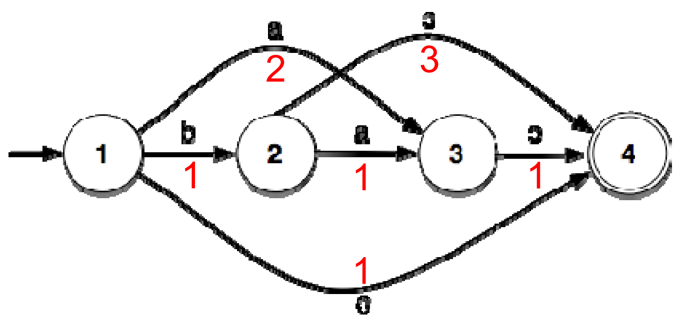
Final state(s)

The problem with FSMs…

- All state transitions are equally likely
- But what if we *know* that isn't true?
- How might we *know*?

Weighted FSMs

- What if we know more about state transitions?
  - ‘a’ is twice as likely to be seen in state 1 as ‘b’ or ‘c’
  - ‘c’ is three times as likely to be seen in state 2 as ‘a’



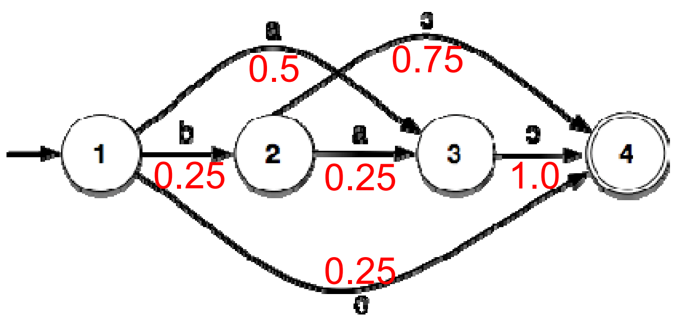
- FSM → Weighted FSM
- What do we get of it?
  - score(‘ab’) = 2 (?)
  - score(‘bc’) = 3 (?)

Introducing Probabilities

- What's the problem with adding weights to transitions?
- What if we replace weights with probabilities?
  - Probabilities provide a theoretically-sound way to model uncertainly (ambiguity in language)
  - But how do we assign probabilities?

Probabilistic FSMs

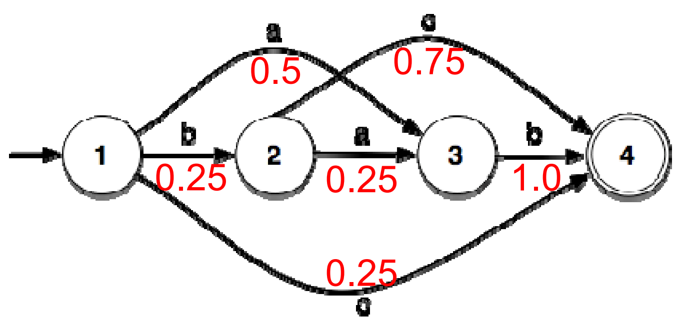
- What if we know more about state transitions?
  - 'a' is twice as likely to be seen in state 1 as 'b' or 'c'
  - 'c' is three times as likely to be seen in state 2 as 'a'



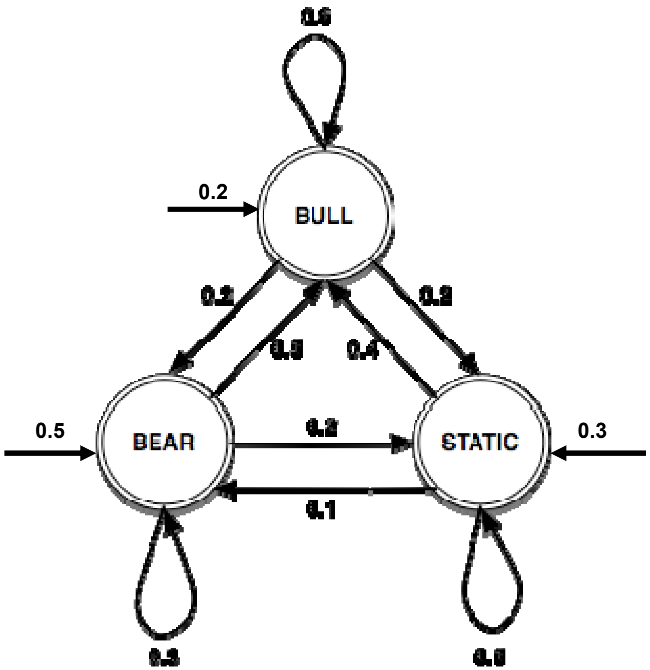
- What do we get of it? What's the interpretation?
  - P('ab') = 0.5
  - P('bc') = 0.1875
- This is a Markov chain

Markov Chain: Formal Specification

- Q: a finite set of N states
  - $Q = \{q_0, q_1, q_2, q_3, \ldots\}$
- The start state
  - An explicit start state: q_0
  - Alternatively, a probability distribution over start states: $\{\pi_1, \pi_2, \pi_3, \ldots\}$, $\Sigma\, \pi_i = 1$
- The set of final states: q_F
- $N \times N$ Transition probability matrix $A = [a_{ij}]$
  - $a_{ij} = P(q_j|q_i)$, $\Sigma\, a_{ij} = 1 \quad \forall i$



Let's model the stock market…



Each state corresponds to a physical state in the world

What's missing? Add "priors"

- What's special about this FSM?
  - Present state only depends on the previous state!
- The (1st order) Markov assumption
  - $P(q_i|q_0 \ldots q_{i-1}) = P(q_i|q_{i-1})$

Are states always observable ?

Day: 1 2 3 4 5 6

Not observable !

B_{ull} B_{ear} S B_{ear} S B_{ull}

Bull: Bull Market
Bear: Bear Market
S: Static Market

Here's what you actually observe:

↑ ↓ ↔ ↑ ↓ ↔

↑: Market is up
↓: Market is down
↔: Market hasn't changed

Hidden Markov Models

- Markov chains aren’t enough!
  - What if you can’t directly observe the states?
  - We need to model problems where observations don’t directly correspond to states…
- Solution: A Hidden Markov Model (HMM)
  - Assume two probabilistic processes
  - Underlying process (state transition) is hidden
  - Second process generates sequence of observed events

HMM: Formal Specification

- Q: a finite set of N states
  - $Q = \{q_0, q_1, q_2, q_3, \ldots\}$
- $N \times N$ Transition probability matrix A = $[a_{ij}]$
  - $a_{ij} = P(q_j|q_i)$, $\Sigma\, a_{ij} = 1 \quad \forall i$
- Sequence of observations $O = o_1, o_2, \ldots\, o_T$
  - Each drawn from a given set of symbols (vocabulary V)
- $N \times |V|$ Emission probability matrix, $B = [b_{it}]$
  - $b_{it} = b_i(o_t) = P(o_t|q_i)$, $\Sigma\, b_{it} = 1 \quad \forall i$
- Start and end states
  - An explicit start state q_0 or alternatively,
    a prior distribution over start states: $\{\pi_1, \pi_2, \pi_3, \ldots\}$, $\Sigma\, \pi_i = 1$
  - The set of final states: q_F

Stock Market HMM

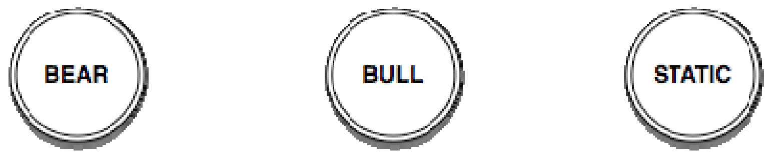
States? ✓

Transitions?

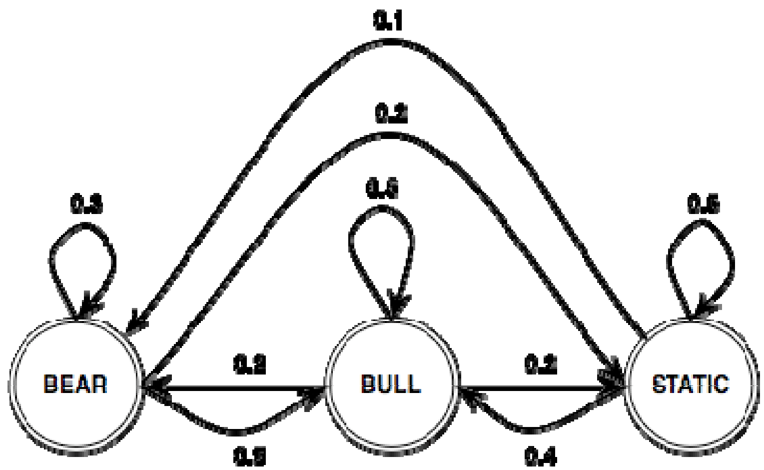
Vocabulary?

Emissions?

Priors?



Stock Market HMM



States? ✓

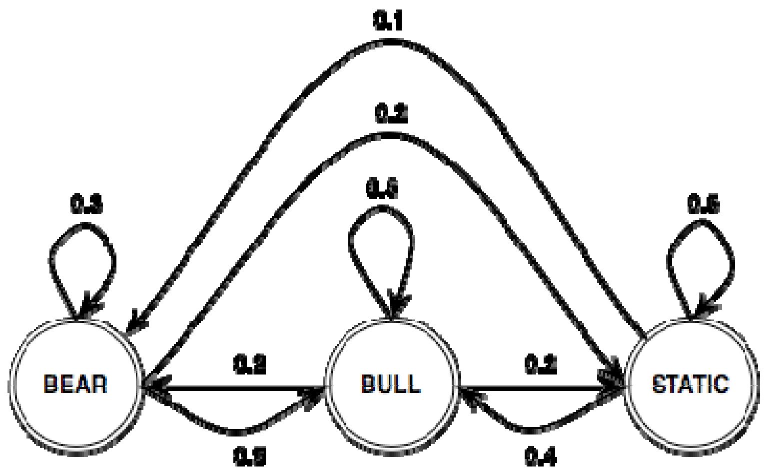
Transitions? ✓

Vocabulary?

Emissions?

Priors?

Stock Market HMM



States? ✓

Transitions? ✓

Vocabulary? ✓

Emissions?

Priors?

$$V = \{\uparrow, \downarrow, \leftrightarrow\}$$

Stock Market HMM

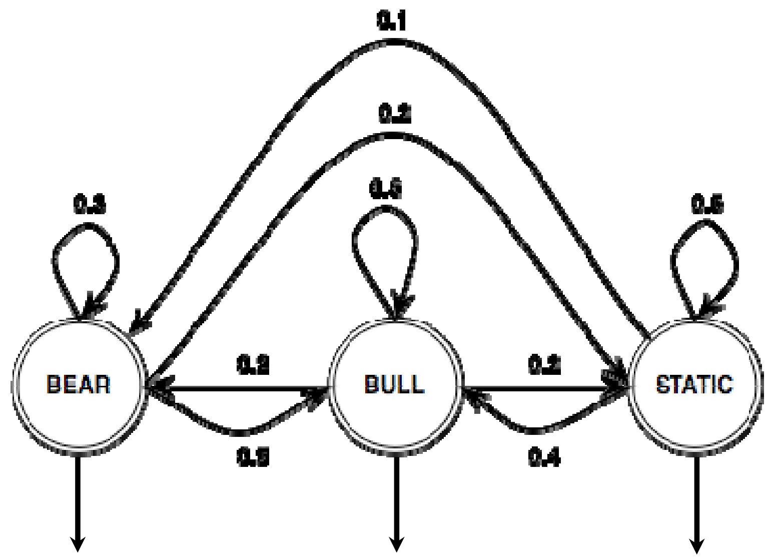
States? ✓

Transitions? ✓

Vocabulary? ✓

Emissions? ✓

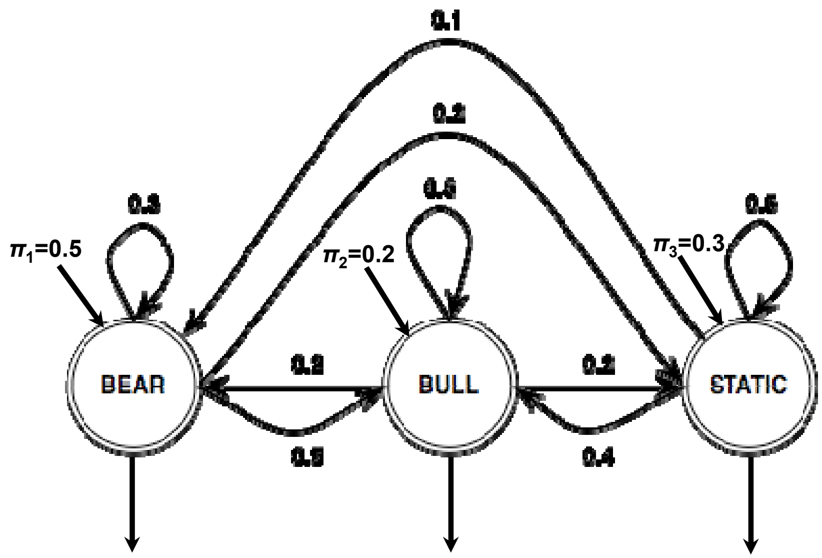
Priors?



$$\begin{bmatrix} P(\uparrow|Bear) = 0.1 \\ P(\downarrow|Bear) = 0.6 \\ P(\leftrightarrow|Bear) = 0.3 \end{bmatrix} \quad \begin{bmatrix} P(\uparrow|Bull) = 0.7 \\ P(\downarrow|Bull) = 0.1 \\ P(\leftrightarrow|Bull) = 0.2 \end{bmatrix} \quad \begin{bmatrix} P(\uparrow|Static) = 0.3 \\ P(\downarrow|Static) = 0.3 \\ P(\leftrightarrow|Static) = 0.4 \end{bmatrix}$$

$$V = \{\uparrow, \downarrow, \leftrightarrow\}$$

Stock Market HMM



$$\begin{bmatrix} P(\uparrow|Bear) = 0.1 \\ P(\downarrow|Bear) = 0.6 \\ P(\leftrightarrow|Bear) = 0.3 \end{bmatrix} \quad \begin{bmatrix} P(\uparrow|Bull) = 0.7 \\ P(\downarrow|Bull) = 0.1 \\ P(\leftrightarrow|Bull) = 0.2 \end{bmatrix} \quad \begin{bmatrix} P(\uparrow|Static) = 0.3 \\ P(\downarrow|Static) = 0.3 \\ P(\leftrightarrow|Static) = 0.4 \end{bmatrix}$$

$$V = \{\uparrow, \downarrow, \leftrightarrow\}$$

States? ✓

Transitions? ✓

Vocabulary? ✓

Emissions? ✓

Priors? ✓

Properties of HMMs

- The (first-order) Markov assumption holds
- The probability of an output symbol depends only on the state generating it

$$P(o_t | q_1, q_2, \ldots, q_N, o_1, o_2, \ldots, o_T) = P(o_t | q_i)$$

- The number of states (N) does not have to equal the number of observations (T)

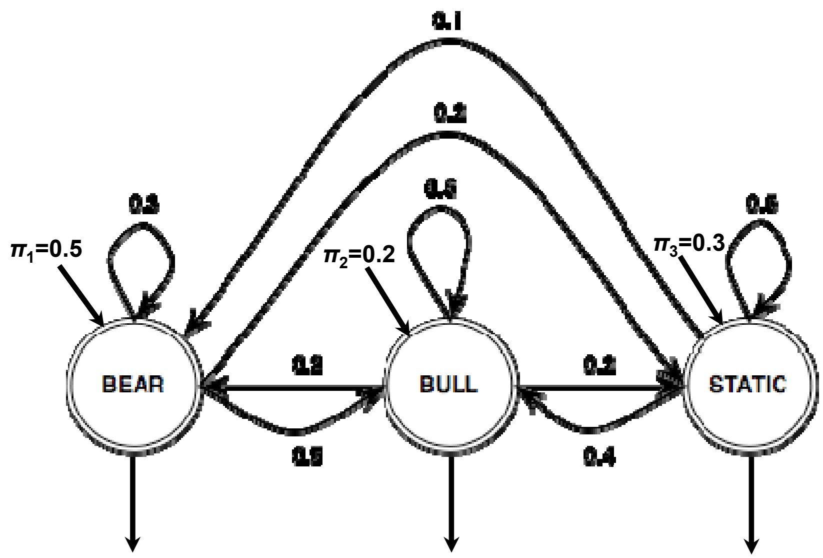
HMMs: Three Problems

- **Likelihood:** Given an HMM $\lambda = (A, B, \prod)$, and a sequence of observed events O, find $P(O|\lambda)$
- **Decoding:** Given an HMM $\lambda = (A, B, \prod)$, and an observation sequence O, find the most likely (hidden) state sequence
- **Learning:** Given a set of observation sequences and the set of states Q in λ, compute the parameters A and B

Okay, but where did the structure of the HMM come from?

HMM Problem #1: Likelihood

Computing Likelihood



$$\begin{bmatrix} P(\uparrow|Bear)=0.1 \\ P(\downarrow|Bear)=0.6 \\ P(\leftrightarrow|Bear)=0.3 \end{bmatrix} \quad \begin{bmatrix} P(\uparrow|Bull)=0.7 \\ P(\downarrow|Bull)=0.1 \\ P(\leftrightarrow|Bull)=0.2 \end{bmatrix} \quad \begin{bmatrix} P(\uparrow|Static)=0.3 \\ P(\downarrow|Static)=0.3 \\ P(\leftrightarrow|Static)=0.4 \end{bmatrix}$$

λ_{stock}

t: 1 2 3 4 5 6

O: $\uparrow$ $\downarrow$ $\leftrightarrow$ $\uparrow$ $\downarrow$ $\leftrightarrow$

Assuming λ_{stock} models the stock market, how likely are we to observe the sequence of outputs?

Computing Likelihood

- Easy, right?
  - Sum over all possible ways in which we could generate *O* from *λ*

$$P(O|\lambda) = \sum_{Q} P(O,Q|\lambda) = \sum_{Q} P(O|Q,\lambda)P(Q|\lambda)$$

$$= \boxed{\sum_{q_1,q_2\ldots q_T} \pi_{q_1} b_{q_1}(o_1) a_{q_1 q_2} \ldots a_{q_{T-1} q_T} b_{q_T}(o_T)}$$

  - What's the problem? Takes O(N^T) time to compute!
- Right idea, wrong algorithm!

Computing Likelihood

- What are we doing wrong?
  - State sequences may have a lot of overlap…
  - We're recomputing the shared subsequences every time
  - Let's store intermediate results and reuse them!
  - Can we do this?
- Sounds like a job for dynamic programming!

Forward Algorithm

- Use an $N \times T$ trellis or chart $[\alpha_{tj}]$
- Forward probabilities: α_{tj} or $\alpha_t(j)$
  - = P(being in state j after seeing t observations)
  - = $P(o_1, o_2, \ldots o_t, q_t=j)$
- Each cell = ∑ extensions of all paths from other cells
  $\alpha_t(j) = \sum_i \alpha_{t-1}(i)\, a_{ij}\, b_j(o_t)$
  - $\alpha_{t-1}(i)$: forward path probability until $(t-1)$
  - a_{ij}: transition probability of going from state i to j
  - $b_j(o_t)$: probability of emitting symbol o_t in state j
- $P(O|\lambda) = \sum_i \alpha_T(i)$
- What's the running time of this algorithm?

Forward Algorithm: Formal Definition

- Initialization

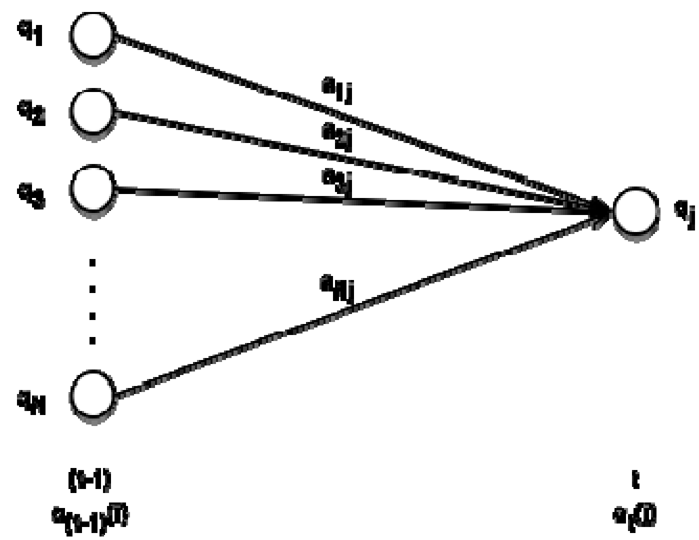
$$\alpha_1(j) = \pi_j b_j(o_1), 1 \leq j \leq N$$

- Recursion

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t); 1 \leq j \leq N, 2 \leq t \leq T$$

- Termination

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

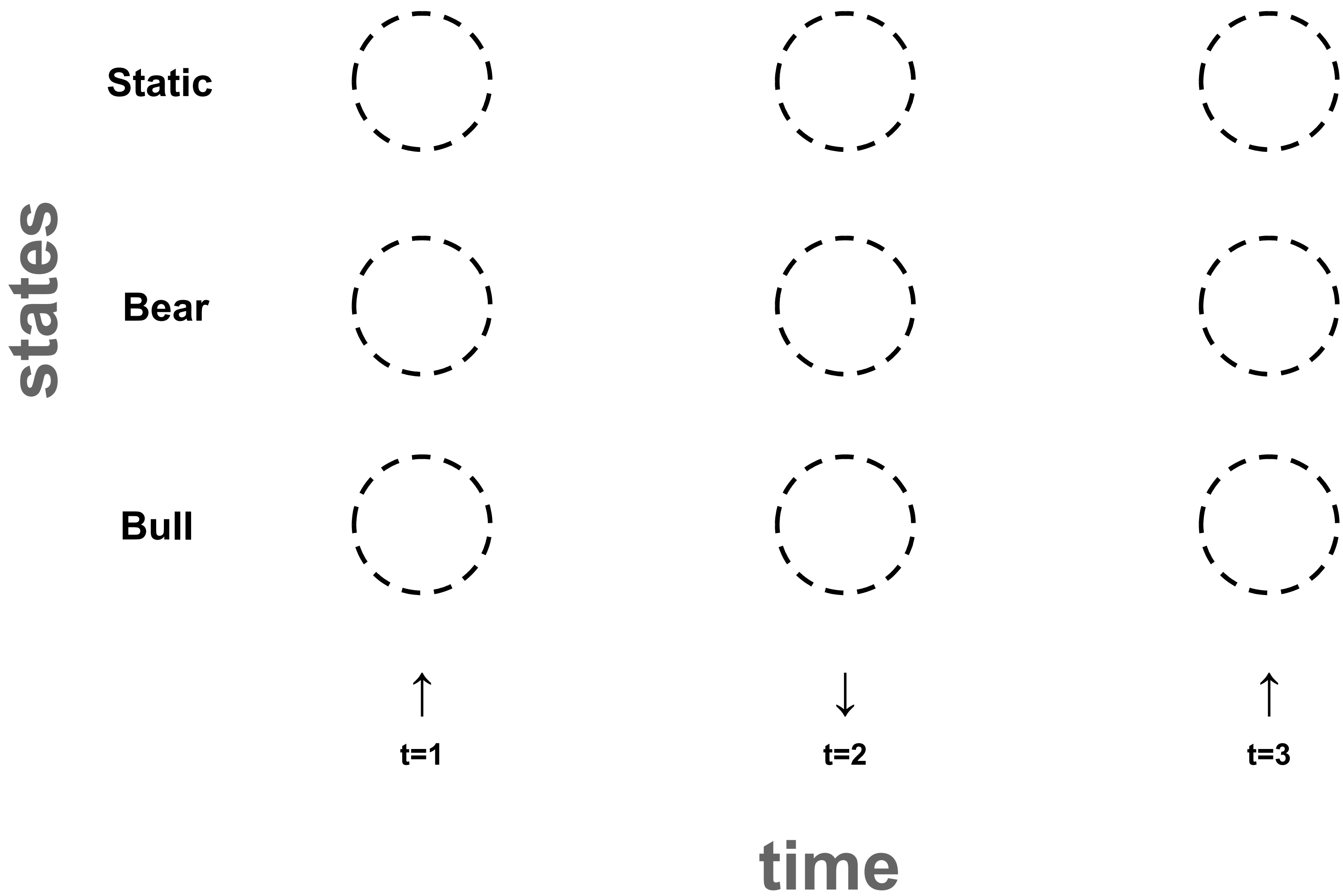


Forward Algorithm

$$O = \uparrow\downarrow\uparrow$$

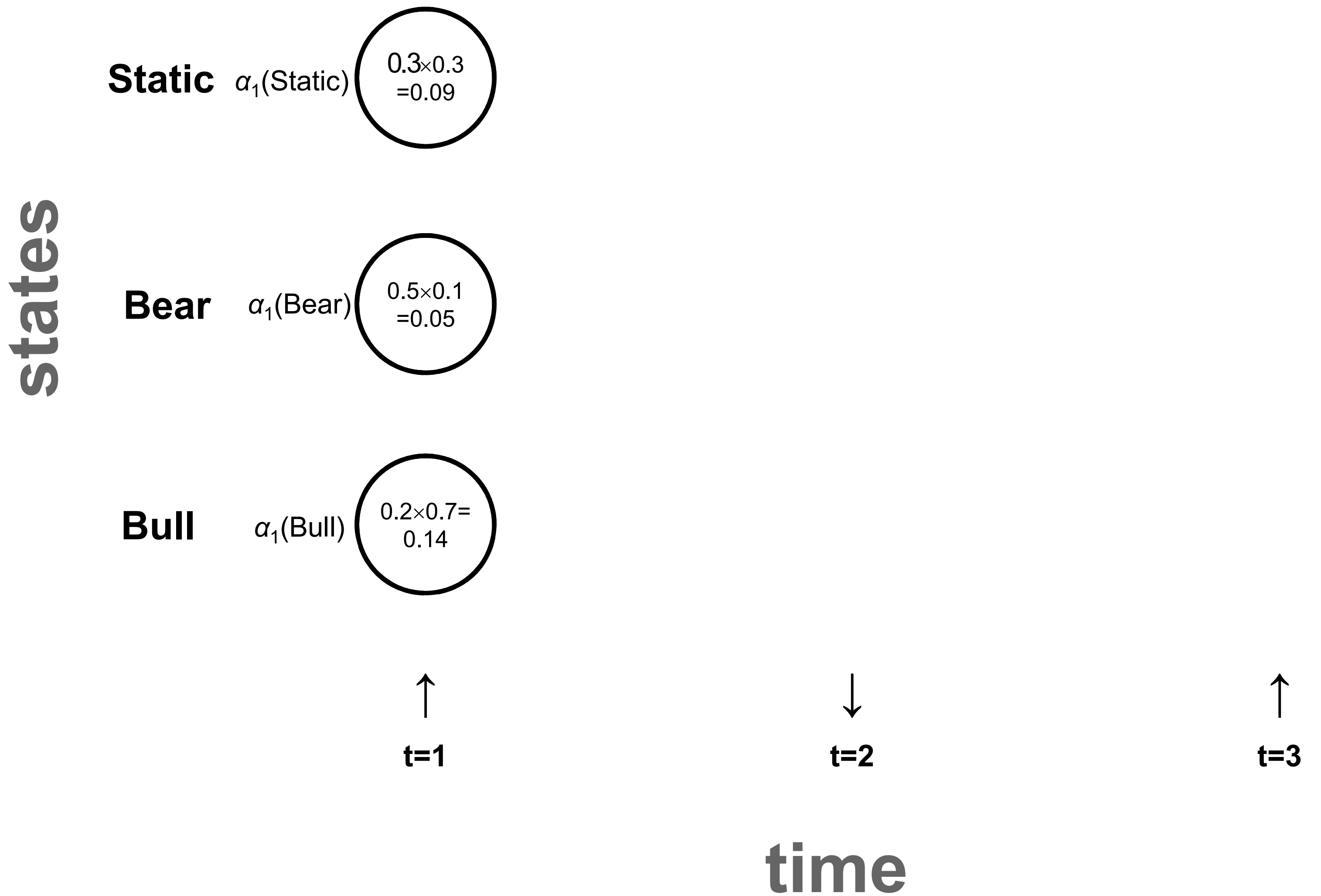
$$\text{find } P(O|\lambda_{stock})$$

Forward Algorithm



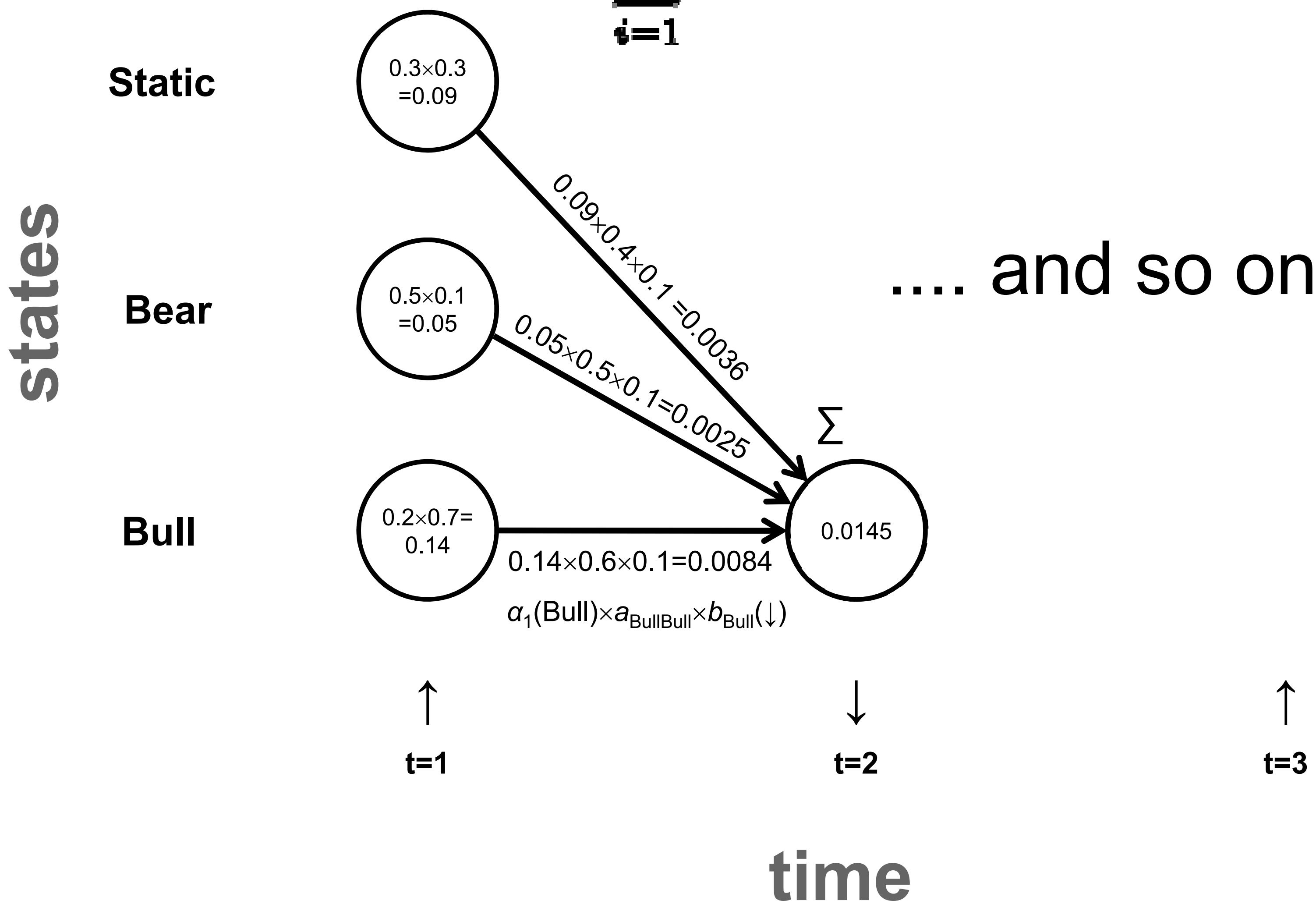
Forward Algorithm: Initialization

$$\alpha_1(j) = \pi_j b_j(o_1), 1 \leq j \leq N$$



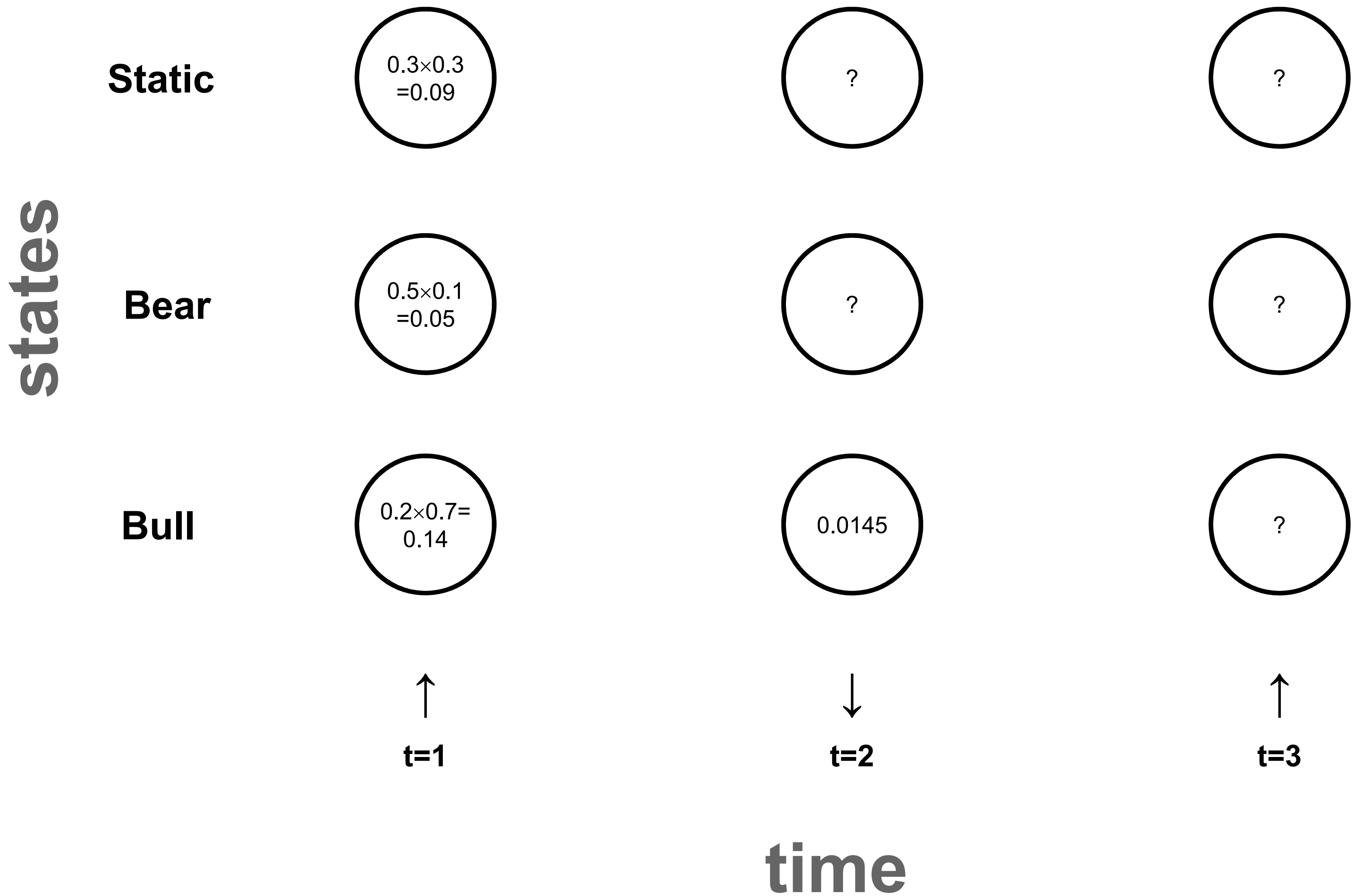
Forward Algorithm: Recursion

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t); 1 \leq j \leq N, 2 \leq t \leq T$$



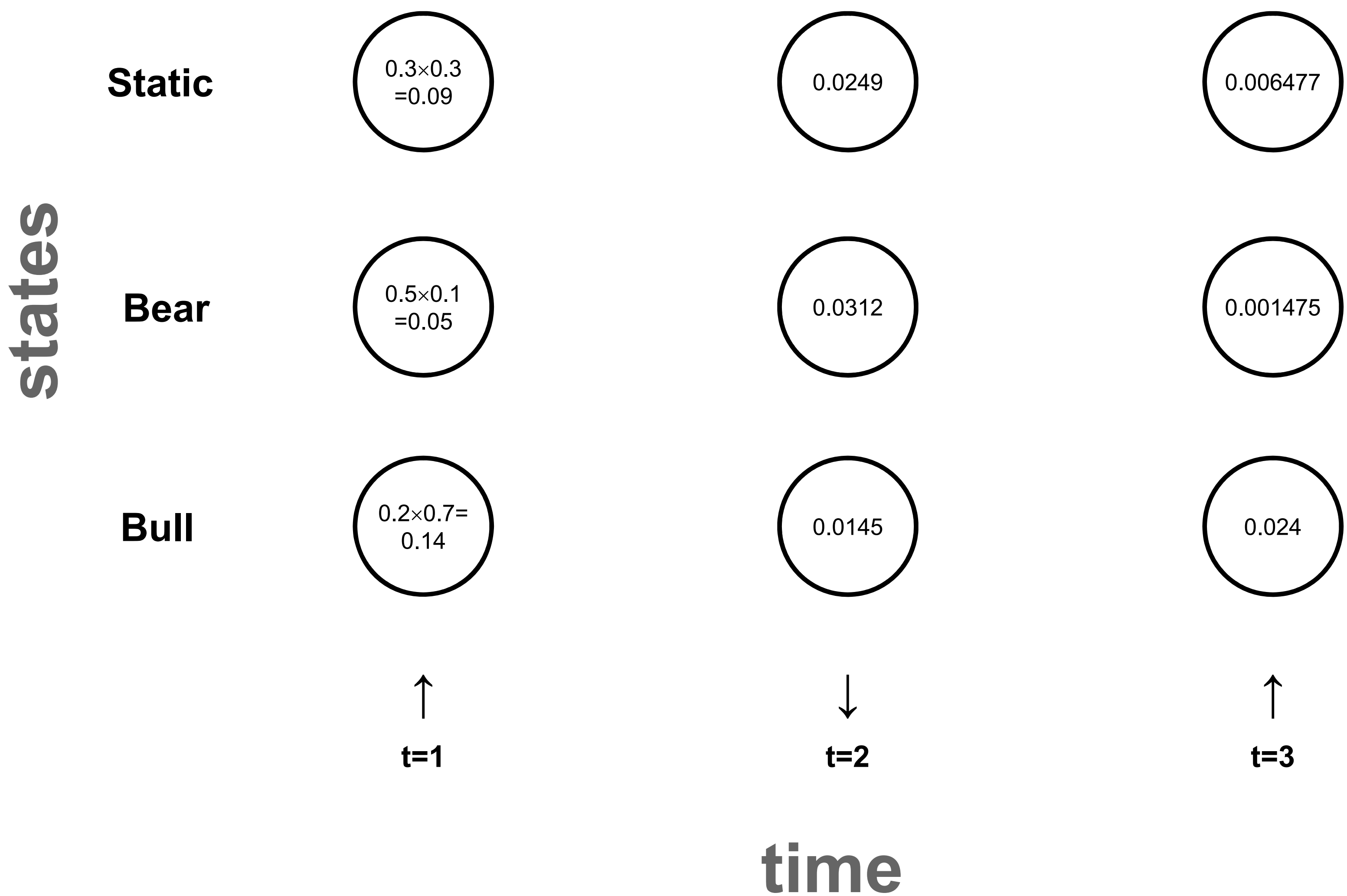
Forward Algorithm: Recursion

Work through the rest of these numbers…



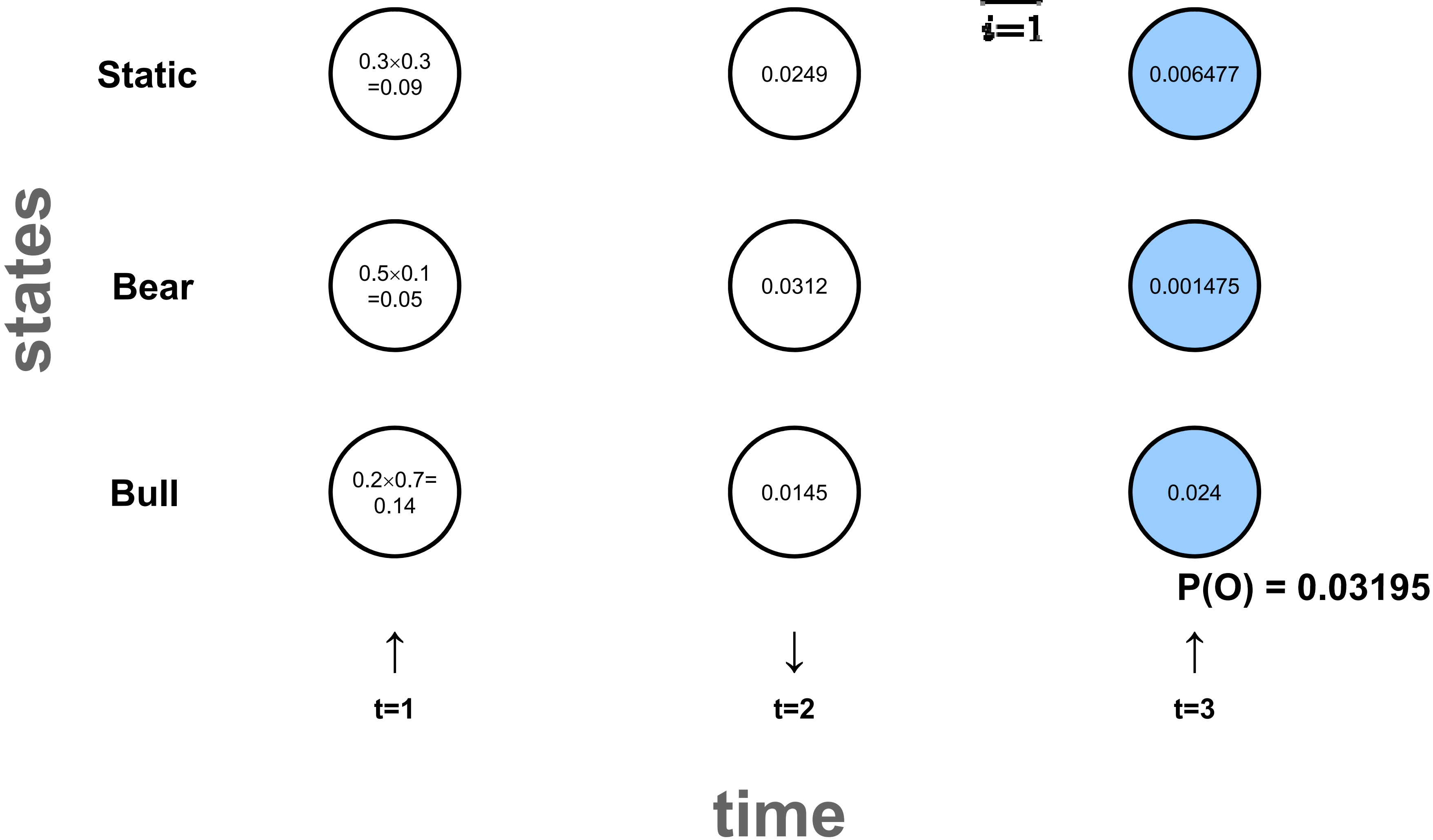
What's the asymptotic complexity of this algorithm?

Forward Algorithm: Recursion



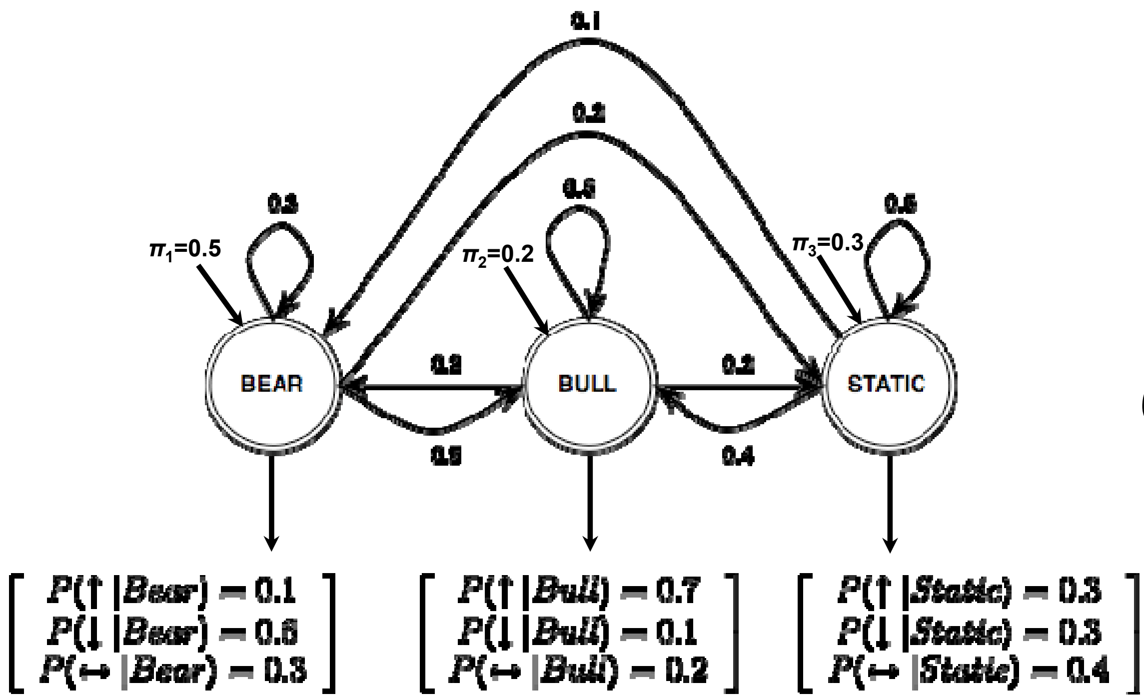
Forward Algorithm: Termination

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$



HMM Problem #2: Decoding

Decoding



λ_{stock}

t:	1	2	3	4	5	6
O:	↑	↓	↔	↑	↓	↔

Given λ_{stock} as our model and *O* as our observations, what are the most likely states the market went through to produce *O*?

Decoding

- "Decoding" because states are hidden
- First try:
  - Compute $P(O)$ for all possible state sequences, then choose sequence with highest probability
  - What's the problem here?
- Second try:
  - For each possible hidden state sequence, compute $P(O)$ using the forward algorithm
  - What's the problem here?

Viterbi Algorithm

- "Decoding" = computing most likely state sequence
  - Another dynamic programming algorithm
  - Efficient: polynomial vs. exponential (brute force)
- Same idea as the forward algorithm
  - Store intermediate computation results in a trellis
  - Build new cells from existing cells

Viterbi Algorithm

- Use an $N \times T$ trellis $[v_{tj}]$
  - Just like in forward algorithm
- v_{tj} or $v_t(j)$
  - $= P$(in state j after seeing t observations and passing through the most likely state sequence so far)
  - $= P(q_1, q_2, \dots q_{t-1}, q_{t=j}, o_1, o_2, \dots o_t)$
- Each cell = extension of most likely path from other cells
  $v_t(j) = \max_i v_{t-1}(i)\, a_{ij}\, b_j(o_t)$
  - $v_{t-1}(i)$: Viterbi probability until $(t-1)$
  - a_{ij}: transition probability of going from state i to j
  - $b_j(o_t)$: probability of emitting symbol o_t in state j
- $P = \max_i v_T(i)$

Viterbi vs. Forward

- Maximization instead of summation over previous paths
- This algorithm is still missing something!
  - In forward algorithm, we only care about the probabilities
  - What's different here?
- We need to store the most likely path (transition):
  - Use "backpointers" to keep track of most likely transition
  - At the end, follow the chain of backpointers to recover the most likely state sequence

Viterbi Algorithm: Formal Definition

- Initialization

$$
\begin{aligned}
v_1(j) &= \pi_i b_i(o_1); 1 \leq i \leq N \\
BT_1(i) &= 0
\end{aligned}
$$

- Recursion

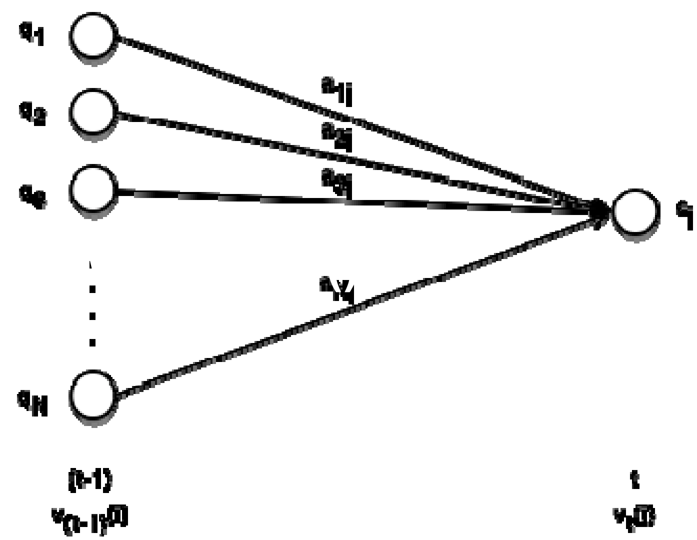
$$
\begin{aligned}
v_t(j) &= \max_{i=1}^{N} [v_{t-1}(i) a_{ij}]\, b_j(o_t); 1 \leq i \leq N, 2 \leq t \leq T \\
BT_1(i) &= \arg\max_{i=1}^{N} [v_{t-1}(i) a_{ij}]
\end{aligned}
$$

But here?

Why no $b_j(o_t)$ here?

- Termination

$$
\begin{aligned}
P^* &= \max_{1=1}^{N} v_T(j) \\
q_T^* &= \arg\max_{1=i}^{N} v_T(j)
\end{aligned}
$$

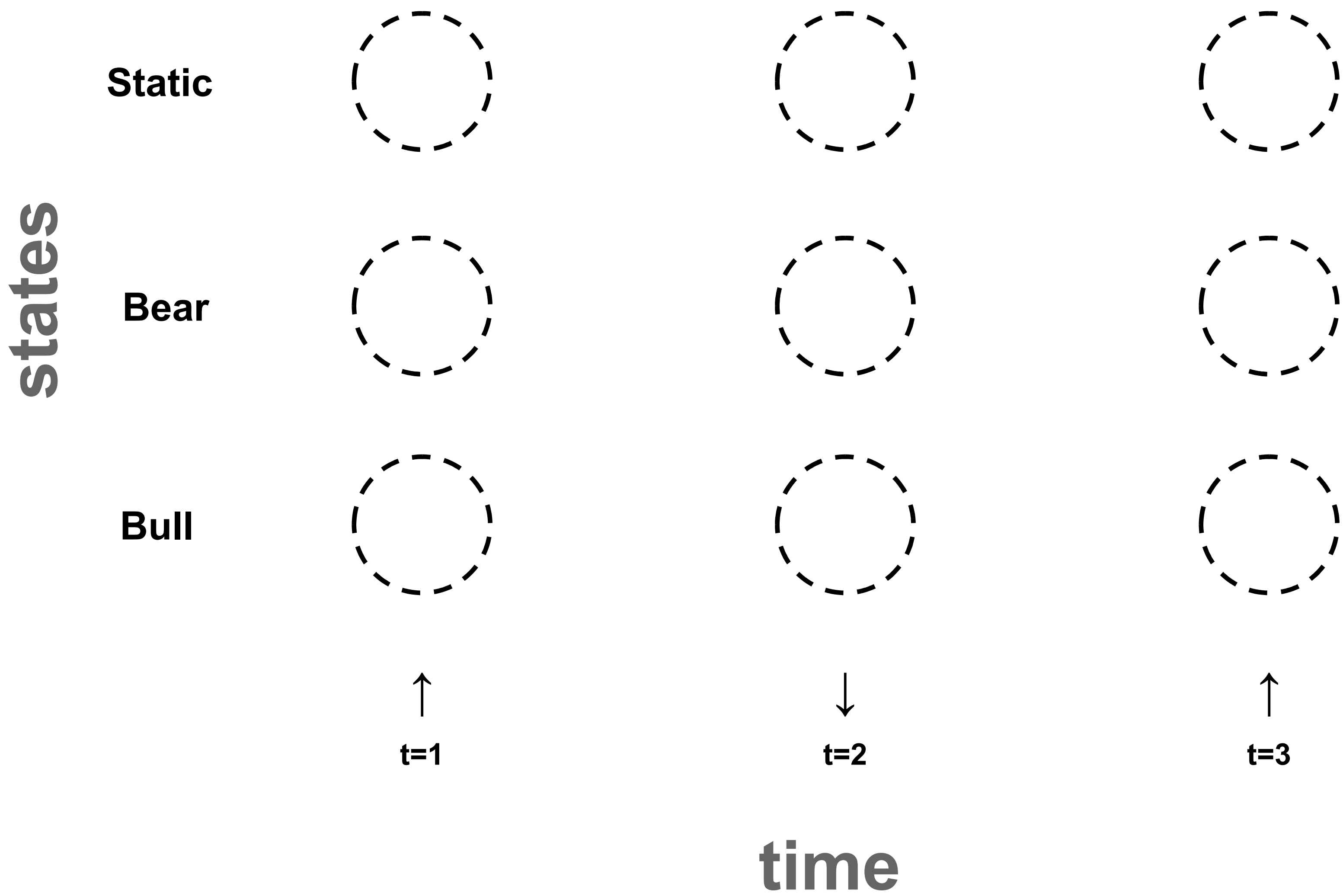


Viterbi Algorithm

$$O = \uparrow \downarrow \uparrow$$

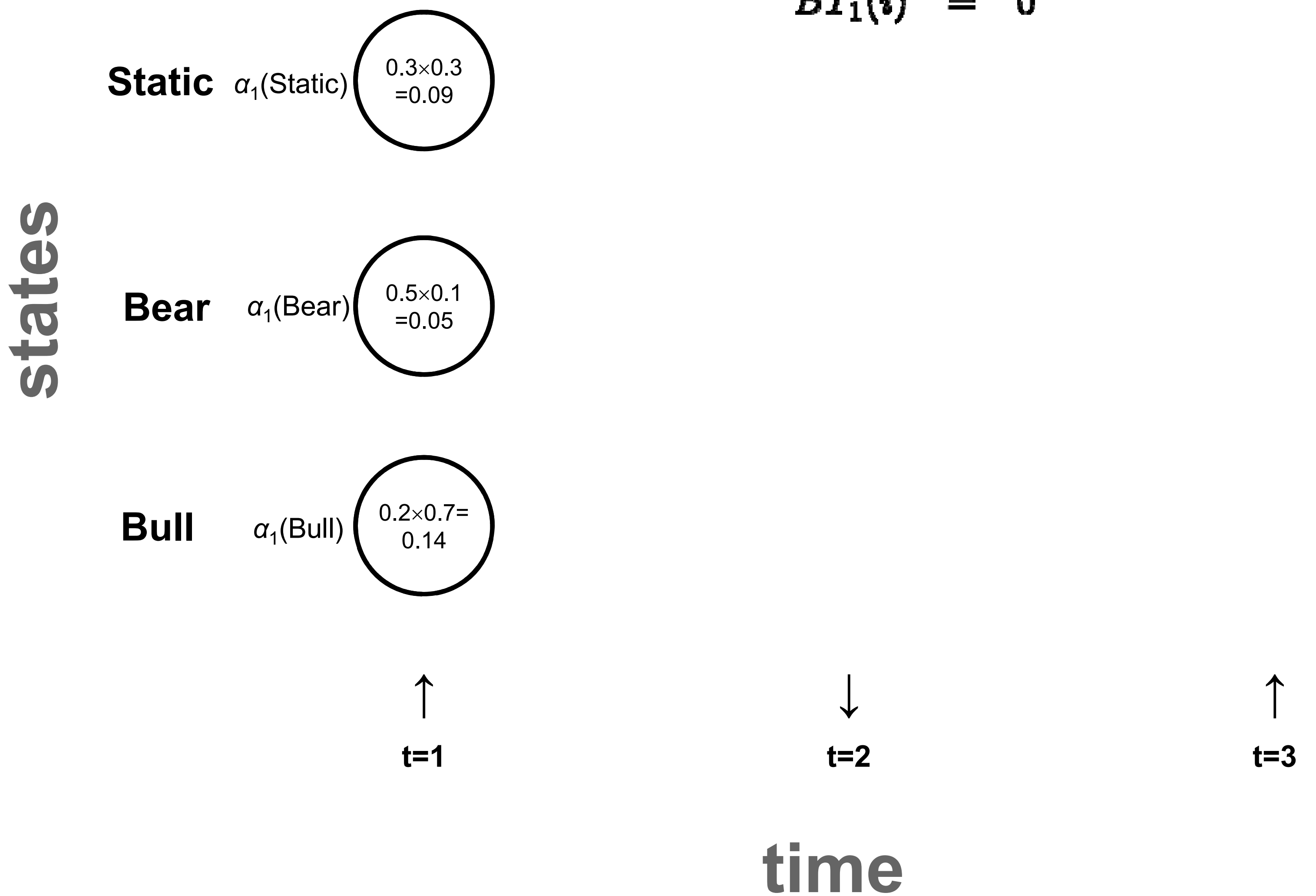
find most likely state sequence given λ_{stock}

Viterbi Algorithm



Viterbi Algorithm: Initialization

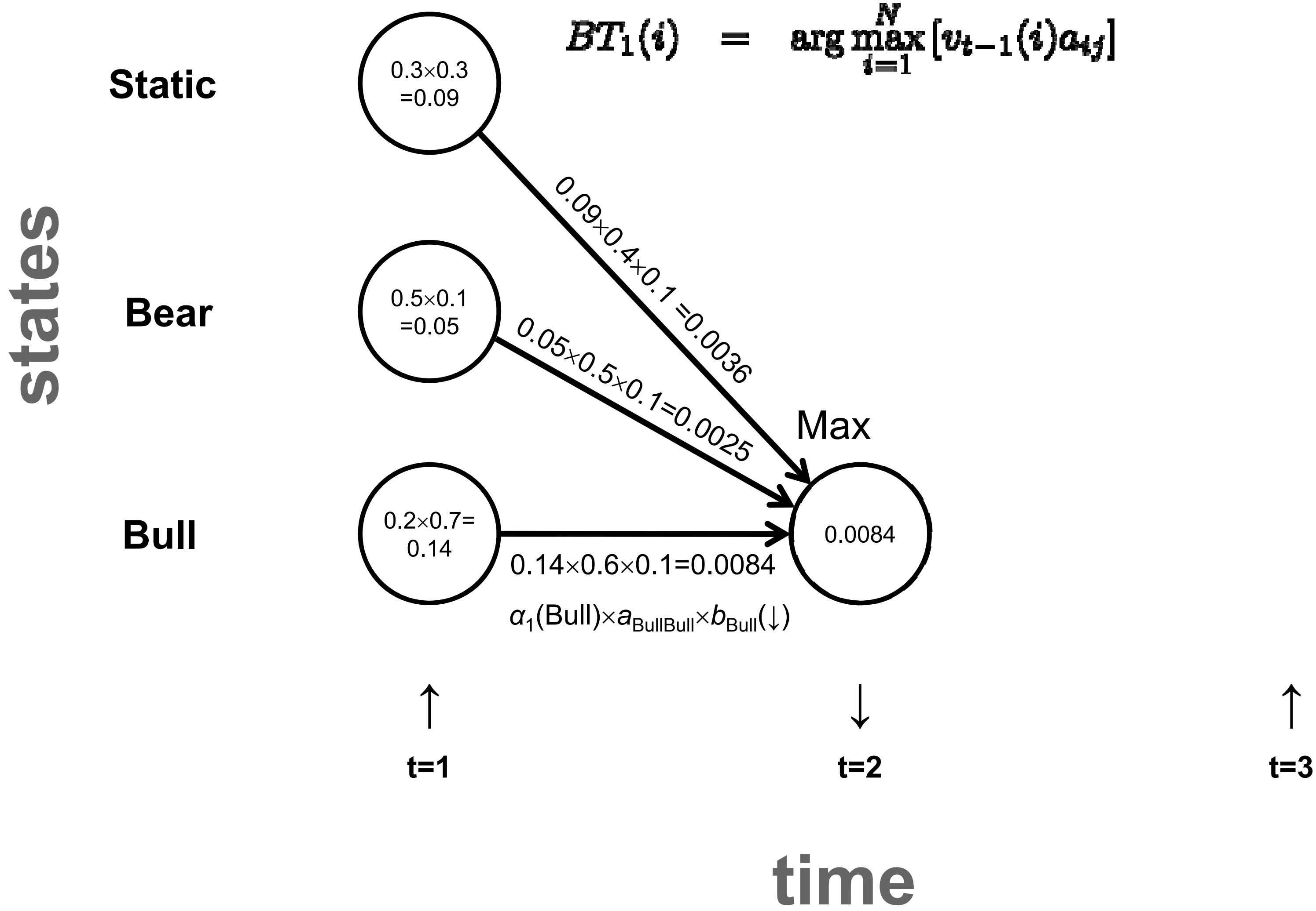
$$
\begin{aligned}
v_1(j) &= \pi_i b_i(o_1); 1 \leq i \leq N \\
BT_1(i) &= 0
\end{aligned}
$$



Viterbi Algorithm: Recursion

$$v_t(j) = \max_{i=1}^{N} [v_{t-1}(i) a_{ij}]\, b_j(o_t);\ 1 \le i \le N, 2 \le t \le T$$

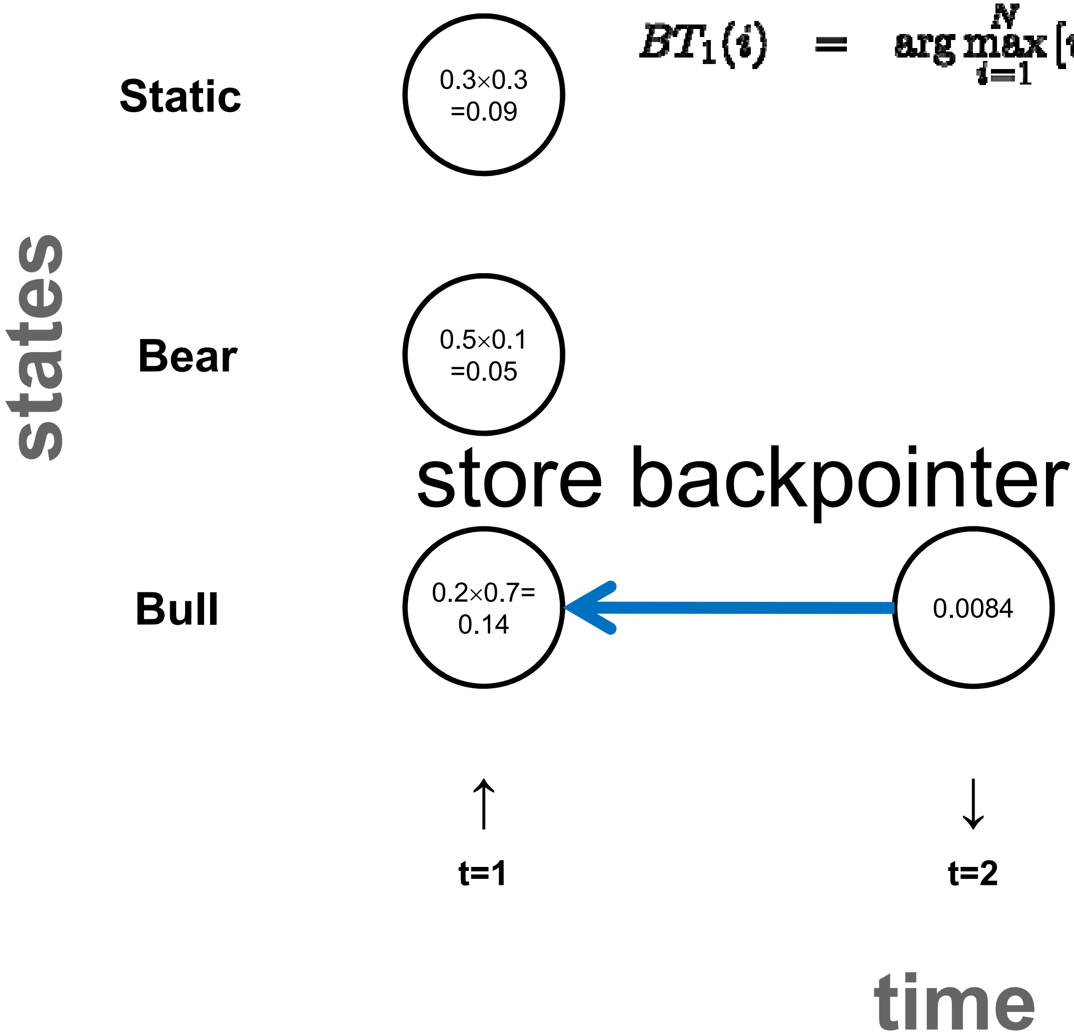
$$BT_1(i) = \arg\max_{i=1}^{N} [v_{t-1}(i) a_{ij}]$$



Viterbi Algorithm: Recursion

$$v_t(j) = \max_{i=1}^{N} \left[v_{t-1}(i) a_{ij}\right] b_j(o_t); 1 \leq i \leq N, 2 \leq t \leq T$$

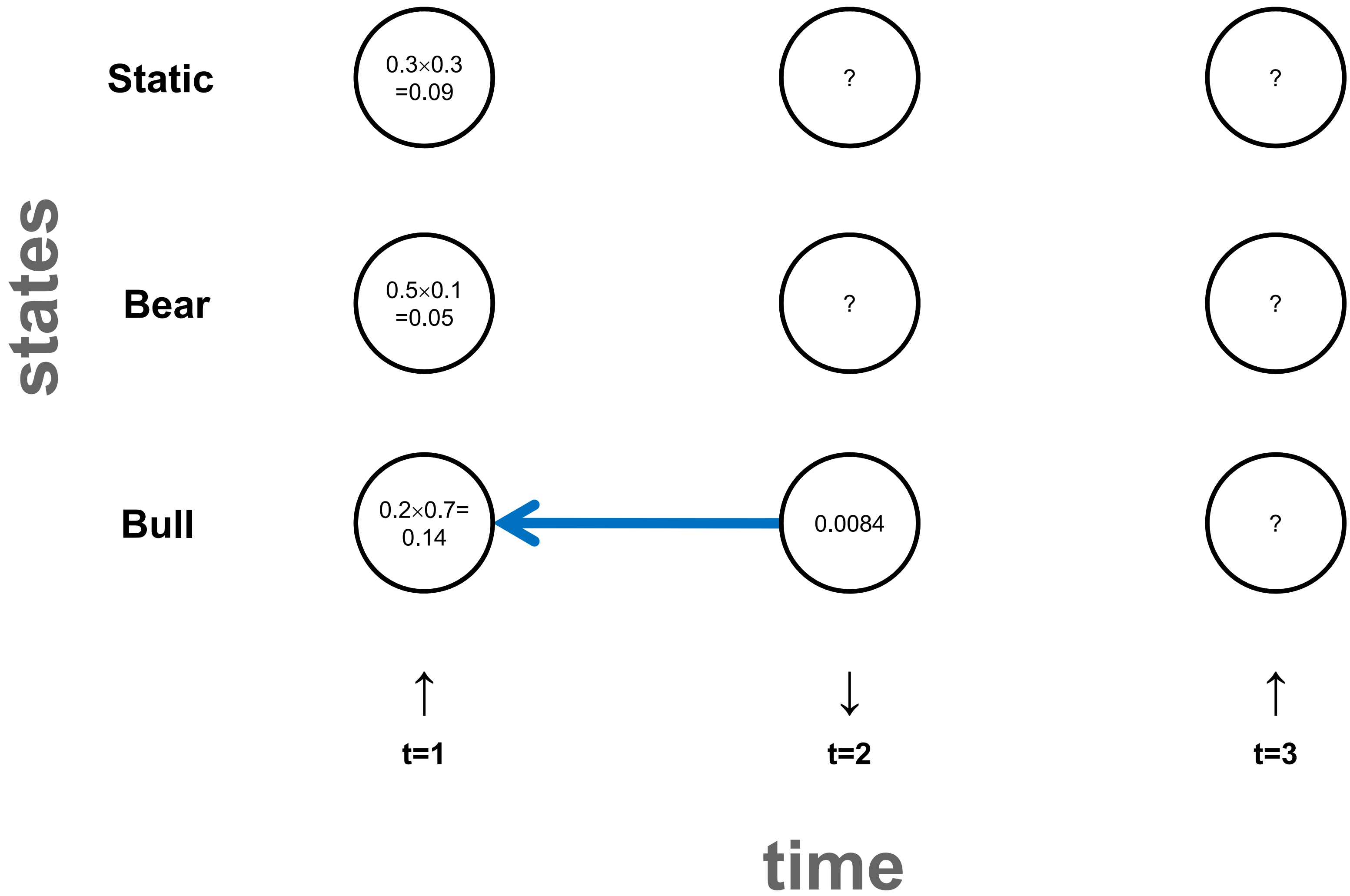
$$BT_1(i) = \arg\max_{i=1}^{N} \left[v_{t-1}(i) a_{ij}\right]$$



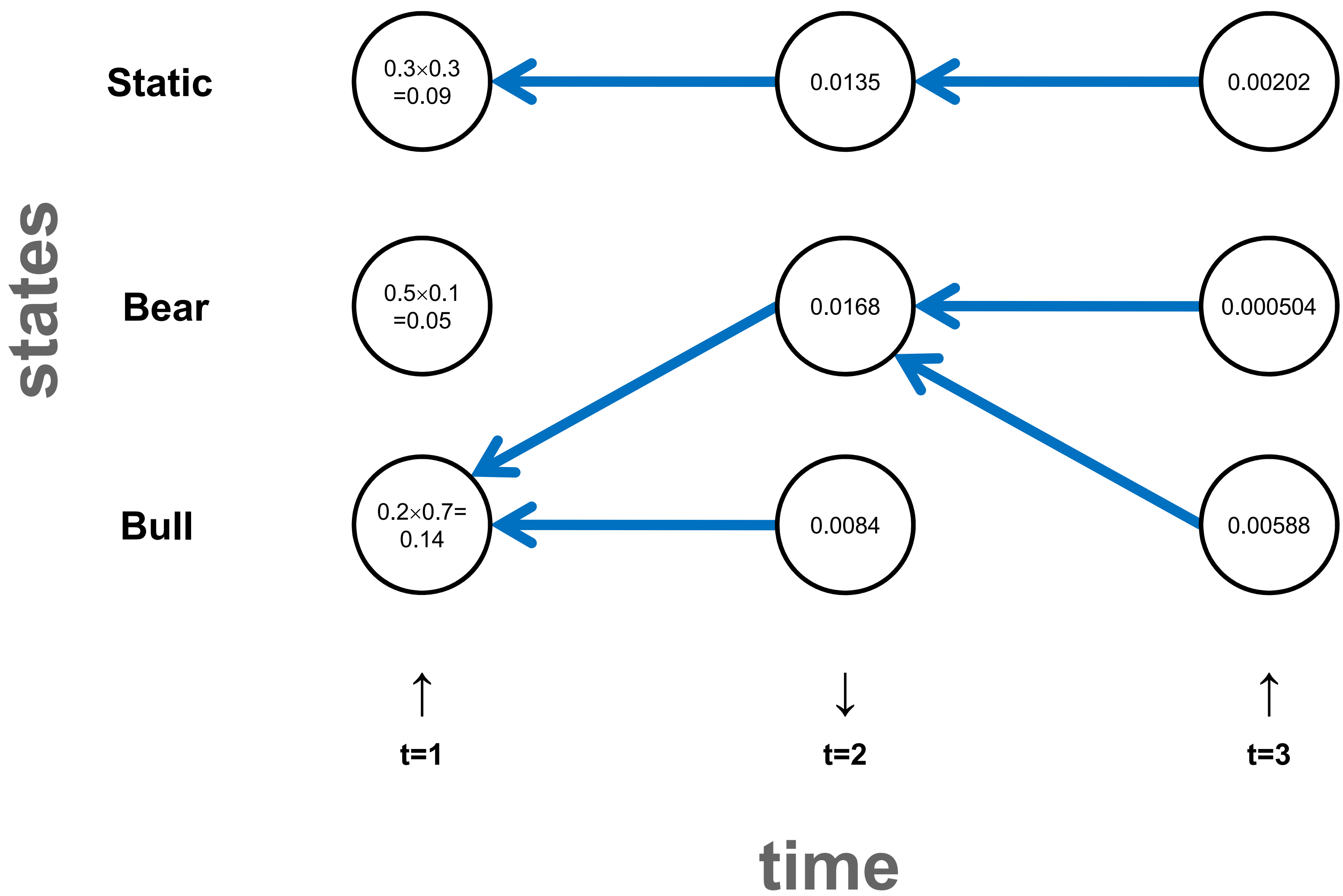
.... and so on

Viterbi Algorithm: Recursion

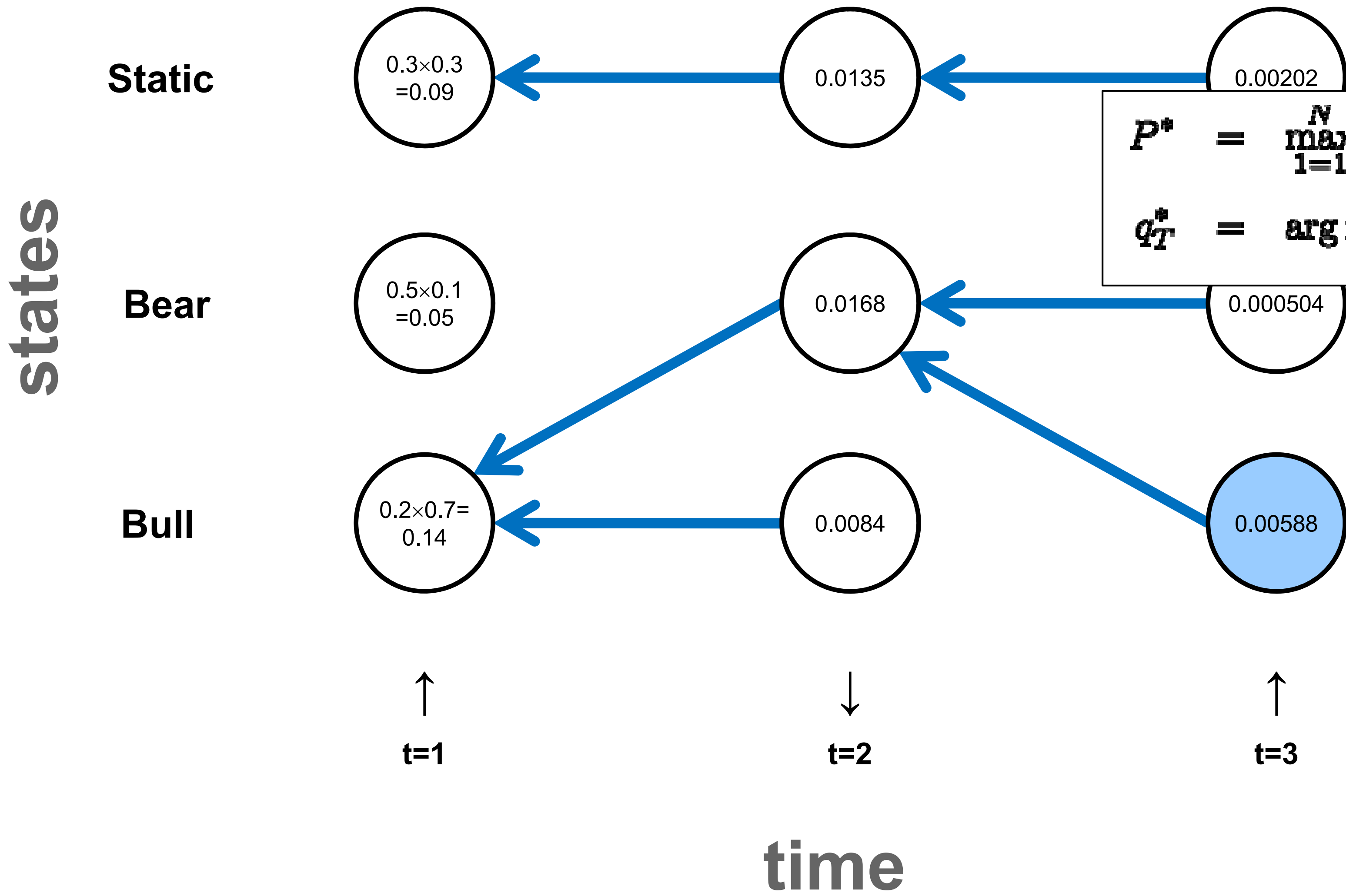
Work through the rest of the algorithm…



Viterbi Algorithm: Recursion



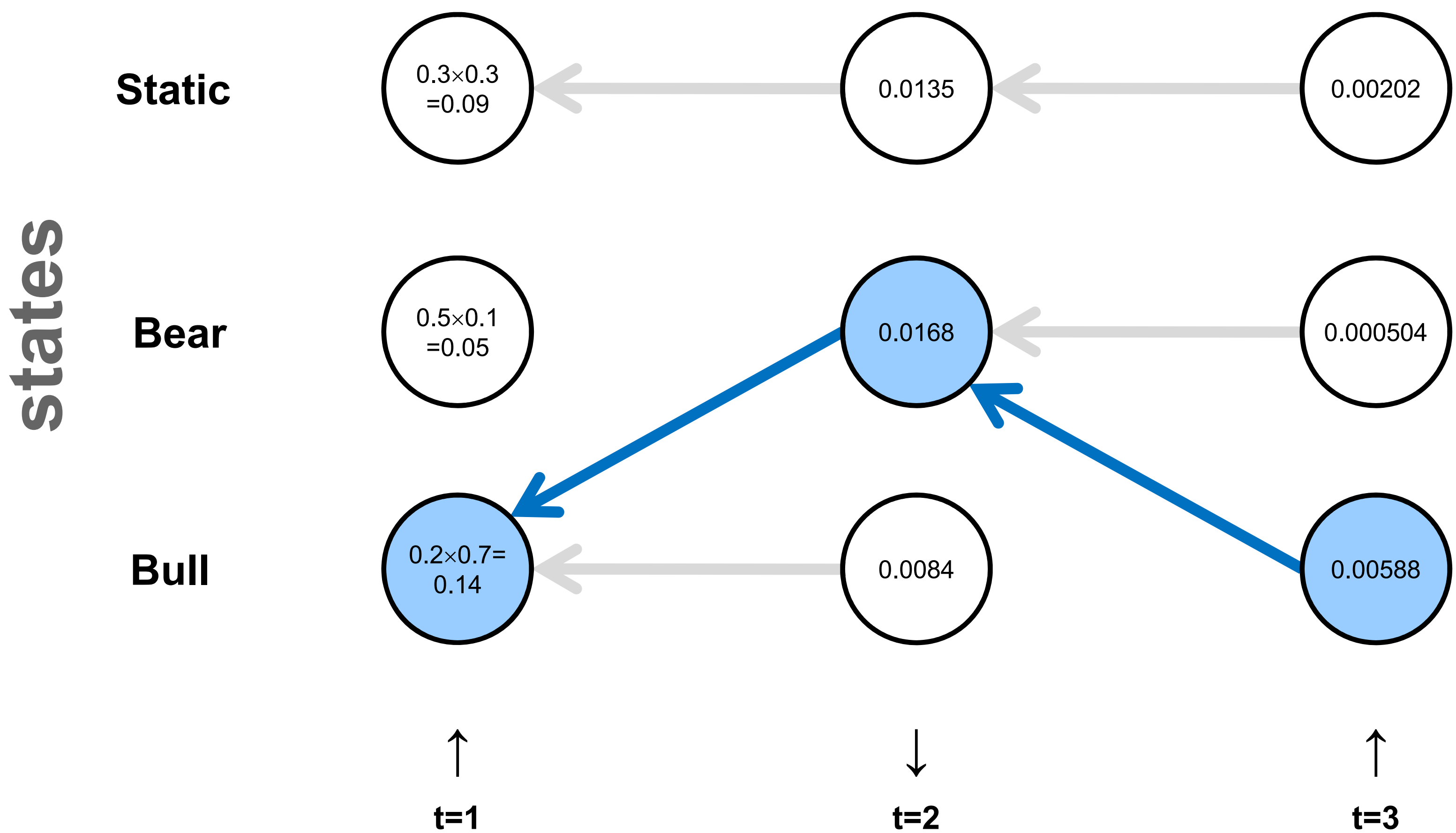
Viterbi Algorithm: Termination



$$P^* = \max_{1=1}^{N} v_T(j)$$

$$q_T^* = \operatorname{arg}\max_{1=i}^{N} v_T(j)$$

Viterbi Algorithm: Termination



Most likely state sequence:
[Bull, Bear, Bull], *P* = 0.00588

POS Tagging with HMMs

Modeling the problem

- What's the problem?
  - The/DT grand/JJ jury/NN commmented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.
- What should the HMM look like ?
  - States: part-of-speech tags (t_1, t_2, ..., t_N)
  - Output symbols: words (w_1, w_2, ..., $w_{|V|}$)
- Given HMM λ (A, B, ∏), POS tagging = reconstructing the best state sequence given input
  - Use Viterbi decoding (best = most likely)
- But wait…

HMM Training

- What are appropriate values for *A*, *B*, ∏?
- Before HMMs can decode, they must be trained…
  - *A*: transition probabilities
  - *B*: emission probabilities
  - ∏: prior
- Two training methods:
  - Supervised training: start with tagged corpus, count stuff to estimate parameters
  - Unsupervised training: start with untagged corpus, bootstrap parameter estimates and improve estimates iteratively

HMMs: Three Problems

- **Likelihood:** Given an HMM $\lambda = (A, B, \prod)$, and a sequence of observed events O, find $P(O|\lambda)$
- **Decoding:** Given an HMM $\lambda = (A, B, \prod)$, and an observation sequence O, find the most likely (hidden) state sequence
- **Learning:** Given a set of observation sequences and the set of states Q in λ, compute the parameters A and B

Supervised Training

- A tagged corpus tells us the hidden states!
- We can compute Maximum Likelihood Estimates (MLEs) for the various parameters
  - MLE = fancy way of saying “count and divide”
- These parameter estimates maximize the likelihood of the data being generated by the model

Supervised Training

- Transition Probabilities
  - Any $P(t_i \mid t_{i-1}) = C(t_{i-1}, t_i) / C(t_{i-1})$, from the tagged data
  - Example: for P(NN|VB), count how many times a noun follows a verb and divide by the total number of times you see a verb
- Emission Probabilities
  - Any $P(w_i \mid t_i) = C(w_i, t_i) / C(t_i)$, from the tagged data
  - For P(bank|NN), count how many times bank is tagged as a noun and divide by how many times anything is tagged as a noun
- Priors
  - Any $P(q_1 = t_i) = \pi_i = C(t_i)/N$, from the tagged data
  - For π_{NN} , count the number of times NN occurs and divide by the total number of tags (states)
  - A better way?

Unsupervised Training

- No labeled/tagged training data
- No way to compute MLEs directly
- How do we deal?
  - Make an initial guess for parameter values
  - Use this guess to get a better estimate
  - Iteratively improve the estimate until some convergence criterion is met

Expectation Maximization (EM)

Expectation Maximization

- A fundamental tool for unsupervised machine learning techniques
- Forms basis of state-of-the-art systems in MT, parsing, WSD, speech recognition and more

Motivating Example

- Let observed events be the grades given out in, say, CMSC723
- Assume grades are generated by a probabilistic model described by single parameter μ
  - P(A) = 1/2, P(B) = μ, P(C) = 2 μ, P(D) = 1/2 - 3 μ
  - Number of ‘A’s observed = ‘a’, ‘b’ number of ‘B’s, etc.
- Compute MLE of μ given ‘a’, ‘b’, ‘c’ and ‘d’

Motivating Example

- Recall the definition of MLE:
  "…. maximizes likelihood of data given the model."
- Okay, so what's the likelihood of data given the model?
  - $P(Data|Model) = P(a,b,c,d|\mu) = (1/2)^a(\mu)^b(2\mu)^c(1/2-3\mu)^d$
  - L = log-likelihood = $\log P(a,b,c,d|\mu)$
    $= a \log(1/2) + b \log \mu + c \log 2\mu + d \log(1/2-3\mu)$
- How to maximize L w.r.t μ ? [Think Calculus]
  - $\delta L/\delta\mu = 0$; $(b/\mu) + (2c/2\mu) - (3d/(1/2-3\mu)) = 0$
  - $\mu = (b+c)/6(b+c+d)$
- We got our answer without EM. Boring!

Motivating Example

- Now suppose:
  - P(A) = 1/2, P(B) = μ, P(C) = 2 μ, P(D) = 1/2 - 3 μ
  - Number of ‘A’s and ‘B’s = h, c ‘C’s, and d ‘D’s
- Part of the observable information is hidden
- Can we compute the MLE for μ now?
- Chicken and egg:
  - If we knew ‘b’ (and hence ‘a’), we could compute the MLE for μ
  - But we need μ to know how the model generates ‘a’ and ‘b’
- Circular enough for you?

The EM Algorithm

- Start with an initial guess for μ (μ_0)
- $t = 1$; Repeat:
  - $b_t = \mu_{(t-1)} h / (1/2 + \mu_{(t-1)})$
    [**E-step:** Compute expected value of b given μ]
  - $\mu_t = (b_t + c)/6(b_t + c + d)$
    [**M-step:** Compute MLE of μ given b]
  - $t = t + 1$
- Until some convergence criterion is met

The EM Algorithm

- Algorithm to compute MLEs for model parameters when information is hidden
- Iterate between Expectation (E-step) and Maximization (M-step)
- Each iteration is guaranteed to increase the log-likelihood of the data (improve the estimate)
- Good news: It will always converge to a maximum
- Bad news: It will always converge to a maximum

Applying EM to HMMs

- Just the intuition… gory details in CMSC 773
- The problem:
  - State sequence is unknown
  - Estimate model parameters: A, B & ∏
- Introduce two new observation statistics:
  - Number of transitions from q_i to q_j (ξ)
  - Number of times in state q_i (Υ)
- The EM algorithm can now be applied

Applying EM to HMMs

- Start with initial guesses for A, B and ∏
- t = 1; Repeat:
  - E-step: Compute expected values of ξ, Υ using A_t, B_t, $\prod_t$
  - M-step: Compute MLE of A, B and ∏ using ξ_t, Υ_t
  - $t = t + 1$
- Until some convergence criterion is met

What we covered today…

- The great leap forward in NLP
- Hidden Markov models (HMMs)
  - Forward algorithm
  - Viterbi decoding
  - Supervised training
  - Unsupervised training teaser
- HMMs for POS tagging