

Data-Intensive Computing with MapReduce

Session 8: Sequence Labeling

Jimmy Lin
University of Maryland
Thursday, March 14, 2013



This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States
See <http://creativecommons.org/licenses/by-nc-sa/3.0/us/> for details



Today's Agenda

- Sequence labeling
- Midterm
- Final projects

Sequences Everywhere!

- Examples of sequences
 - Waveforms in utterances
 - Letters in words
 - Words in sentences
 - Bases in DNA
 - Proteins in amino acids
 - Actions in video
 - Closing prices in the stock market
- All share the property of having structural dependencies

Sequence Labeling

- Given a sequence of observations, assign a label to each observation
 - Observations can be discrete or continuous, scalar or vector
 - Assume labels are drawn from a discrete finite set
- Sample applications:
 - Speech recognition
 - POS tagging
 - Handwriting recognition
 - Video analysis
 - Protein secondary structure detection

Approaches

- Treat as a classification problem?
 - Make decisions about each observation independently
- Can we do better?

As with before, lots of background material
before we get to MapReduce!

MapReduce Implementation: Mapper

```

1: class MAPPER
2:   method INITIALIZE(integer iteration)
3:      $\langle \mathcal{S}, \mathcal{O} \rangle \leftarrow \text{READMODEL}$ 
4:      $\theta \leftarrow \langle A, B, \pi \rangle \leftarrow \text{READMODELPARAMS}(iteration)$ 
5:   method MAP(sample id, sequence  $\mathbf{x}$ )
6:      $\alpha \leftarrow \text{FORWARD}(\mathbf{x}, \theta)$ 
7:      $\beta \leftarrow \text{BACKWARD}(\mathbf{x}, \theta)$ 
8:      $I \leftarrow \text{new ASSOCIATIVEARRAY}$ 
9:     for all  $q \in \mathcal{S}$  do
10:        $I\{q\} \leftarrow \alpha_1(q) \cdot \beta_1(q)$ 
11:      $O \leftarrow \text{new ASSOCIATIVEARRAY of ASSOCIATIVEARRAY}$ 
12:     for  $t = 1$  to  $|\mathbf{x}|$  do
13:       for all  $q \in \mathcal{S}$  do
14:          $O\{q\}\{x_t\} \leftarrow O\{q\}\{x_t\} + \alpha_t(q) \cdot \beta_t(q)$ 
15:        $t \leftarrow t + 1$ 
16:      $T \leftarrow \text{new ASSOCIATIVEARRAY of ASSOCIATIVEARRAY}$ 
17:     for  $t = 1$  to  $|\mathbf{x}| - 1$  do
18:       for all  $q \in \mathcal{S}$  do
19:         for all  $r \in \mathcal{S}$  do
20:            $T\{q\}\{r\} \leftarrow T\{q\}\{r\} + \alpha_t(q) \cdot A_q(r) \cdot B_r(x_{t+1}) \cdot \beta_{t+1}(r)$ 
21:        $t \leftarrow t + 1$ 
22:     EMIT(string 'initial', stripe  $I$ )
23:     for all  $q \in \mathcal{S}$  do
24:       EMIT(string 'emit from ' +  $q$ , stripe  $O\{q\}$ )
25:       EMIT(string 'transit from ' +  $q$ , stripe  $T\{q\}$ )

```

$$\hat{b}_j(v_k) = \frac{\sum_{i=1}^T \gamma_t(j)}{\sum_{i=1}^T \gamma_t(i)}$$

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \sum_{j=1}^N \xi_t(i, j)}$$

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)}$$

$$\xi_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{P(O|\lambda)}$$

MapReduce Implementation: Reducer

```

1: class COMBINER
2:   method COMBINE(string  $t$ , stripes [ $C_1, C_2, \dots$ ])
3:      $C_f \leftarrow$  new ASSOCIATIVEARRAY
4:     for all stripe  $C \in$  stripes [ $C_1, C_2, \dots$ ] do
5:       SUM( $C_f, C$ )
6:     EMIT(string  $t$ , stripe  $C_f$ )
7:
8: class REDUCER
9:   method REDUCE(string  $t$ , stripes [ $C_1, C_2, \dots$ ])
10:     $C_f \leftarrow$  new ASSOCIATIVEARRAY
11:    for all stripe  $C \in$  stripes [ $C_1, C_2, \dots$ ] do
12:      SUM( $C_f, C$ )
13:     $z \leftarrow 0$ 
14:    for all  $\langle k, v \rangle \in C_f$  do
15:       $z \leftarrow z + v$ 
16:     $P_f \leftarrow$  new ASSOCIATIVEARRAY
17:    for all  $\langle k, v \rangle \in C_f$  do
18:       $P_f\{k\} \leftarrow v/z$ 
19:    EMIT(string  $t$ , stripe  $P_f$ )

```

$$\hat{b}_j(v_k) = \frac{\sum_{i=1}^T \gamma_t(j)}{\sum_{i=1}^T \gamma_t(i)}$$

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \sum_{j=1}^N \xi_t(i, j)}$$

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)}$$

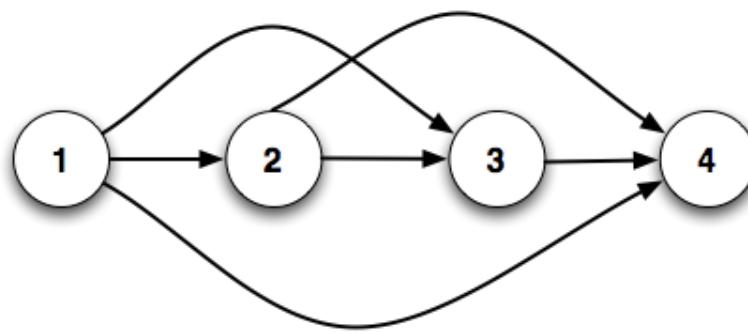
$$\xi_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{P(O|\lambda)}$$

Finite State Machines

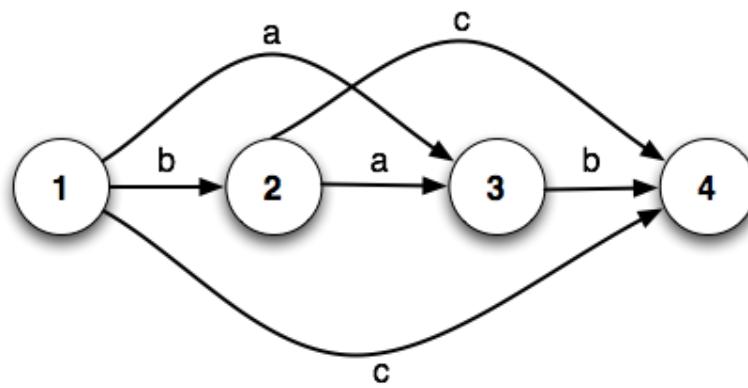
- Q : a finite set of N states
 - $Q = \{q_0, q_1, q_2, q_3, \dots\}$
 - The start state: q_0
 - The set of final states: q_F
- Σ : a finite input alphabet of symbols
- $\delta(q,i)$: transition function
 - Given state q and input symbol i , transition to new state q'



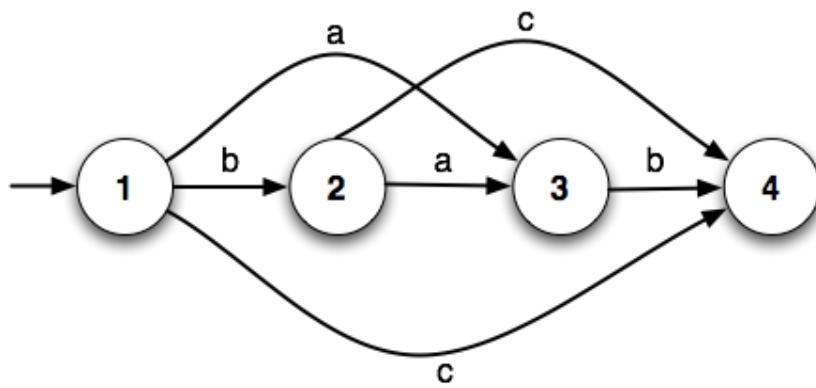
Finite number of states



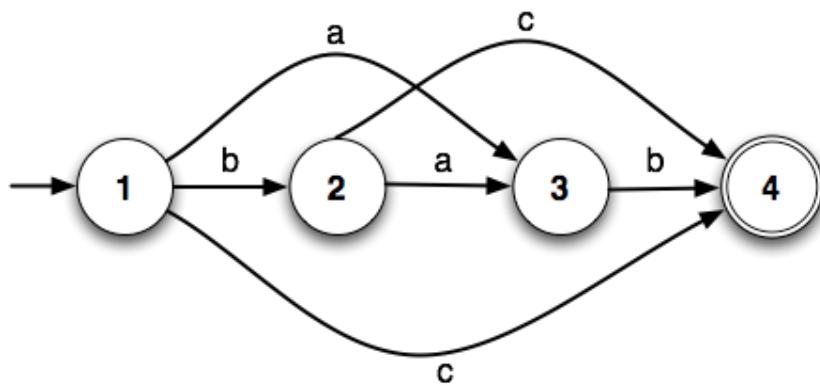
Transitions



Input alphabet



Start state



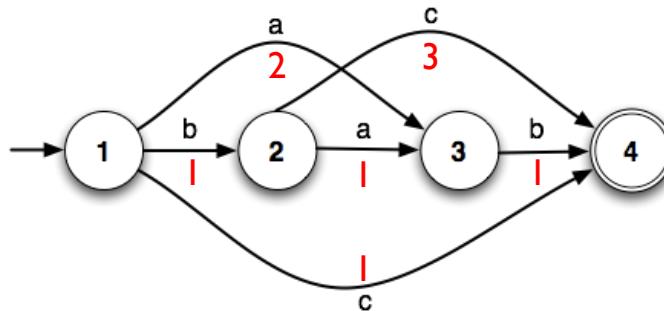
Final state(s)

What can we do with FSMs?

- Generate valid sequences
- Accept valid sequences

Weighted FSMs

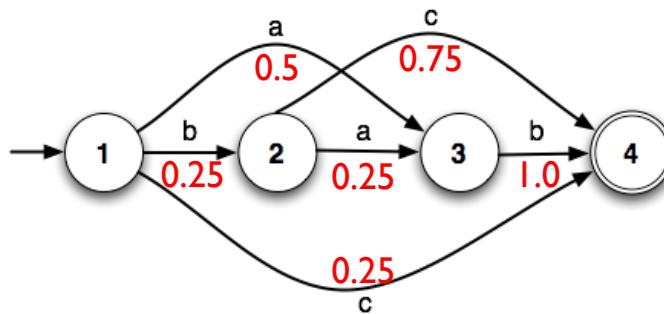
- FSMs treat all transitions as equally likely
- What if we know more about state transitions?
 - ‘a’ is twice as likely to be seen in state 1 as ‘b’ or ‘c’
 - ‘c’ is three times as likely to be seen in state 2 as ‘a’



- What do we get out of it?
 - $\text{score}(\text{'ab'}) = 2$
 - $\text{score}(\text{'bc'}) = 3$

Probabilistic FSMs = Markov Chains

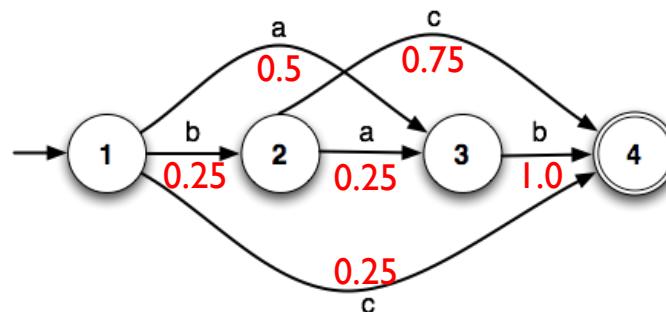
- Let's replace weights with probabilities
- What if we know more about state transitions?
 - 'a' is twice as likely to be seen in state 1 as 'b' or 'c'
 - 'c' is three times as likely to be seen in state 2 as 'a'



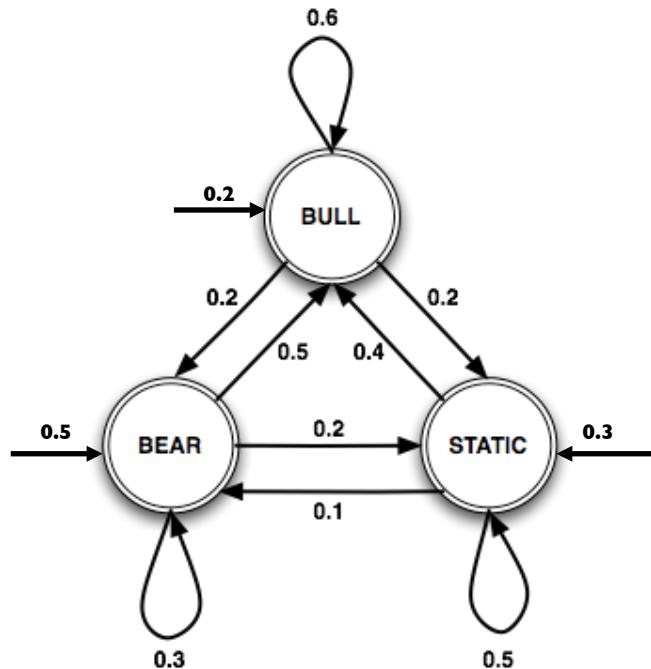
- What do we get out of it?
 - $P('ab') = 0.5$
 - $P('bc') = 0.1875$

Specifying Markov Chains

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



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
$$P(q_i | q_1 \dots q_{i-1}) = P(q_i | q_{i-1})$$

Are states always observable ?

Day: | 1 2 3 4 5 6

Not observable !

Bull Bear S Bear S Bull

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: Hidden Markov Model (HMM)
 - Assume two probabilistic processes
 - Underlying process (state transition) is hidden
 - Second process generates sequence of observed events

Specifying HMMs

An HMM $\lambda = (A, B, \Pi)$ is characterized by:

- N states: $Q = \{q_1, q_2, \dots, q_N\}$
- $N \times N$ Transition probability matrix $A = [a_{ij}]$

$$a_{ij} = p(q_j | q_i) \quad \sum_j a_{ij} = 1 \quad \forall i$$

- V observation symbols: $O = \{o_1, o_2, \dots, o_V\}$
- $N \times |V|$ Emission probability matrix $B = [b_{iv}]$

$$b_{iv} = b_i(o_v) = p(o_v | q_i)$$

- Prior probabilities vector $\Pi = [\pi_1, \pi_2, \dots, \pi_N]$

$$\sum_{i=1}^N \pi_i = 1$$

Stock Market HMM

States? 

Transitions?

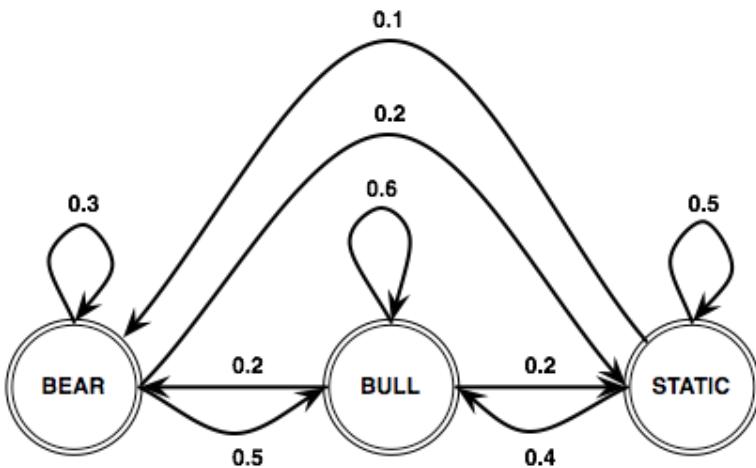
Vocabulary?

Emissions?

Priors?

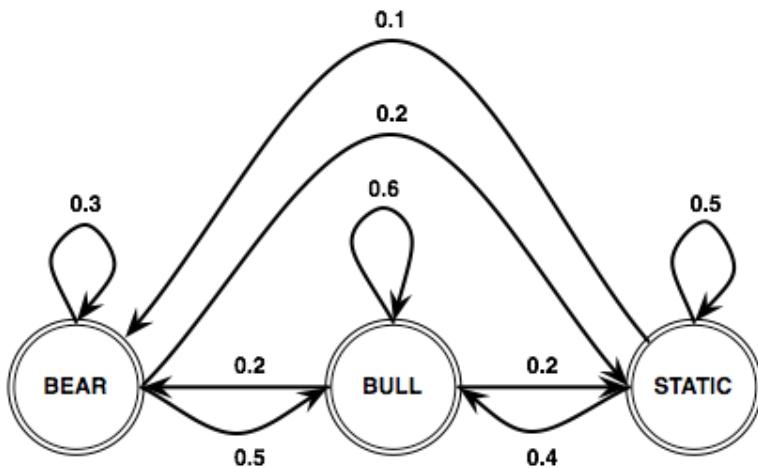


Stock Market HMM



- States? ✓
- Transitions? ✓
- Vocabulary?
- Emissions?
- Priors?

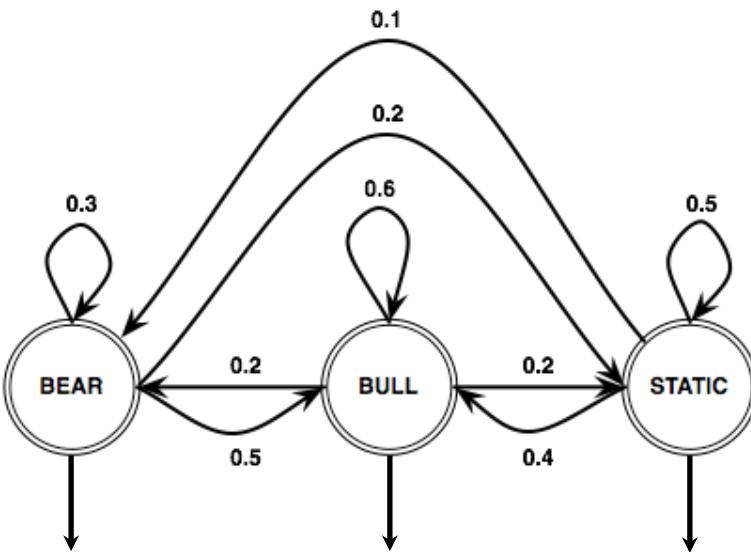
Stock Market HMM



- States? ✓
- Transitions? ✓
- Vocabulary? ✓
- Emissions?
- Priors?

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

Stock Market HMM



$$\begin{bmatrix} P(\uparrow | \text{Bear}) = 0.1 \\ P(\downarrow | \text{Bear}) = 0.6 \\ P(\leftrightarrow | \text{Bear}) = 0.3 \end{bmatrix}$$

$$\begin{bmatrix} P(\uparrow | \text{Bull}) = 0.7 \\ P(\downarrow | \text{Bull}) = 0.1 \\ P(\leftrightarrow | \text{Bull}) = 0.2 \end{bmatrix}$$

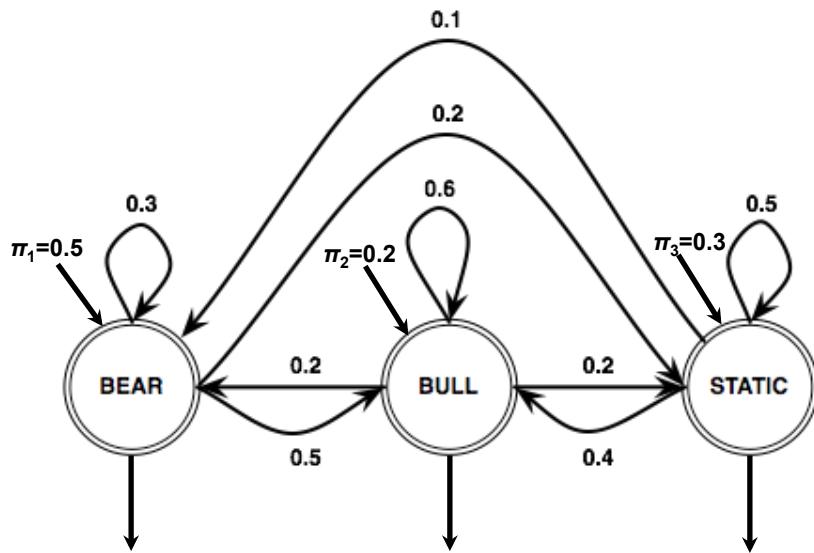
$$\begin{bmatrix} P(\uparrow | \text{Static}) = 0.3 \\ P(\downarrow | \text{Static}) = 0.3 \\ P(\leftrightarrow | \text{Static}) = 0.4 \end{bmatrix}$$

- States? ✓
Transitions? ✓
Vocabulary? ✓
Emissions? ✓
Priors?

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

Stock Market HMM

- States? ✓
- Transitions? ✓
- Vocabulary? ✓
- Emissions? ✓
- Priors? ✓



$$\begin{bmatrix} P(\uparrow | \text{Bear}) = 0.1 \\ P(\downarrow | \text{Bear}) = 0.6 \\ P(\leftrightarrow | \text{Bear}) = 0.3 \end{bmatrix}$$

$$\begin{bmatrix} P(\uparrow | \text{Bull}) = 0.7 \\ P(\downarrow | \text{Bull}) = 0.1 \\ P(\leftrightarrow | \text{Bull}) = 0.2 \end{bmatrix}$$

$$\begin{bmatrix} P(\uparrow | \text{Static}) = 0.3 \\ P(\downarrow | \text{Static}) = 0.3 \\ P(\leftrightarrow | \text{Static}) = 0.4 \end{bmatrix}$$

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

Properties of HMMs

- Markov assumption:

$$P(q_i | q_1 \dots q_{i-1}) = P(q_i | q_{i-1})$$

- Output independence:

$$P(o_i | q_1 \dots q_i \dots q_T, o_1 \dots o_i \dots o_T) = P(o_i | q_i)$$

- Remember, these are distinct:

- Number of states (N)
- The number of distinct observation symbols (V)
- The length of the sequence (T)

“Running” an HMM

1. Set $t = 1$, select initial state based on $\Pi = [\pi_i, \pi_2, \dots, \pi_N]$
2. Select an observation to emit based on $B = [b_{iv}]$
3. If $t = T$, then done
4. Select transition into a new state based on $A = [a_{ij}]$
5. Set $t = t + 1$
6. Goto step 2

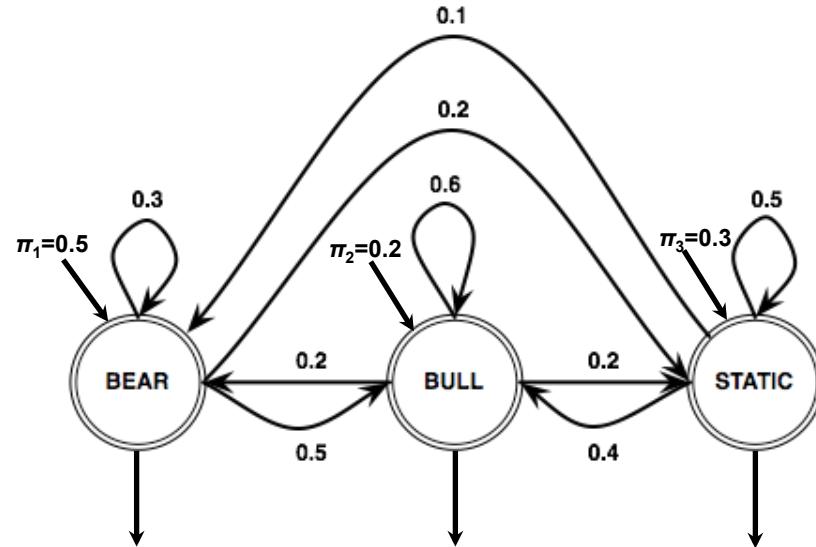
HMMs: Three Problems

- **Likelihood:** Given an HMM λ and a sequence of observed events O , find the likelihood of observing the sequence
- **Decoding:** Given an HMM λ and an observation sequence O , find the most likely (hidden) state sequence
- **Learning:** Given a set of observation sequences, learn the parameters A and B for λ

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

HMM Problem #1: Likelihood

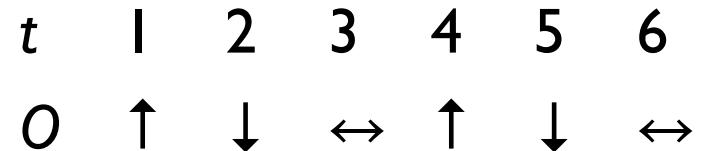
Computing Likelihood



$$\begin{bmatrix} P(\uparrow | \text{Bear}) = 0.1 \\ P(\downarrow | \text{Bear}) = 0.6 \\ P(\leftrightarrow | \text{Bear}) = 0.3 \end{bmatrix}$$

$$\begin{bmatrix} P(\uparrow | \text{Bull}) = 0.7 \\ P(\downarrow | \text{Bull}) = 0.1 \\ P(\leftrightarrow | \text{Bull}) = 0.2 \end{bmatrix}$$

$$\begin{bmatrix} P(\uparrow | \text{Static}) = 0.3 \\ P(\downarrow | \text{Static}) = 0.3 \\ P(\leftrightarrow | \text{Static}) = 0.4 \end{bmatrix}$$



Given this model of 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

$$P(O|\lambda) = \sum_Q P(O, Q|\lambda) = \sum_Q P(O|Q, \lambda)P(Q|\lambda)$$

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

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

Computing Likelihood

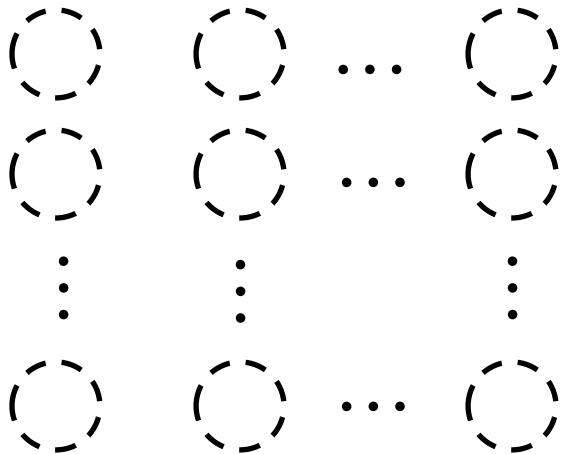
- What are we doing wrong?
 - State sequences may have a lot of overlap
 - We're recomputing the shared subsequences every time
 - Let's store intermediate results and reuse them
- Sounds like a job for dynamic programming!

Forward Algorithm

- Forward probability
 - Probability of being in state j after t observations

$$\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda)$$

- Build an $N \times T$ trellis



- Intuition: forward probability only depends on the previous state and observation

Forward Algorithm

$$\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda)$$

- Initialization

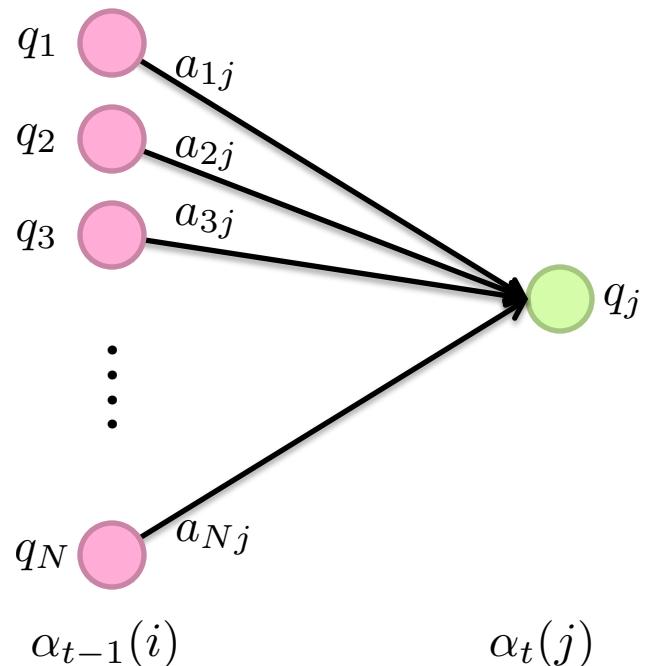
$$\alpha_1(j) = \pi_j b_j(o_i) \quad 1 \leq j \leq N$$

- Recursion

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

- Termination

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

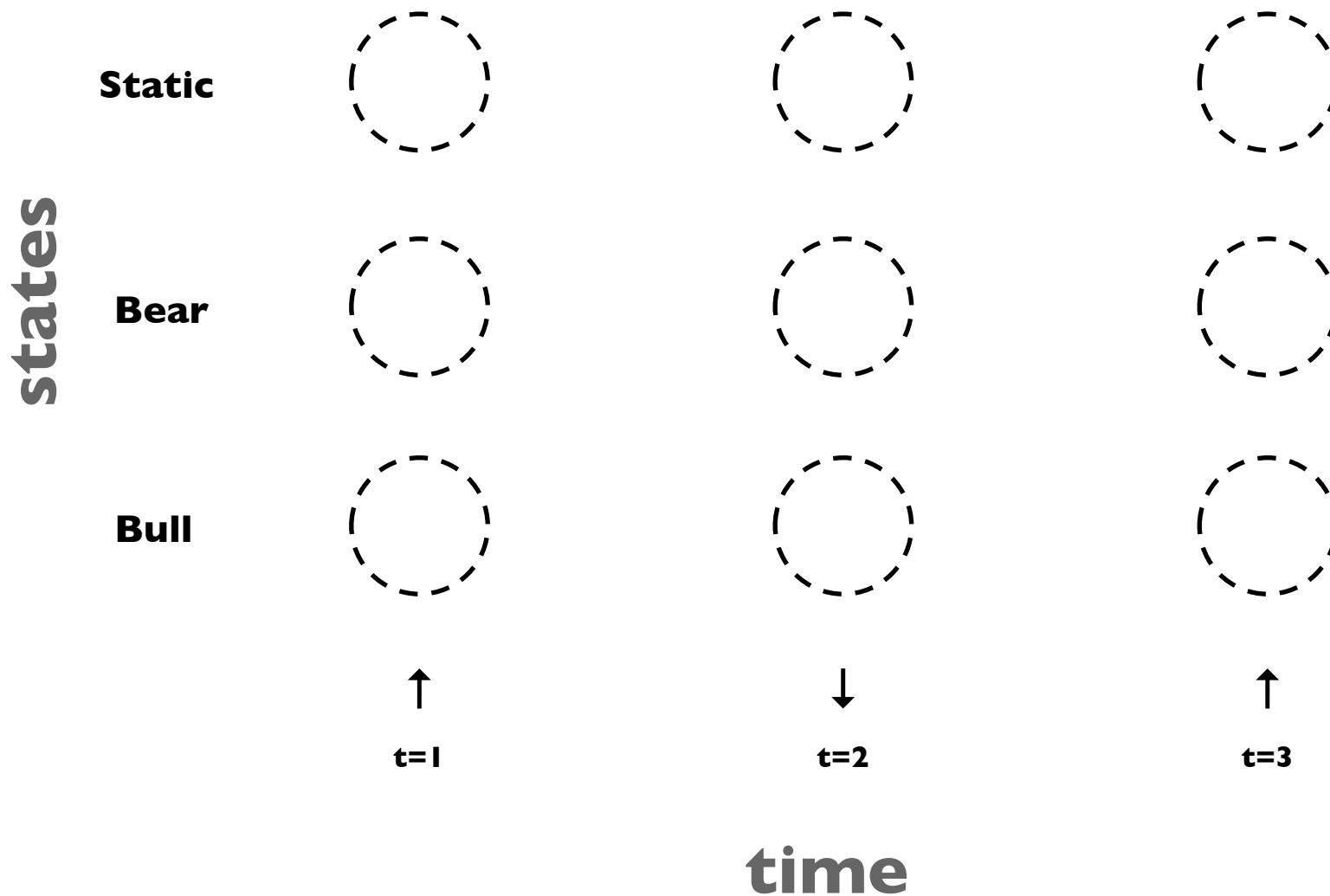


Forward Algorithm

$O = \uparrow \downarrow \uparrow$

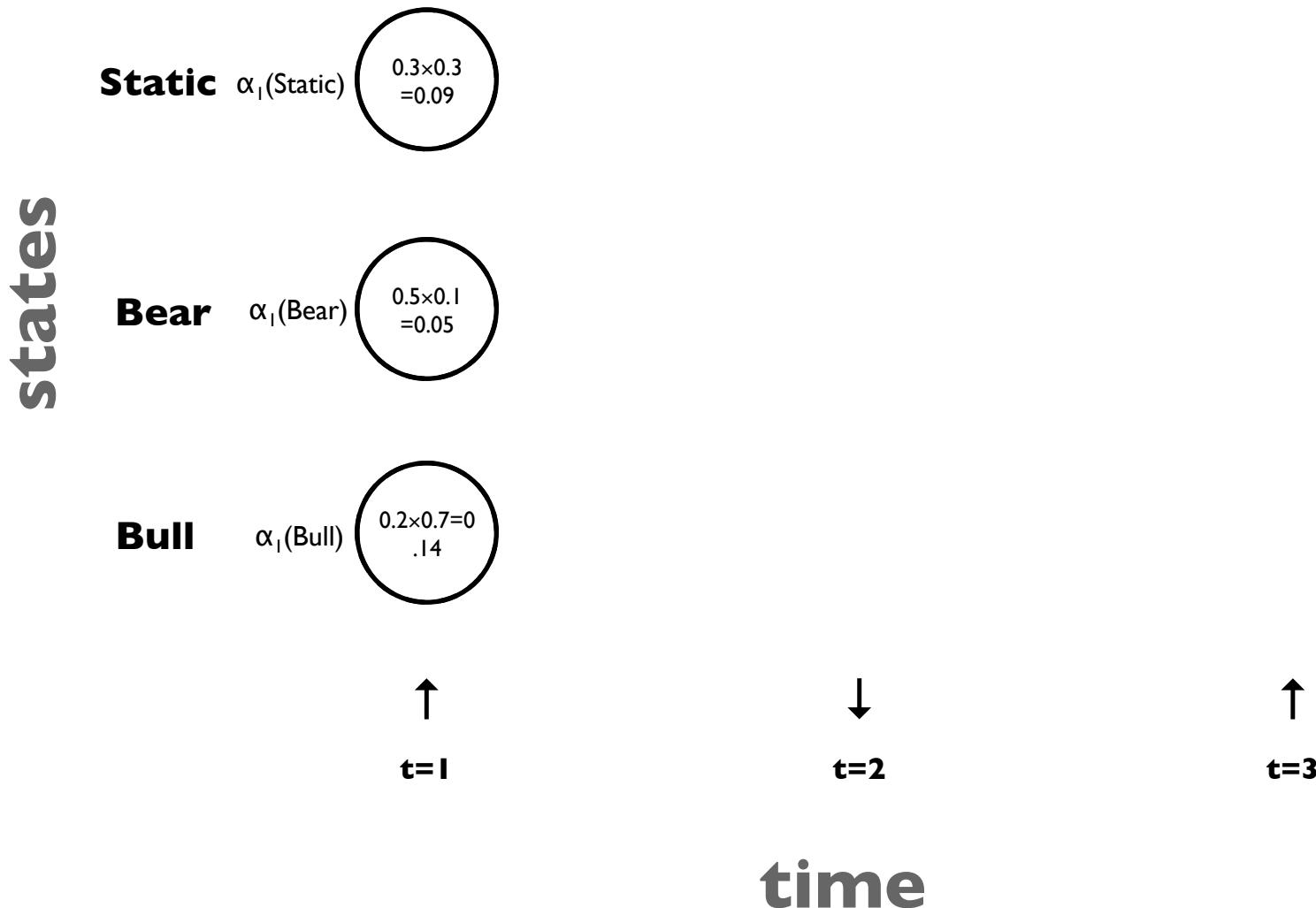
find $P(O|\lambda)$

Forward Algorithm



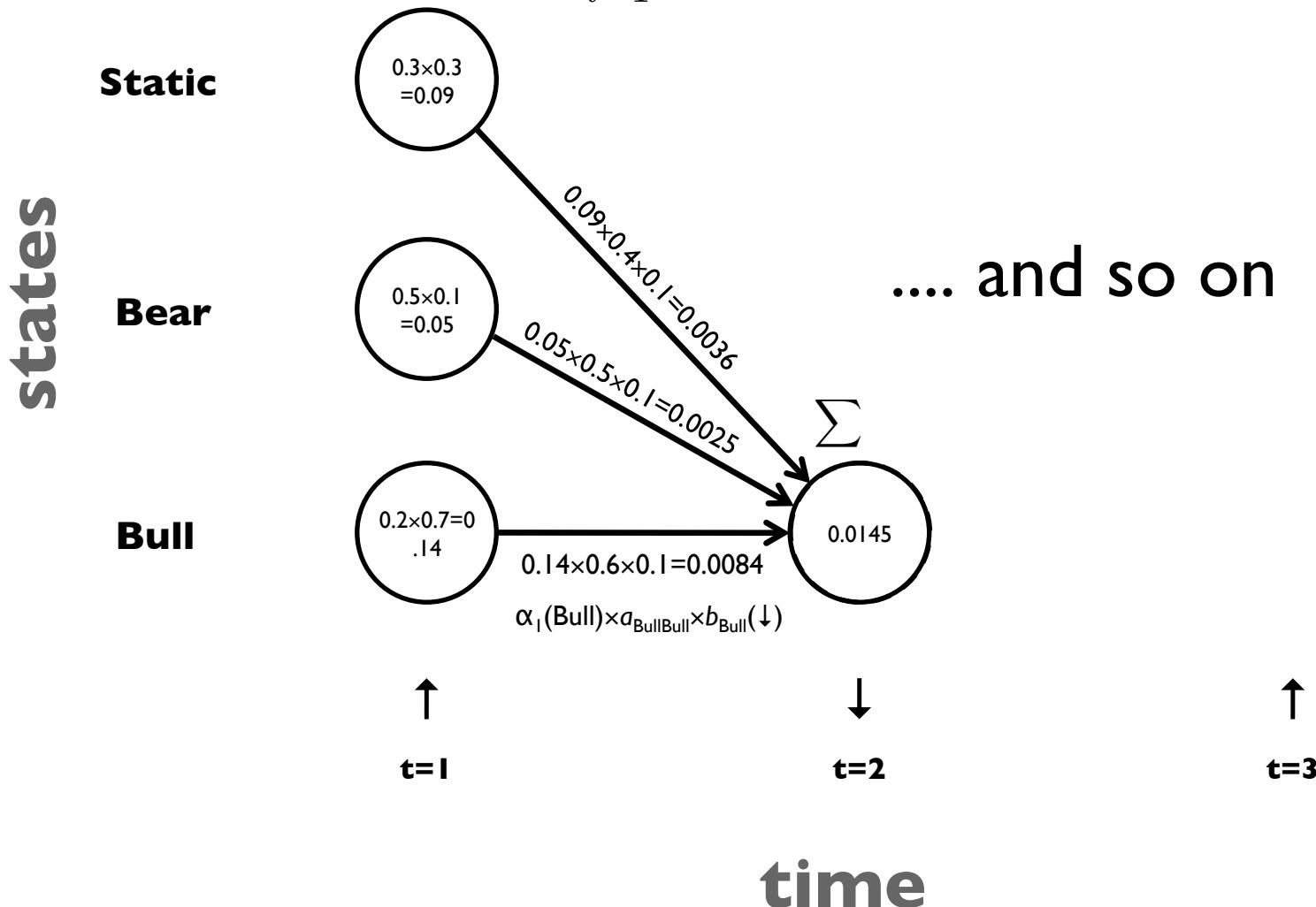
Forward Algorithm: Initialization

$$\alpha_1(j) = \pi_j b_j(o_i) \quad 1 \leq j \leq N$$



Forward Algorithm: Recursion

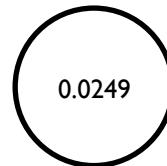
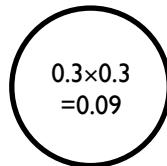
$$\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t) \quad 1 \leq j \leq N, 2 \leq t \leq T$$



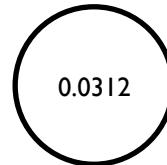
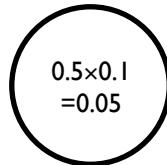
Forward Algorithm: Recursion

states

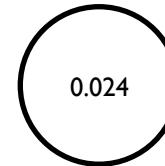
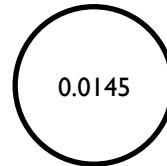
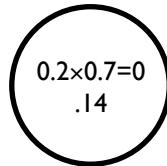
Static



Bear



Bull



t=1



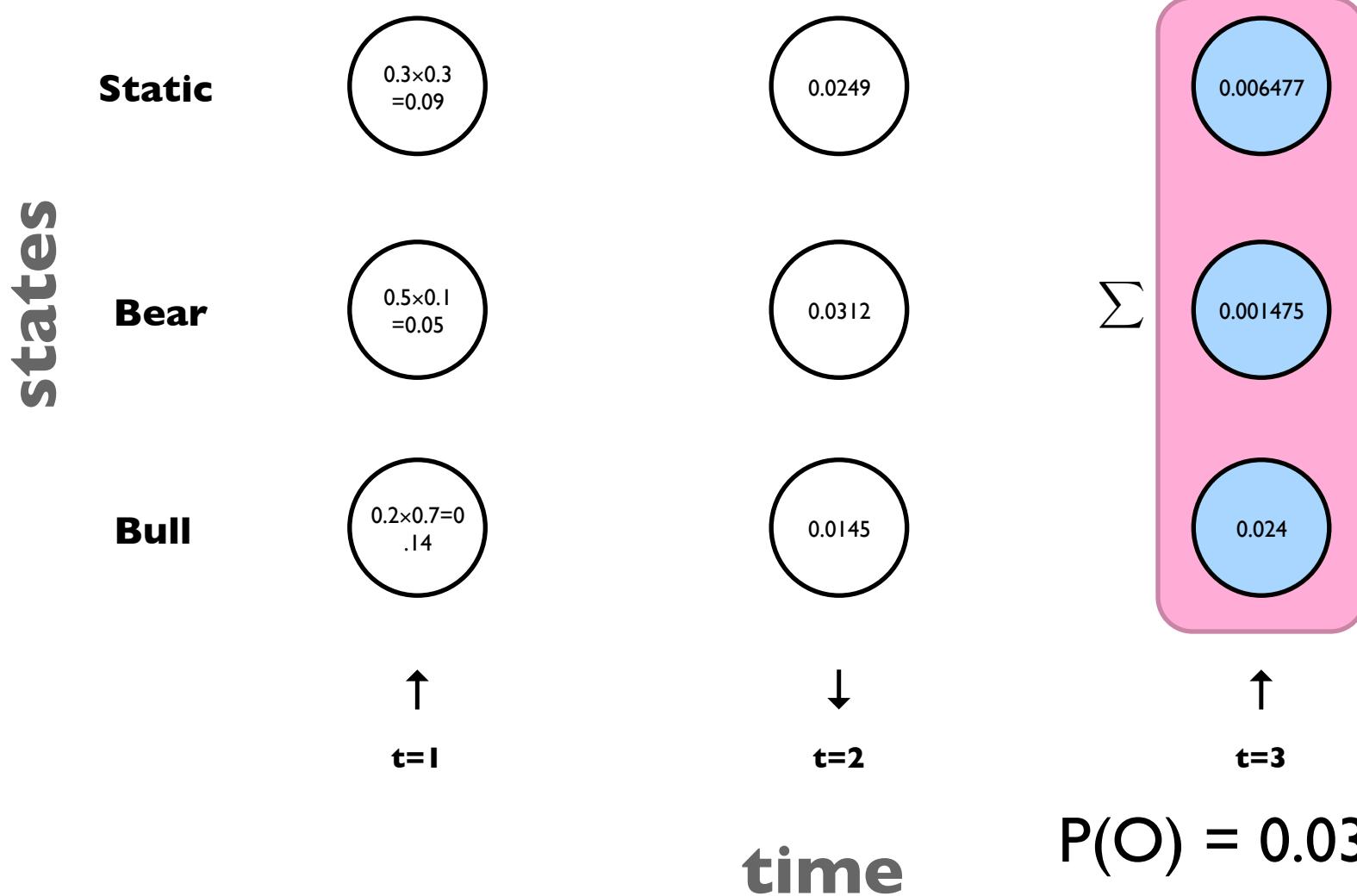
t=2



t=3

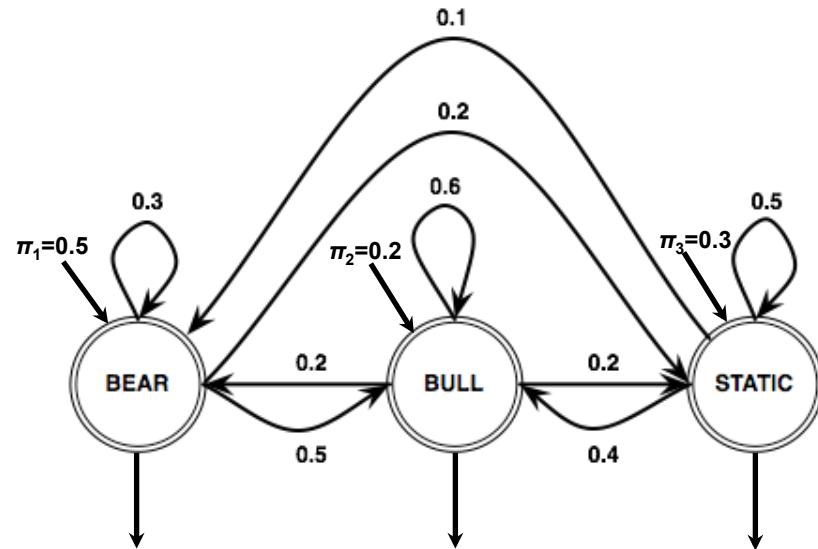
time

Forward Algorithm: Termination



HMM Problem #2: Decoding

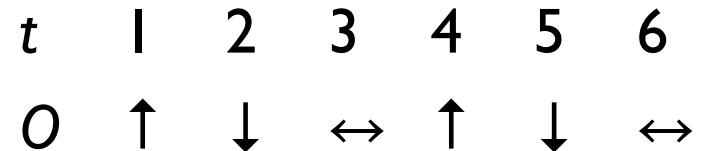
Decoding



$$\begin{bmatrix} P(\uparrow | \text{Bear}) = 0.1 \\ P(\downarrow | \text{Bear}) = 0.6 \\ P(\leftrightarrow | \text{Bear}) = 0.3 \end{bmatrix}$$

$$\begin{bmatrix} P(\uparrow | \text{Bull}) = 0.7 \\ P(\downarrow | \text{Bull}) = 0.1 \\ P(\leftrightarrow | \text{Bull}) = 0.2 \end{bmatrix}$$

$$\begin{bmatrix} P(\uparrow | \text{Static}) = 0.3 \\ P(\downarrow | \text{Static}) = 0.3 \\ P(\leftrightarrow | \text{Static}) = 0.4 \end{bmatrix}$$



Given this model of the stock market, 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
- Same idea as the forward algorithm
 - Store intermediate computation results in a trellis
 - Build new cells from existing cells

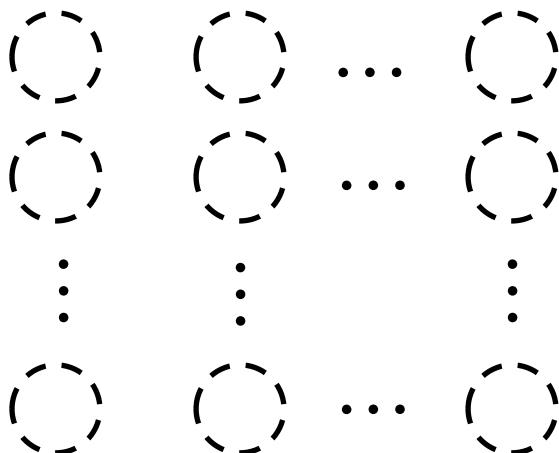
Viterbi Algorithm

- Viterbi probability

- Probability of being in state j after seeing t observations and passing through the most likely state sequence so far

$$v_t(j) = \max_{q_0, q_1, \dots, q_{t-1}} P(q_0, q_1 \dots q_{t-1}, o_1, o_2 \dots o_t, q_t = j | \lambda)$$

- Build an $N \times T$ trellis



- Intuition: forward probability only depends on the previous state and observation

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

$$v_1(j) = \pi_j b_j(o_1) \quad 1 \leq j \leq N$$

$$\text{BT}_1(j) = \emptyset \quad 1 \leq j \leq N$$

- Recursion

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

$$\text{BT}_t(j) = \arg \max_{i=1}^N [v_{t-1}(i)a_{ij}] \quad 1 \leq j \leq N, 2 \leq t \leq T$$

- Termination

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

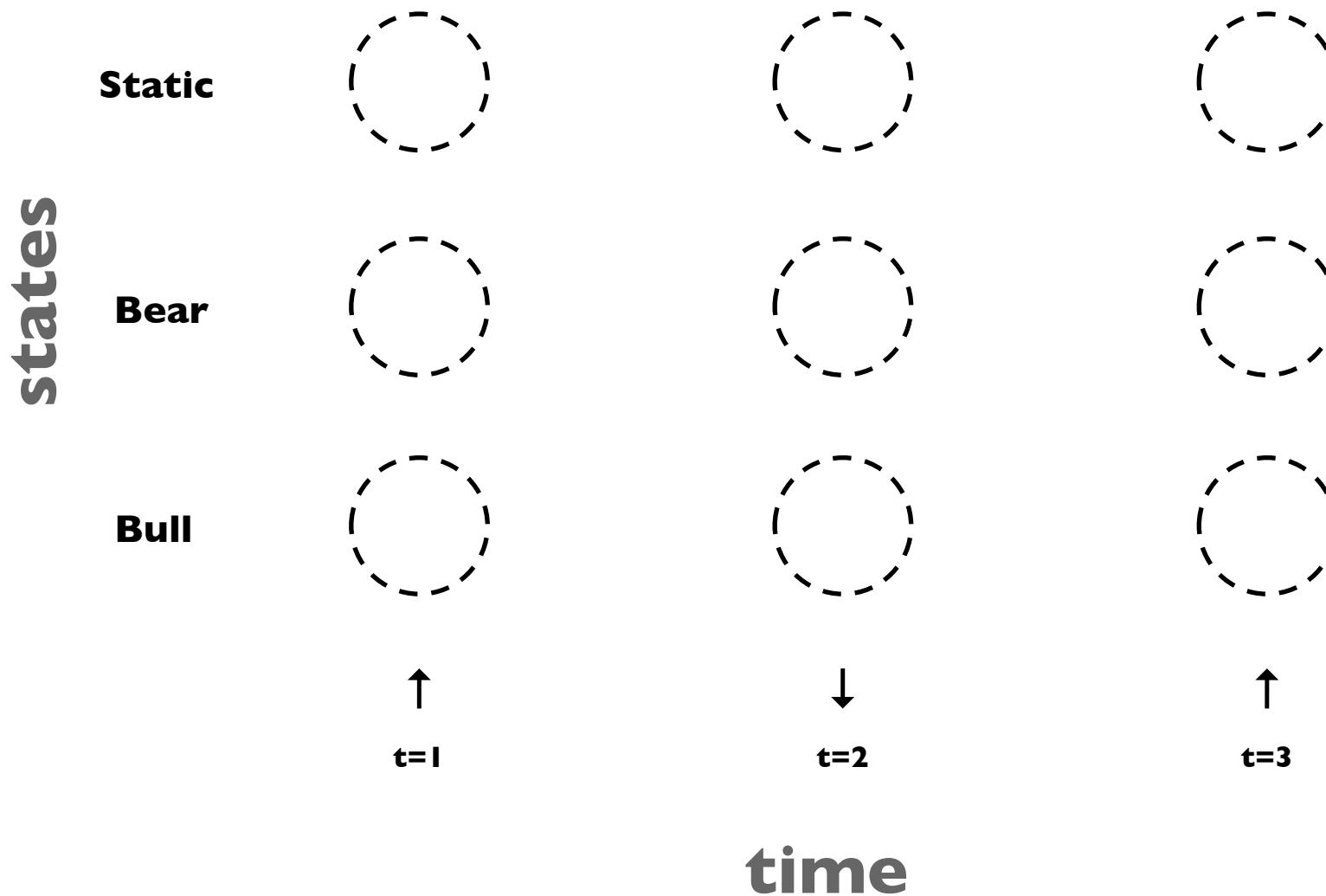
$$q_T^* = \arg \max_{j=1}^N v_T(j)$$

Viterbi Algorithm

$O = \uparrow \downarrow \uparrow$

find most likely state sequence

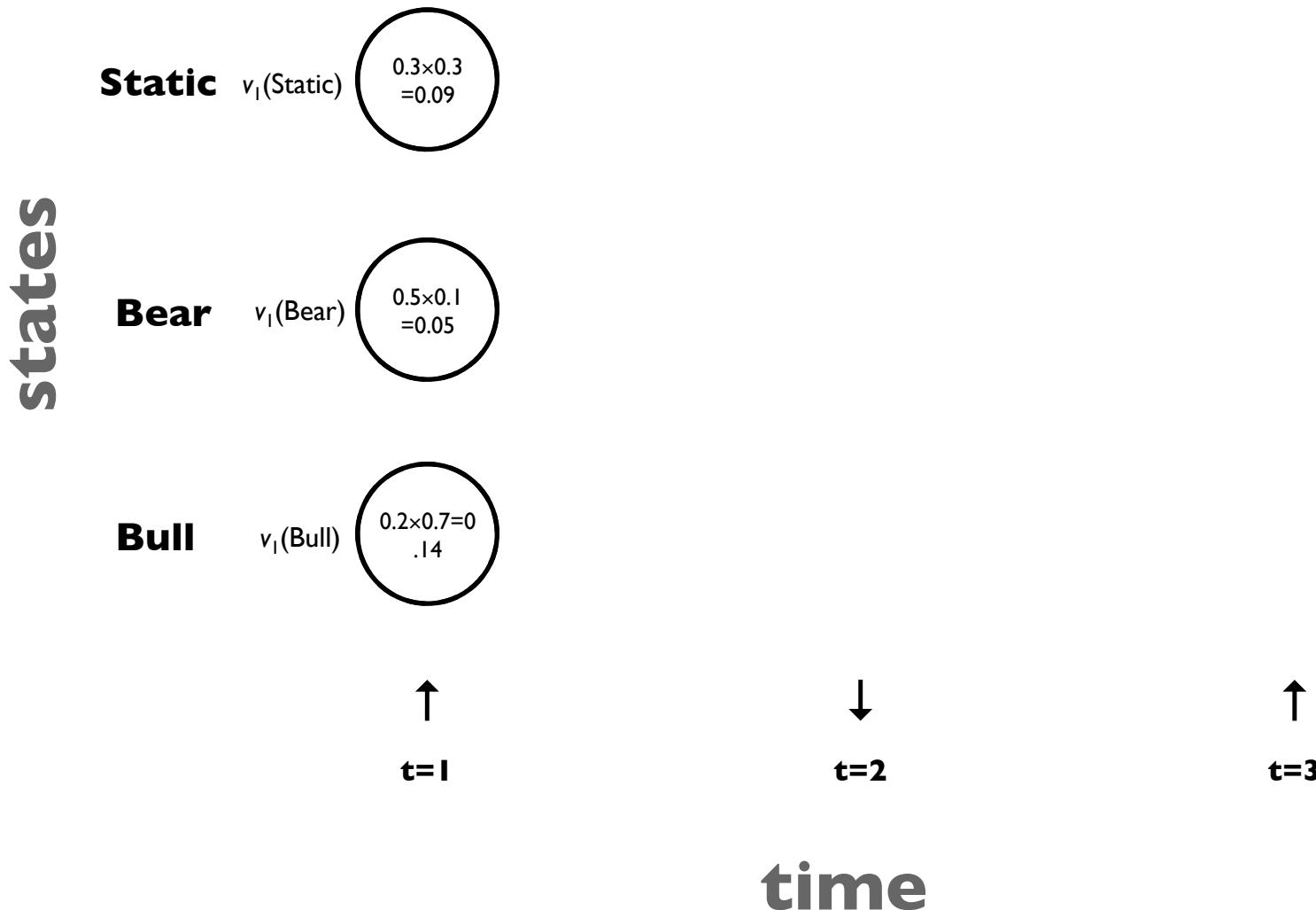
Viterbi Algorithm



Viterbi Algorithm: Initialization

$$v_1(j) = \pi_j b_j(o_1) \quad 1 \leq j \leq N$$

$$BT_1(j) = \emptyset \quad 1 \leq j \leq N$$



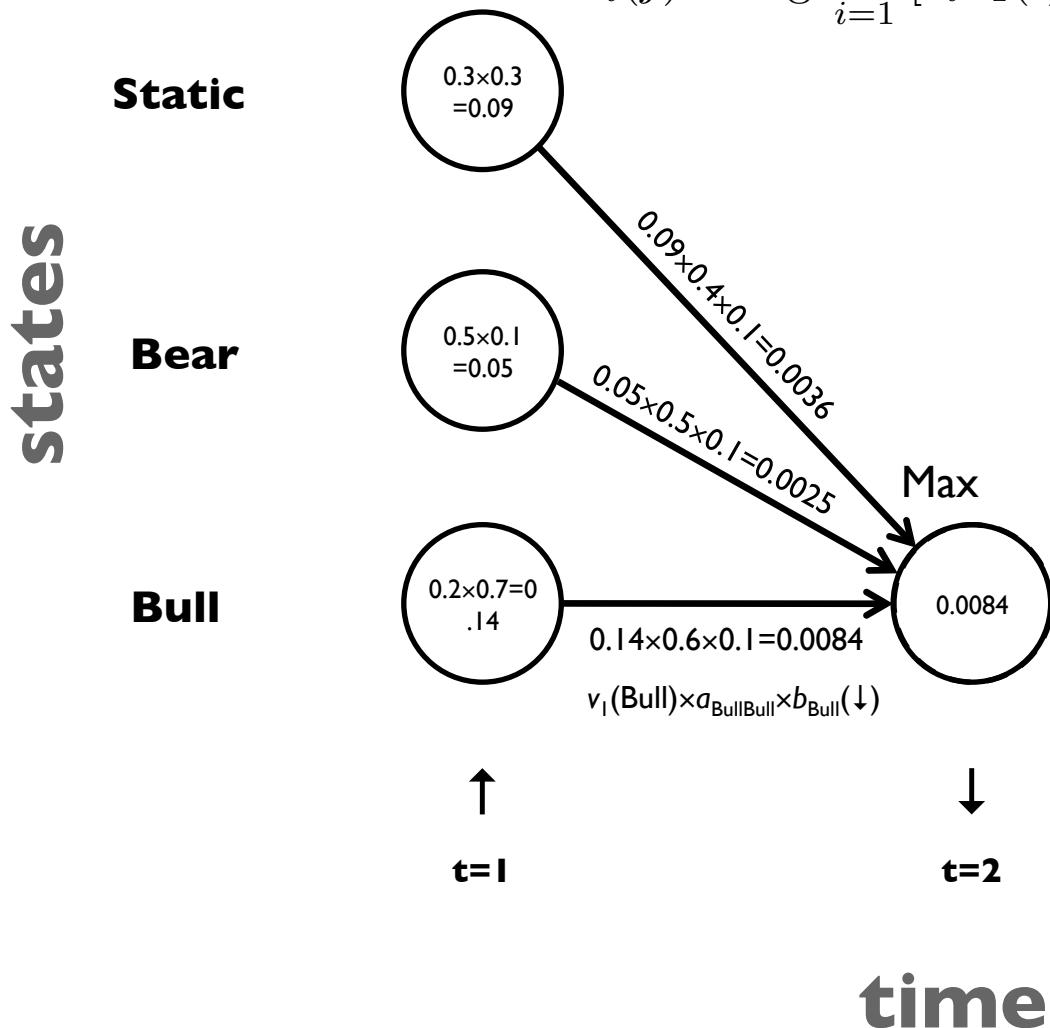
Viterbi Algorithm: Recursion

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

$1 \leq j \leq N, 2 \leq t \leq T$

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

$1 \leq j \leq N, 2 \leq t \leq T$



Viterbi Algorithm: Recursion

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

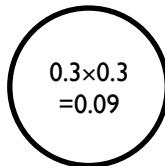
$1 \leq j \leq N, 2 \leq t \leq T$

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

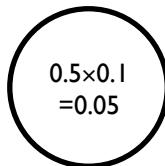
$1 \leq j \leq N, 2 \leq t \leq T$

states

Static



Bear



.... and so on

Bull



$t=1$



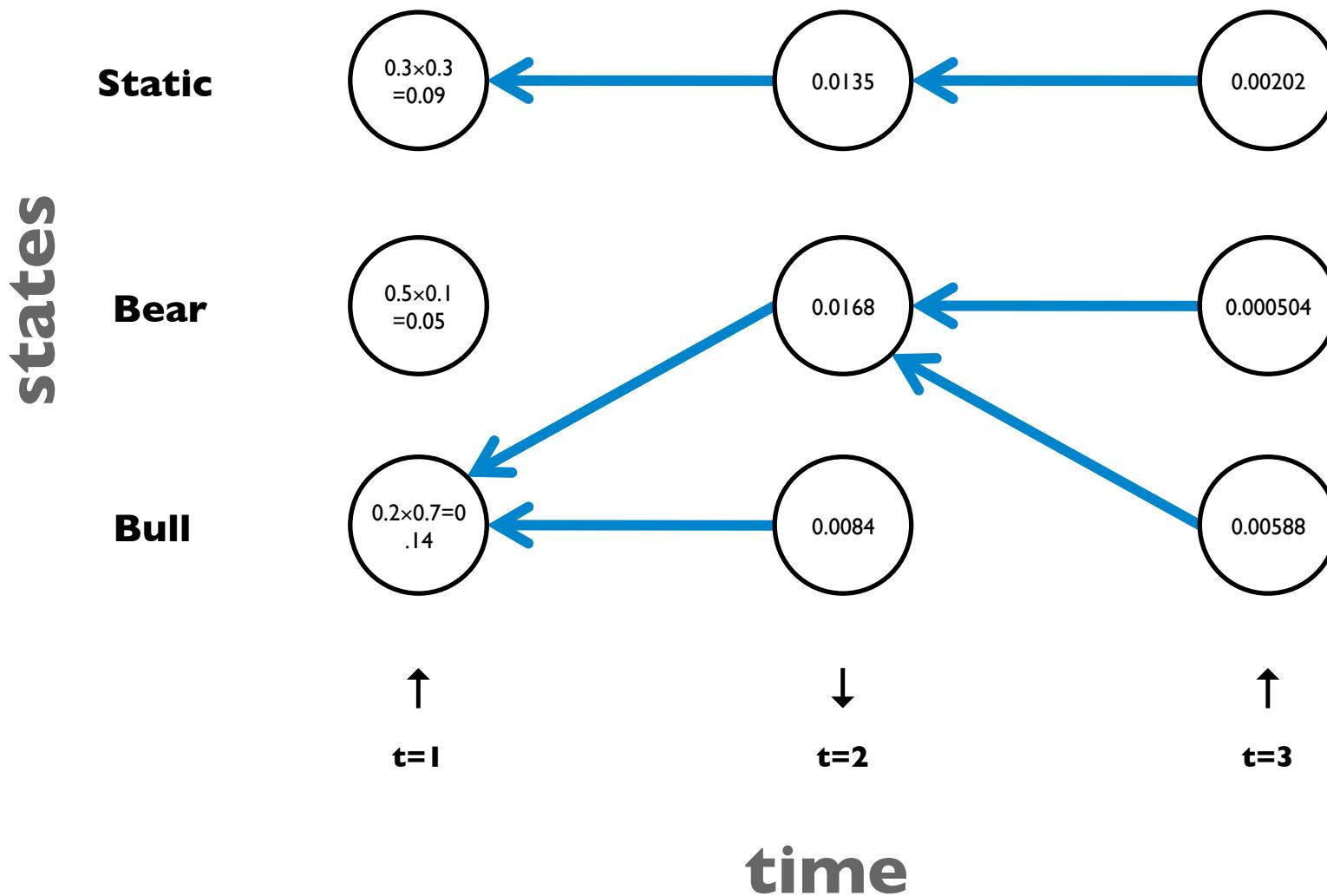
$t=2$



$t=3$

time

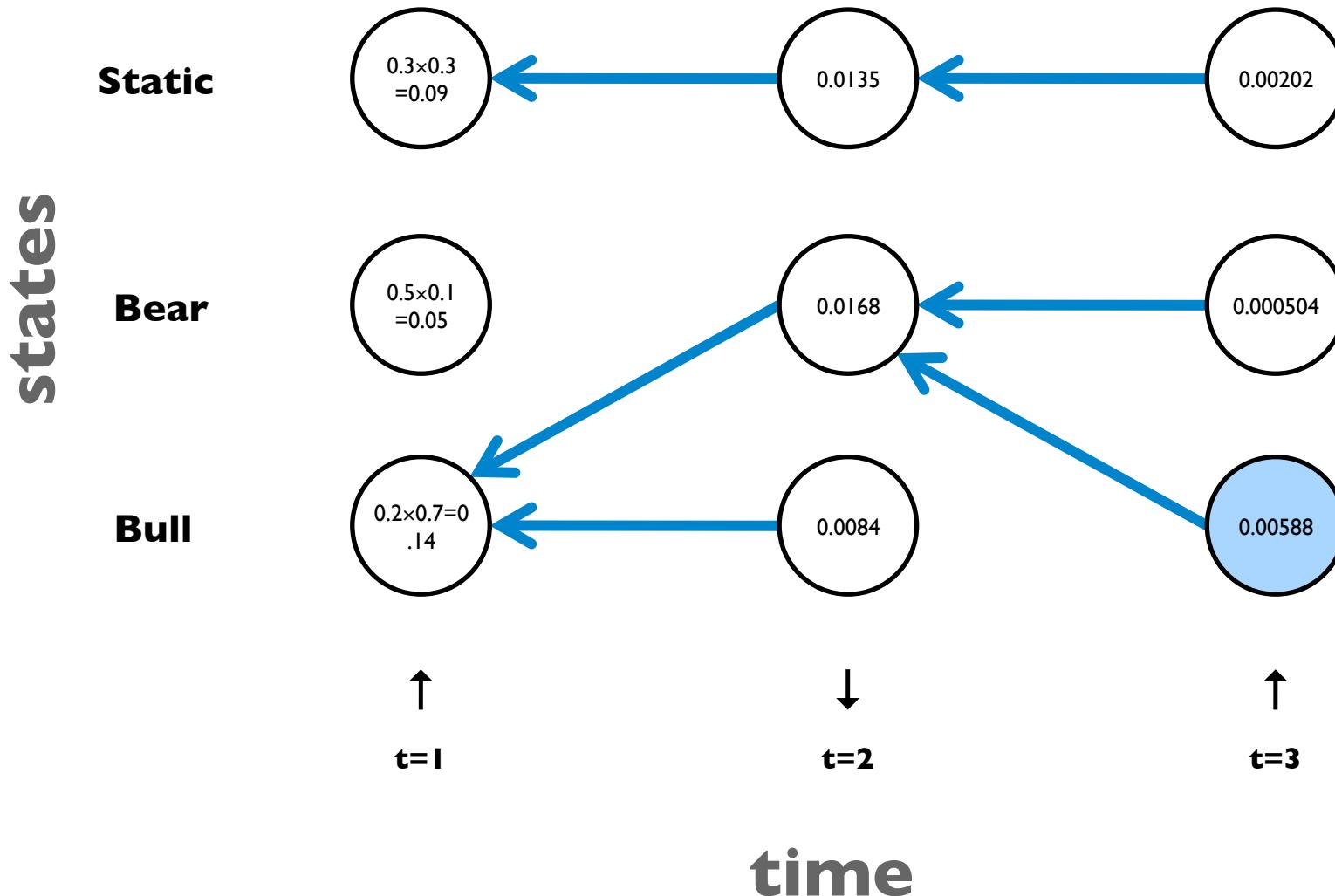
Viterbi Algorithm: Recursion



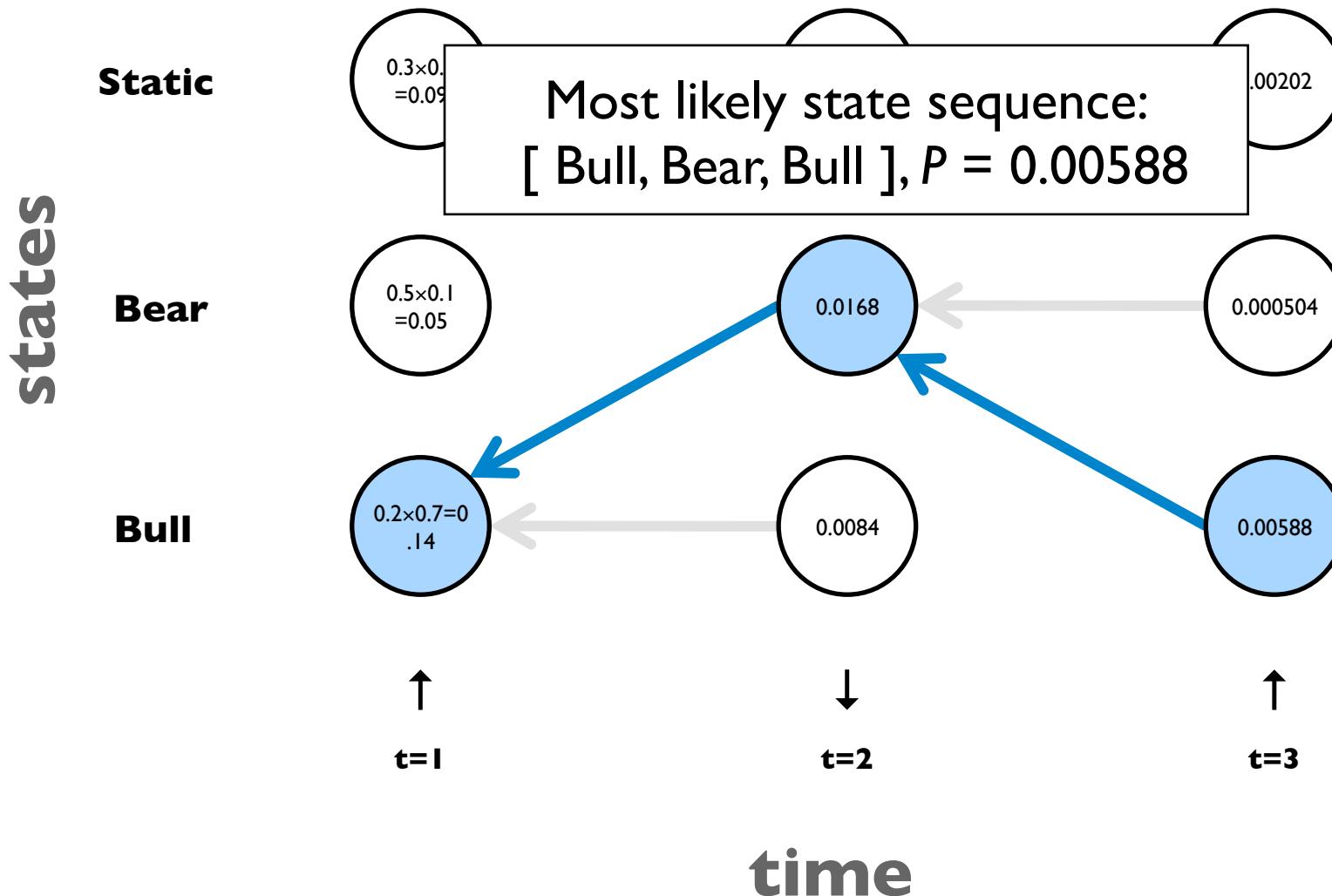
Viterbi Algorithm: Termination

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

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



Viterbi Algorithm: Termination



HMM Problem #3: Learning

The Problem

- Given training data, learn parameters for the HMM
$$\lambda = (A, B, \Pi)$$
 - A : transition probabilities
 - B : emission probabilities
- Two cases:
 - Supervised training: sequences where observations are paired with states
 - Unsupervised training: sequences with observations only

Supervised Training

- Easy! Just like relative frequency counting
- Compute maximum likelihood estimates directly from data
- Transition probabilities:

$$\mathbb{E}[a_{ij}] = \frac{C(i \rightarrow j)}{\sum_{j'} C(i \rightarrow j')}$$

- Emission probabilities:

$$\mathbb{E}[b_j(v)] = \frac{C(j \uparrow v)}{\sum_{v'} C(j \uparrow v')}$$

Supervised Training

- Easy! Just like relative frequency counting
- Remember the usual MapReduce tricks:
 - Pairs vs. stripes for keep track of joint counts
 - Combiners and in-mapper combining apply

Unsupervised Training

- Hard!
- Actually, it's not that bad...
 - Pseudo-counts instead of counts
 - More complex bookkeeping

Unsupervised Training

- What's the fundamental problem?
- Chick-and-egg problem:
 - If we knew what the hidden variables were, then computing the model parameters would be easy
 - If we knew the model parameters, we'd be able to compute the hidden variables (but of course, that's the whole point)

Where have we seen this before?

EM to the Rescue!

- Start with initial parameters
- Iterate until convergence:
 - E-step: Compute posterior distribution over latent (hidden) variables given the model parameters
 - M-step: Update model parameters using posterior distribution computed in the E-step

**How does this work for HMMs?
First, we need one more definition...**

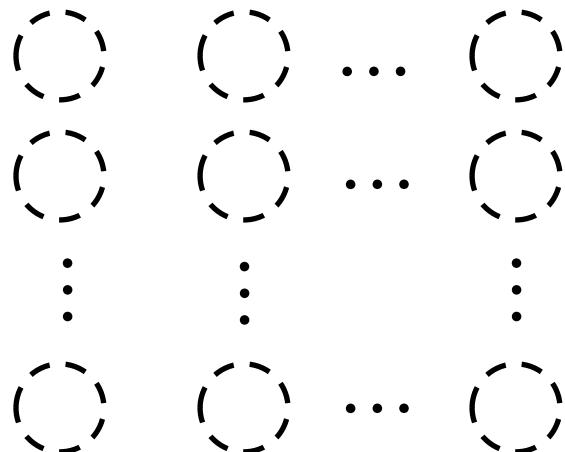
Backward Algorithm

- Backward probability

- Probability of seeing observations from time $t+1$ to T given that we're in state j at time t

$$\beta_t(j) = P(o_{t+1}, o_{t+2} \dots o_T | q_t = i, \lambda)$$

- Guess what? Build an $N \times T$ trellis



Backward Algorithm

$$\beta_t(j) = P(o_{t+1}, o_{t+2} \dots o_T | q_t = j, \lambda)$$

- Initialization

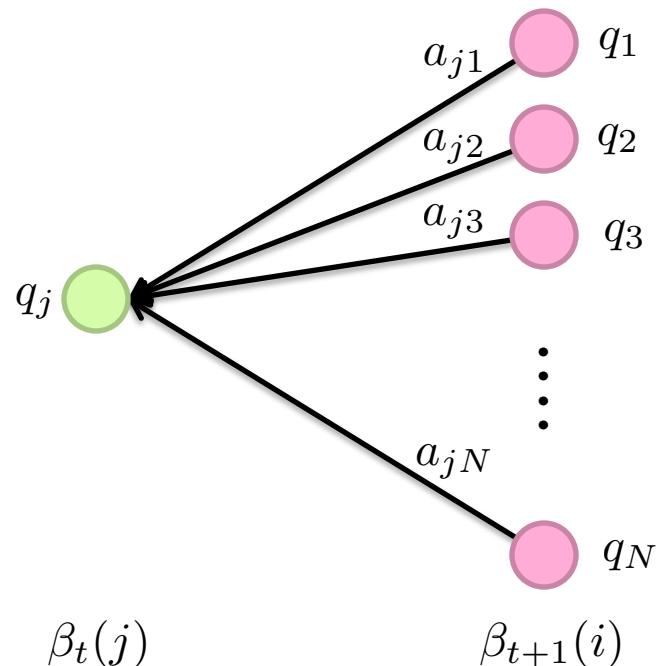
$$\beta_T(j) = 1 \quad 1 \leq j \leq N$$

- Recursion

$$\beta_t(j) = \sum_{i=1}^N a_{ji} b_i(o_{t+1}) \beta_{t+1}(i) \quad 1 \leq j \leq N, 1 \leq t < T$$

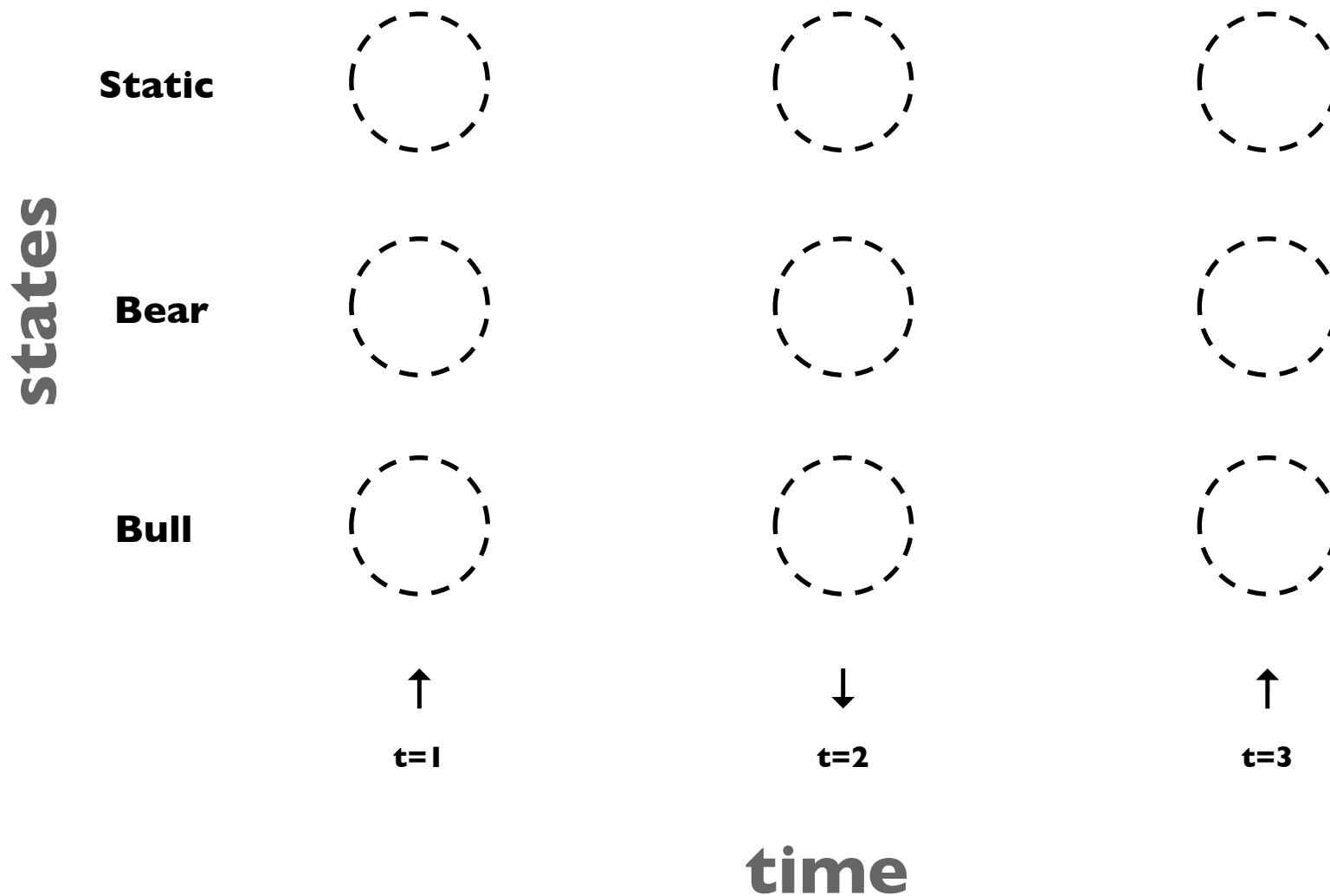
- Termination

$$P(O|\lambda) = \sum_{j=1}^N \pi_j b_j(o_1) \beta_1(j)$$



Backward Algorithm

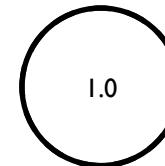
$$\beta_T(j) = 1 \quad 1 \leq j \leq N$$



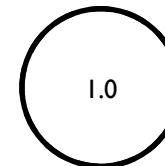
Backward Algorithm: Initialization

states

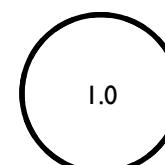
Static



Bear



Bull



↑
t=1

↓
t=2

↑
t=3

time

Backward Algorithm: Recursion

$$\beta_t(j) = \sum_{i=1}^N a_{ji} b_i(o_{t+1}) \beta_{t+1}(i) \quad 1 \leq j \leq N, 1 \leq t < T$$

states

Static

Bear

Bull

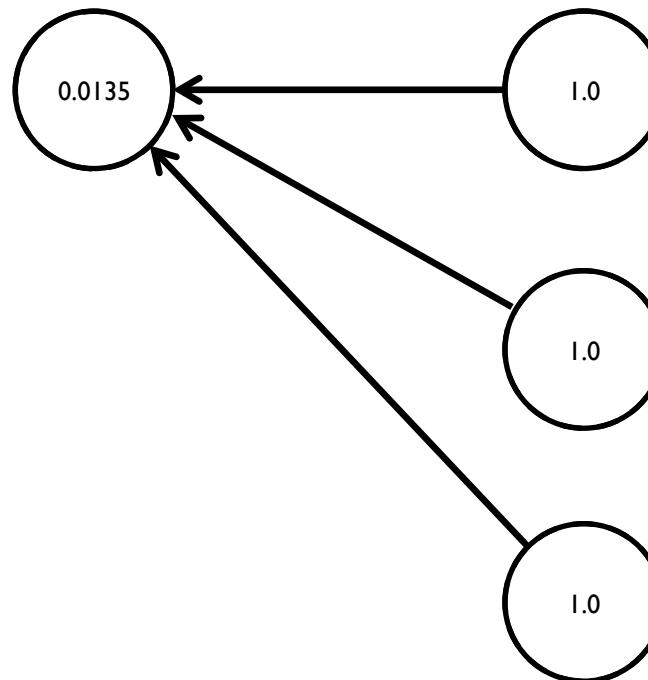
.... and so on

↑
t=1

↓
t=2

↑
t=3

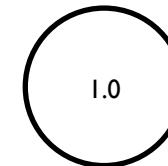
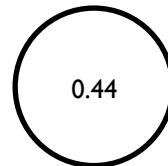
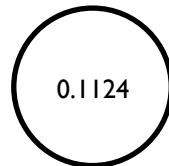
time



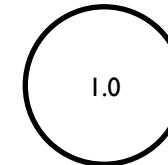
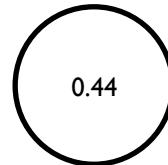
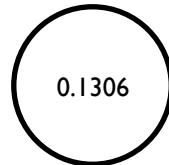
Backward Algorithm: Recursion

states

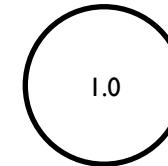
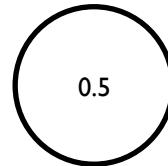
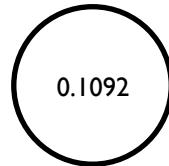
Static



Bear



Bull



t=1



t=2



t=3

time

Forward-Backward

- Forward probability

- Probability of being in state j after t observations

$$\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda)$$

- Backward probability

- Probability of seeing observations from time $t+1$ to T given that we're in state j at time t

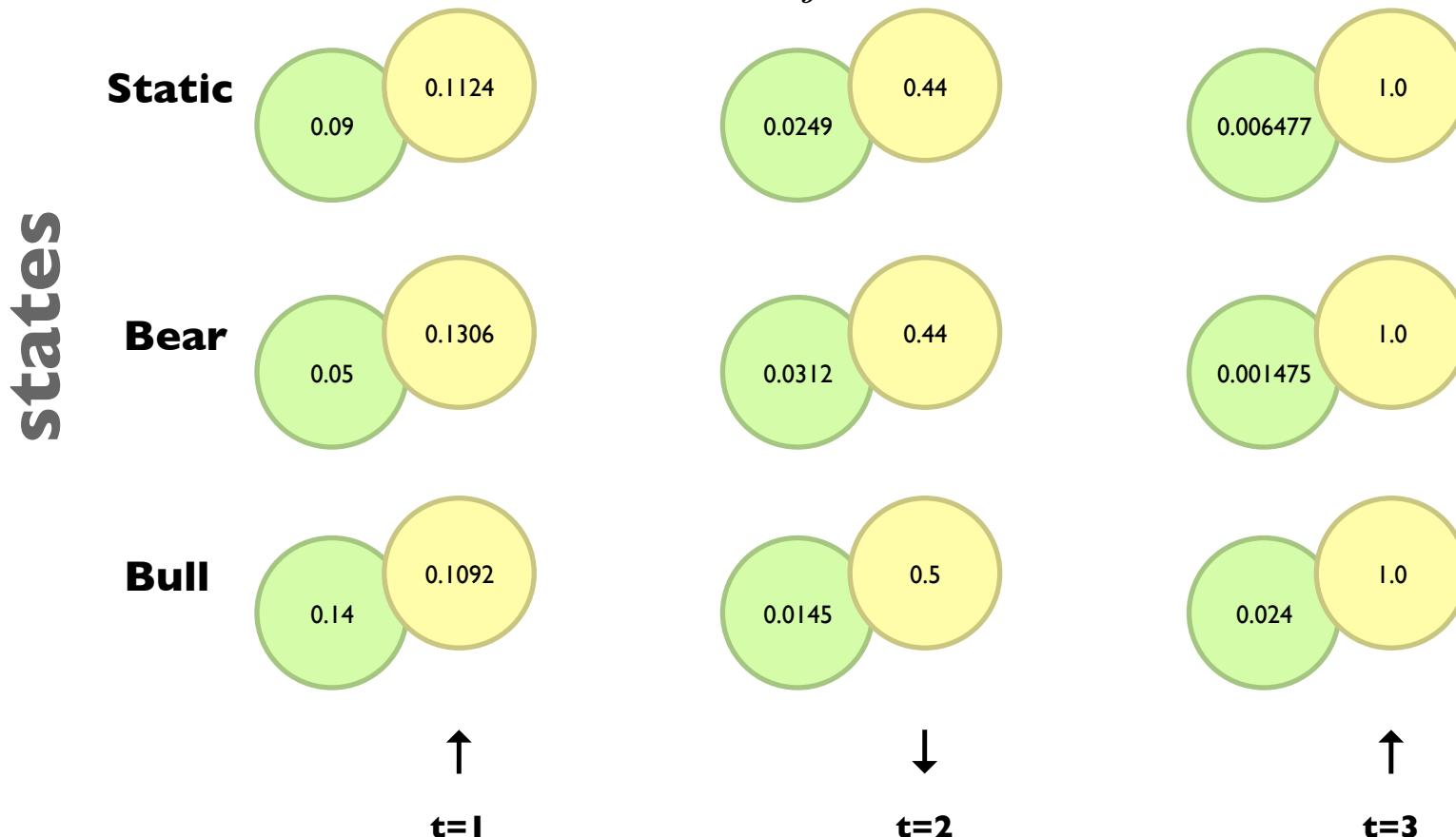
$$\beta_t(j) = P(o_{t+1}, o_{t+2} \dots o_T | q_t = i, \lambda)$$

Neat Observation:

$$P(O|\lambda) = \sum_{j=1}^N \alpha_t(j)\beta_t(j)$$

Forward-Backward

$$P(O|\lambda) = \sum_{j=1}^N \alpha_t(j) \beta_t(j)$$



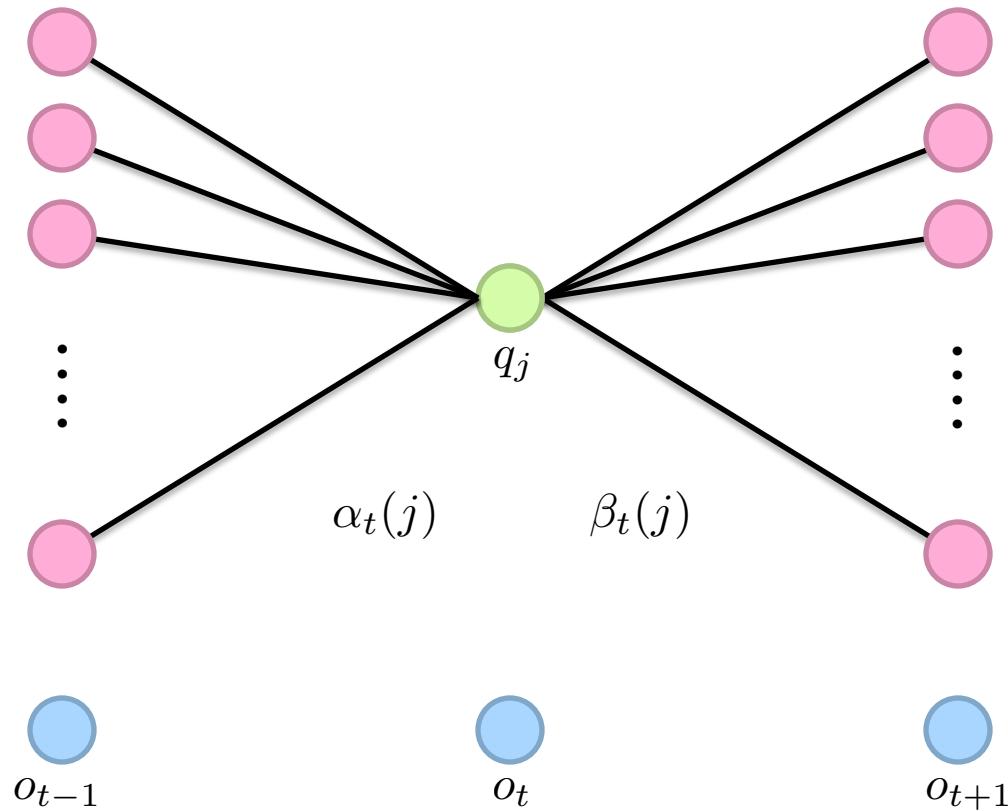
$$P(O|\lambda) = \sum_{j=1}^N \pi_j b_j(o_1) \beta_1(j)$$

time

Forward-Backward

$$\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda)$$

$$\beta_t(j) = P(o_{t+1}, o_{t+2} \dots o_T | q_t = i, \lambda)$$



Estimating Emissions Probabilities

- Basic idea:

$$b_j(v_k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j}$$

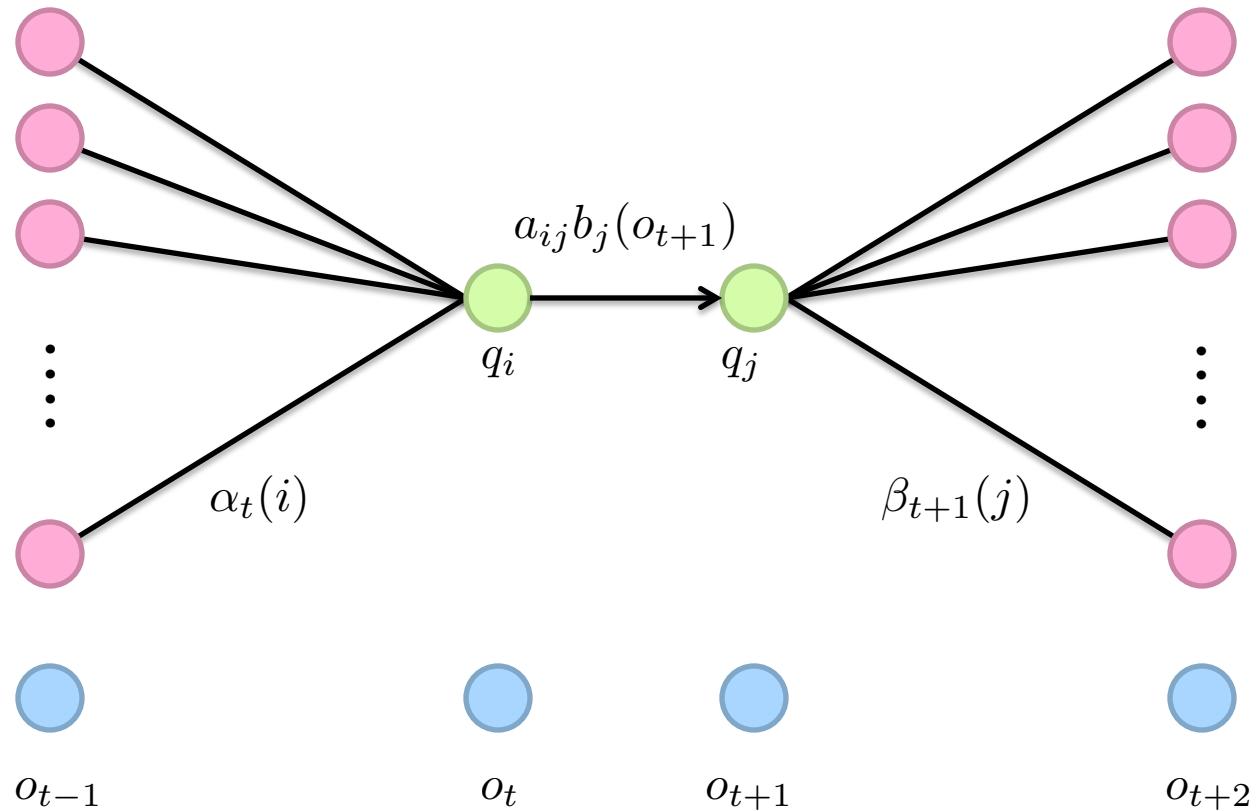
- Let's define:

$$\gamma_t(j) = \frac{P(q_t = j, O | \lambda)}{P(O | \lambda)} = \frac{\alpha_t(j)\beta_t(j)}{P(O | \lambda)}$$

- Thus:

$$\hat{b}_j(v_k) = \frac{\sum_{i=1}^T \gamma_t(j) \text{ where } O_t = v_k}{\sum_{i=1}^T \gamma_t(j)}$$

Forward-Backward



Estimating Transition Probabilities

- Basic idea:

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

- Let's define:

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)}$$

- Thus:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \sum_{j=1}^N \xi_t(i, j)}$$

MapReduce Implementation: Mapper

```

1: class MAPPER
2:   method INITIALIZE(integer iteration)
3:      $\langle \mathcal{S}, \mathcal{O} \rangle \leftarrow \text{READMODEL}$ 
4:      $\theta \leftarrow \langle A, B, \pi \rangle \leftarrow \text{READMODELPARAMS}(iteration)$ 
5:   method MAP(sample id, sequence  $\mathbf{x}$ )
6:      $\alpha \leftarrow \text{FORWARD}(\mathbf{x}, \theta)$ 
7:      $\beta \leftarrow \text{BACKWARD}(\mathbf{x}, \theta)$ 
8:      $I \leftarrow \text{new ASSOCIATIVEARRAY}$ 
9:     for all  $q \in \mathcal{S}$  do
10:        $I\{q\} \leftarrow \alpha_1(q) \cdot \beta_1(q)$ 
11:      $O \leftarrow \text{new ASSOCIATIVEARRAY of ASSOCIATIVEARRAY}$ 
12:     for  $t = 1$  to  $|\mathbf{x}|$  do
13:       for all  $q \in \mathcal{S}$  do
14:          $O\{q\}\{x_t\} \leftarrow O\{q\}\{x_t\} + \alpha_t(q) \cdot \beta_t(q)$ 
15:        $t \leftarrow t + 1$ 
16:      $T \leftarrow \text{new ASSOCIATIVEARRAY of ASSOCIATIVEARRAY}$ 
17:     for  $t = 1$  to  $|\mathbf{x}| - 1$  do
18:       for all  $q \in \mathcal{S}$  do
19:         for all  $r \in \mathcal{S}$  do
20:            $T\{q\}\{r\} \leftarrow T\{q\}\{r\} + \alpha_t(q) \cdot A_q(r) \cdot B_r(x_{t+1}) \cdot \beta_{t+1}(r)$ 
21:        $t \leftarrow t + 1$ 
22:     EMIT(string 'initial', stripe  $I$ )
23:     for all  $q \in \mathcal{S}$  do
24:       EMIT(string 'emit from ' +  $q$ , stripe  $O\{q\}$ )
25:       EMIT(string 'transit from ' +  $q$ , stripe  $T\{q\}$ )

```

$$\hat{b}_j(v_k) = \frac{\sum_{i=1}^T \gamma_t(j)}{\sum_{i=1}^T \gamma_t(i)}$$

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \sum_{j=1}^N \xi_t(i, j)}$$

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)}$$

$$\xi_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{P(O|\lambda)}$$

MapReduce Implementation: Reducer

```

1: class COMBINER
2:   method COMBINE(string  $t$ , stripes [ $C_1, C_2, \dots$ ])
3:      $C_f \leftarrow$  new ASSOCIATIVEARRAY
4:     for all stripe  $C \in$  stripes [ $C_1, C_2, \dots$ ] do
5:       SUM( $C_f, C$ )
6:     EMIT(string  $t$ , stripe  $C_f$ )
7:
8: class REDUCER
9:   method REDUCE(string  $t$ , stripes [ $C_1, C_2, \dots$ ])
10:     $C_f \leftarrow$  new ASSOCIATIVEARRAY
11:    for all stripe  $C \in$  stripes [ $C_1, C_2, \dots$ ] do
12:      SUM( $C_f, C$ )
13:     $z \leftarrow 0$ 
14:    for all  $\langle k, v \rangle \in C_f$  do
15:       $z \leftarrow z + v$ 
16:     $P_f \leftarrow$  new ASSOCIATIVEARRAY
17:    for all  $\langle k, v \rangle \in C_f$  do
18:       $P_f\{k\} \leftarrow v/z$ 
19:    EMIT(string  $t$ , stripe  $P_f$ )

```

$$\hat{b}_j(v_k) = \frac{\sum_{i=1}^T \gamma_t(j)}{\sum_{i=1}^T \gamma_t(i)}$$

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \sum_{j=1}^N \xi_t(i, j)}$$

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)}$$

$$\xi_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{P(O|\lambda)}$$

Implementation Notes

- Standard setup of iterative MapReduce algorithms
 - Driver program sets up MapReduce job
 - Waits for completion
 - Checks for convergence
 - Repeats if necessary
- Additional details:
 - Must load and serialize model parameters at each iteration
 - Reducer receives $2N + 1$ keys, so limits to reducer parallelization
 - Iteration overhead less than other iterative algorithms

A photograph of a traditional Japanese rock garden. In the foreground, a gravel path is raked into fine, parallel lines. Several large, dark, irregular stones are scattered across the garden. A small, shallow pond is visible in the middle ground, surrounded by more stones and low, rounded green shrubs. In the background, there are larger trees with autumn-colored leaves and traditional wooden buildings with tiled roofs.

Questions?

Midterm

Final Projects

- Anything in the big data space
- Teams of 2-3 people
- Available data resources
- Available cluster resources
- Project Ideas?