Data-Intensive Computing with MapReduce

Session 8: Sequence Labeling

Jimmy Lin
University of Maryland
Thursday, March 14, 2013

Source: Wikipedia (The Scream)

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:         ⟨S, O⟩ ← ReadModel
4:         θ ← ⟨A, B, π⟩ ← ReadModelParams(iteration)
5:     method Map(sample id, sequence x)
6:         α ← Forward(x, θ)
7:         β ← Backward(x, θ)
8:         I ← new AssociativeArray
9:         for all q ∈ S do
10:            I{q} ← α_1(q) · β_1(q)
11:        O ← new AssociativeArray of AssociativeArray
12:        for t = 1 to |x| do
13:            for all q ∈ S do
14:                O{q}{x_t} ← O{q}{x_t} + α_t(q) · β_t(q)
15:            t ← t + 1
16:        T ← new AssociativeArray of AssociativeArray
17:        for t = 1 to |x| − 1 do
18:            for all q ∈ S do
19:                for all r ∈ S do
20:                    T{q}{r} ← T{q}{r} + α_t(q) · A_q(r) · B_r(x_{t+1}) · β_{t+1}(r)
21:            t ← t + 1
22:        Emit(string 'initial', stripe I)
23:        for all q ∈ S do
24:            Emit(string 'emit from ' + q, stripe O{q})
25:            Emit(string 'transit from ' + q, stripe T{q})
```

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

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

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

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

MapReduce Implementation: Reducer

```
1: class Combiner
2:     method Combine(string t, stripes [C_1, C_2, ...])
3:         C_f ← new AssociativeArray
4:         for all stripe C ∈ stripes [C_1, C_2, ...] do
5:             Sum(C_f, C)
6:         Emit(string t, stripe C_f)
```

```
1: class Reducer
2:     method Reduce(string t, stripes [C_1, C_2, ...])
3:         C_f ← new AssociativeArray
4:         for all stripe C ∈ stripes [C_1, C_2, ...] do
5:             Sum(C_f, C)
6:         z ← 0
7:         for all ⟨k, v⟩ ∈ C_f do
8:             z ← z + v
9:         P_f ← new AssociativeArray
10:        for all ⟨k, v⟩ ∈ C_f do
11:            P_f{k} ← v/z
12:        Emit(string t, stripe P_f)
```

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

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

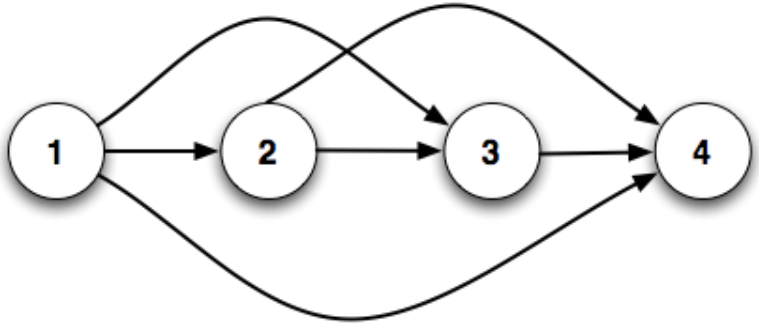
$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)}$$

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

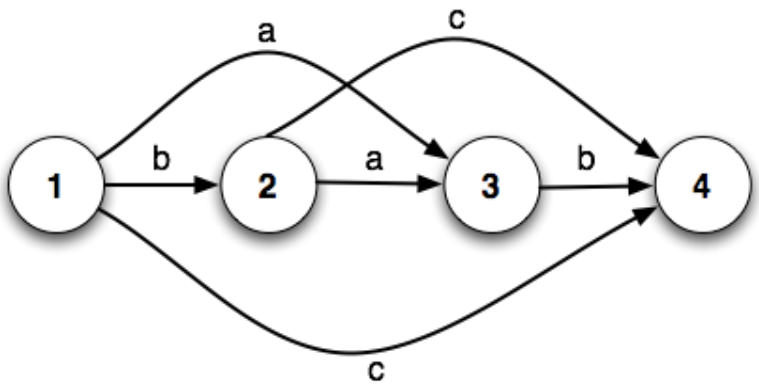
Finite State Machines

- Q: a finite set of N states
  - $Q = \{q_0, q_1, q_2, q_3, \ldots\}$
  - 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'

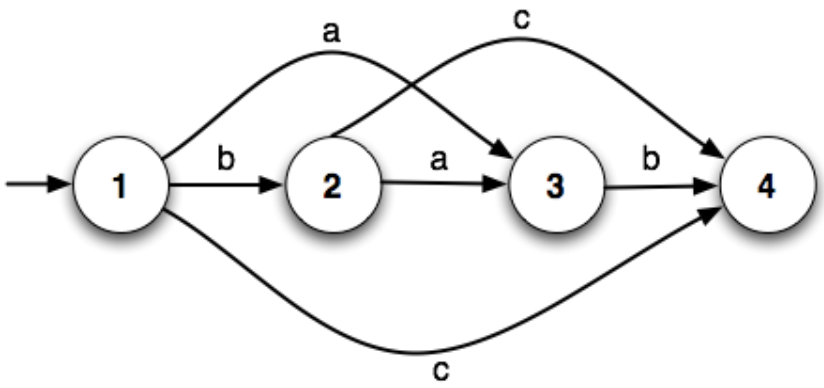
Finite number of states



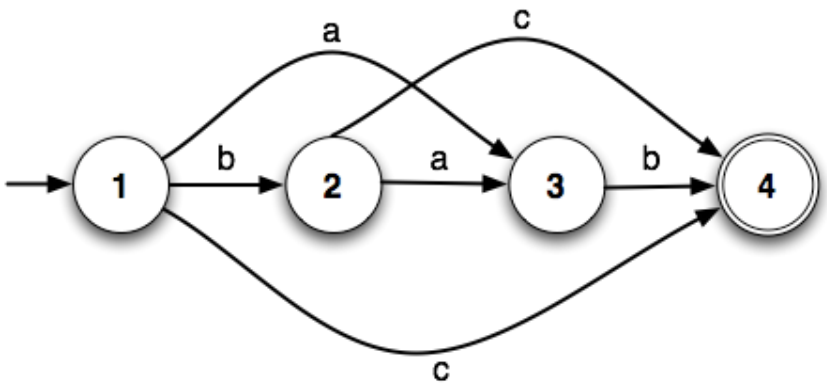
Transitions



Input alphabet



Start state



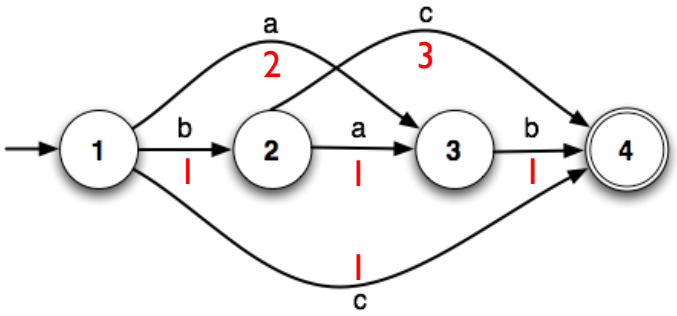
Final state(s)

What can we do with FSMs?

- Generate valid sequences
- Accept valid sequences

Weighted FSMs

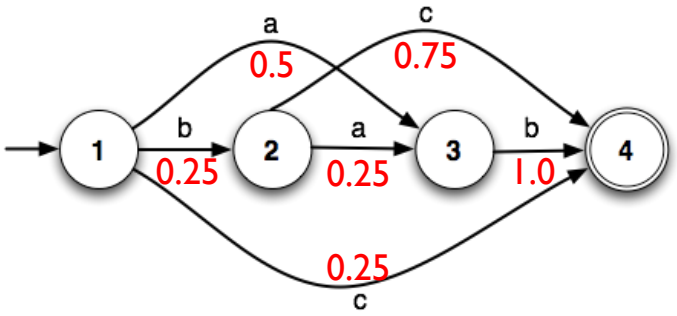
- FSMs treat all transitions as equally likely
- What if we know more about state transitions?
  - ‘a’ is twice as likely to be seen in state 1 as ‘b’ or ‘c’
  - ‘c’ is three times as likely to be seen in state 2 as ‘a’



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

Probabilistic FSMs = Markov Chains

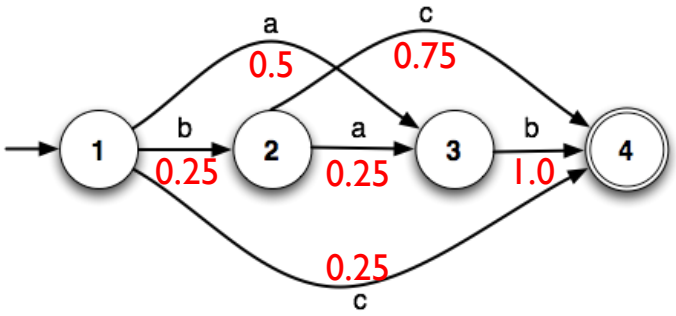
- Let's replace weights with probabilities
- What if we know more about state transitions?
  - 'a' is twice as likely to be seen in state 1 as 'b' or 'c'
  - 'c' is three times as likely to be seen in state 2 as 'a'



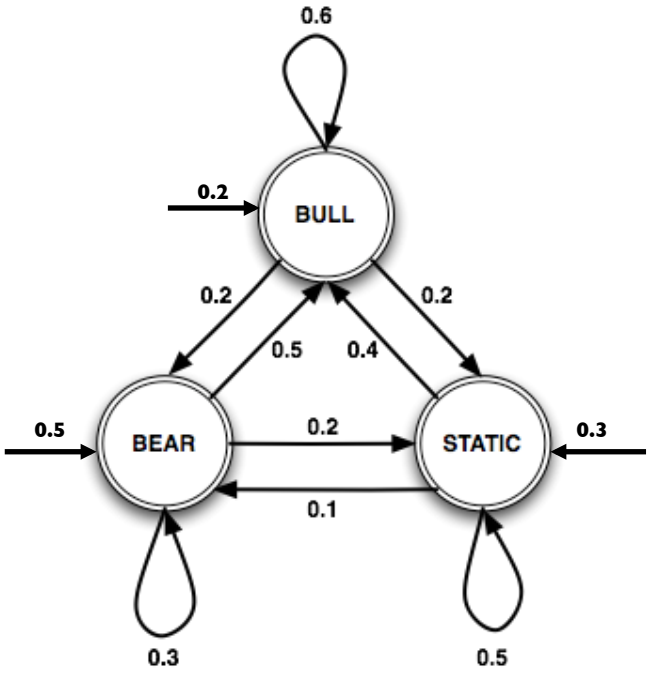
- What do we get out of it?
  - P('ab') = 0.5
  - P('bc') = 0.1875

Specifying Markov Chains

- Q: a finite set of N states
  - $Q = \{q_0, q_1, q_2, q_3, \ldots\}$
- 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

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

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

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

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

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

- Prior probabilities vector $\Pi = [\pi_i, \pi_2, \ldots \pi_N]$

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

Stock Market HMM

States? ✓

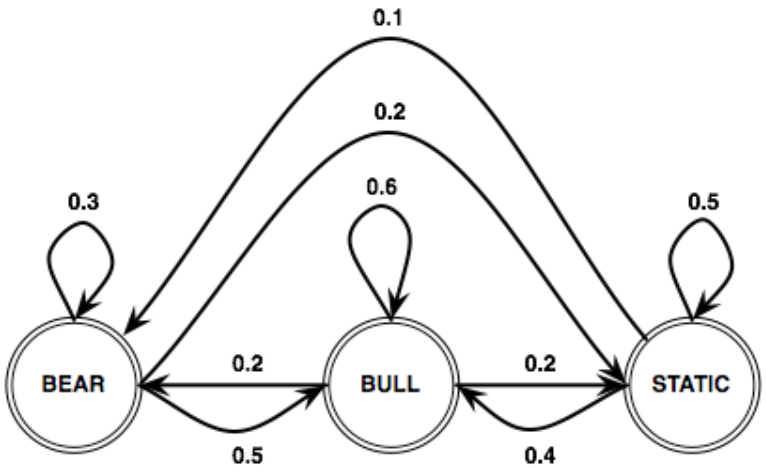
Transitions?

Vocabulary?

Emissions?

Priors?

Stock Market HMM



States? ✓

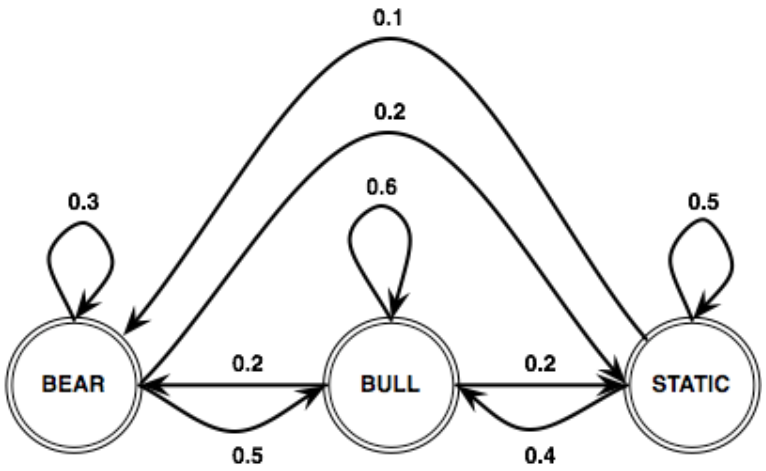
Transitions? ✓

Vocabulary?

Emissions?

Priors?

Stock Market HMM



States? ✓

Transitions? ✓

Vocabulary? ✓

Emissions?

Priors?

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

Stock Market HMM

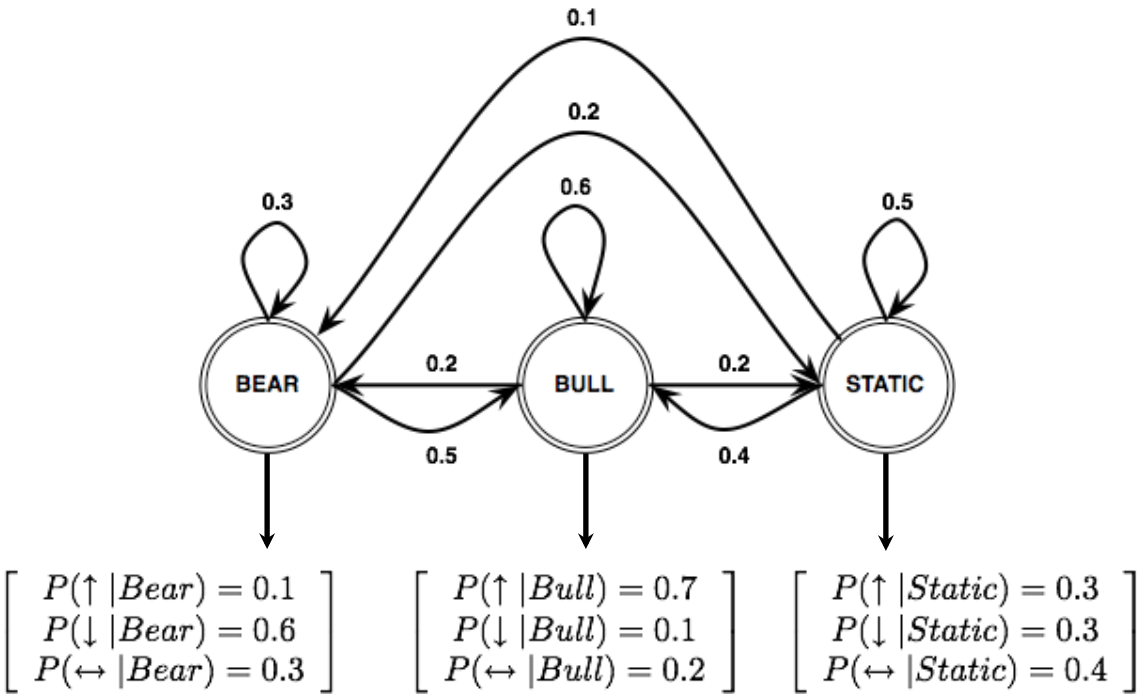
States? ✓

Transitions? ✓

Vocabulary? ✓

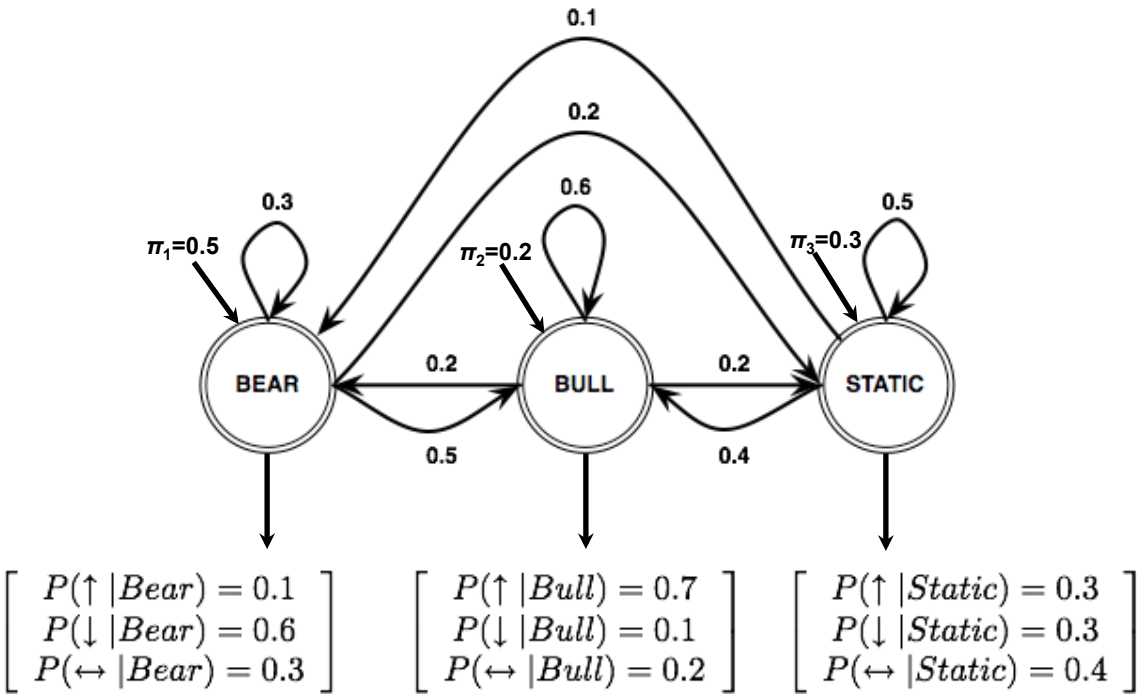
Emissions? ✓

Priors?



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

Stock Market HMM



States? ✓

Transitions? ✓

Vocabulary? ✓

Emissions? ✓

Priors? ✓

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

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 $\Pi = [\pi_i, \pi_2, \ldots \pi_N]$
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

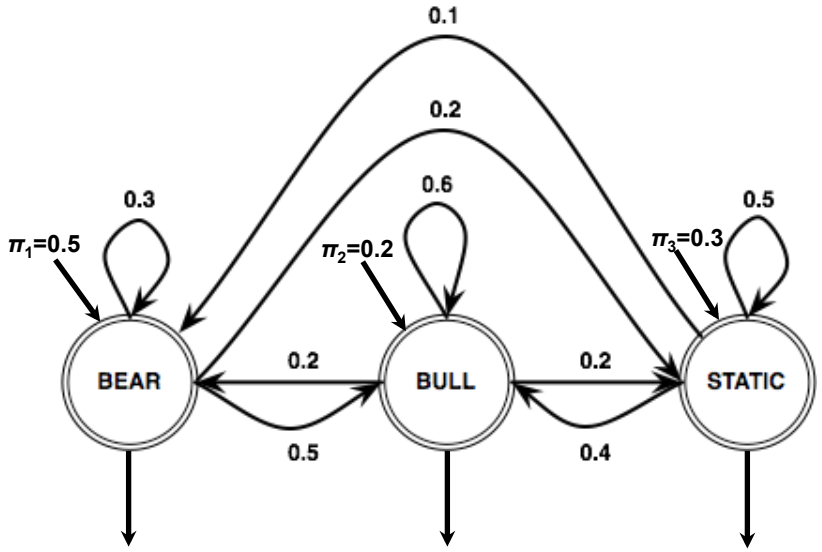
HMMs: Three Problems

- **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



t	1	2	3	4	5	6
O	↑	↓	↔	↑	↓	↔

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

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

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

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

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

Computing Likelihood

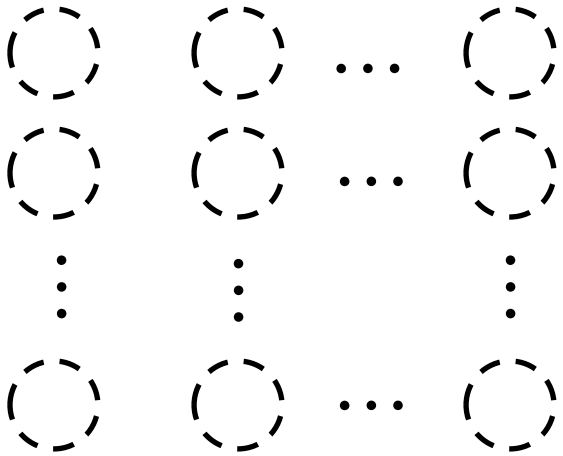
- What are we doing wrong?
  - State sequences may have a lot of overlap
  - We're recomputing the shared subsequences every time
  - Let's store intermediate results and reuse them
- Sounds like a job for dynamic programming!

Forward Algorithm

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

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

- Build an *N* × *T* trellis



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

Forward Algorithm

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

- Initialization

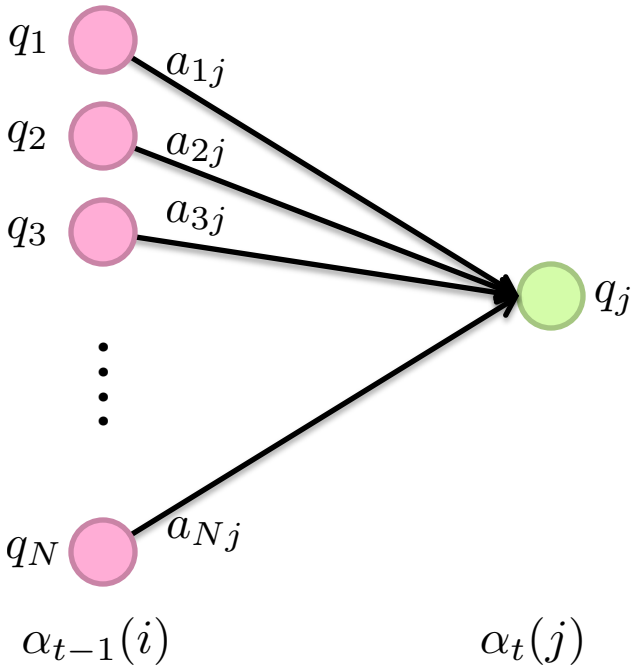
$$\alpha_1(j) = \pi_j b_j(o_i) \quad 1 \leq j \leq N$$

- 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

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

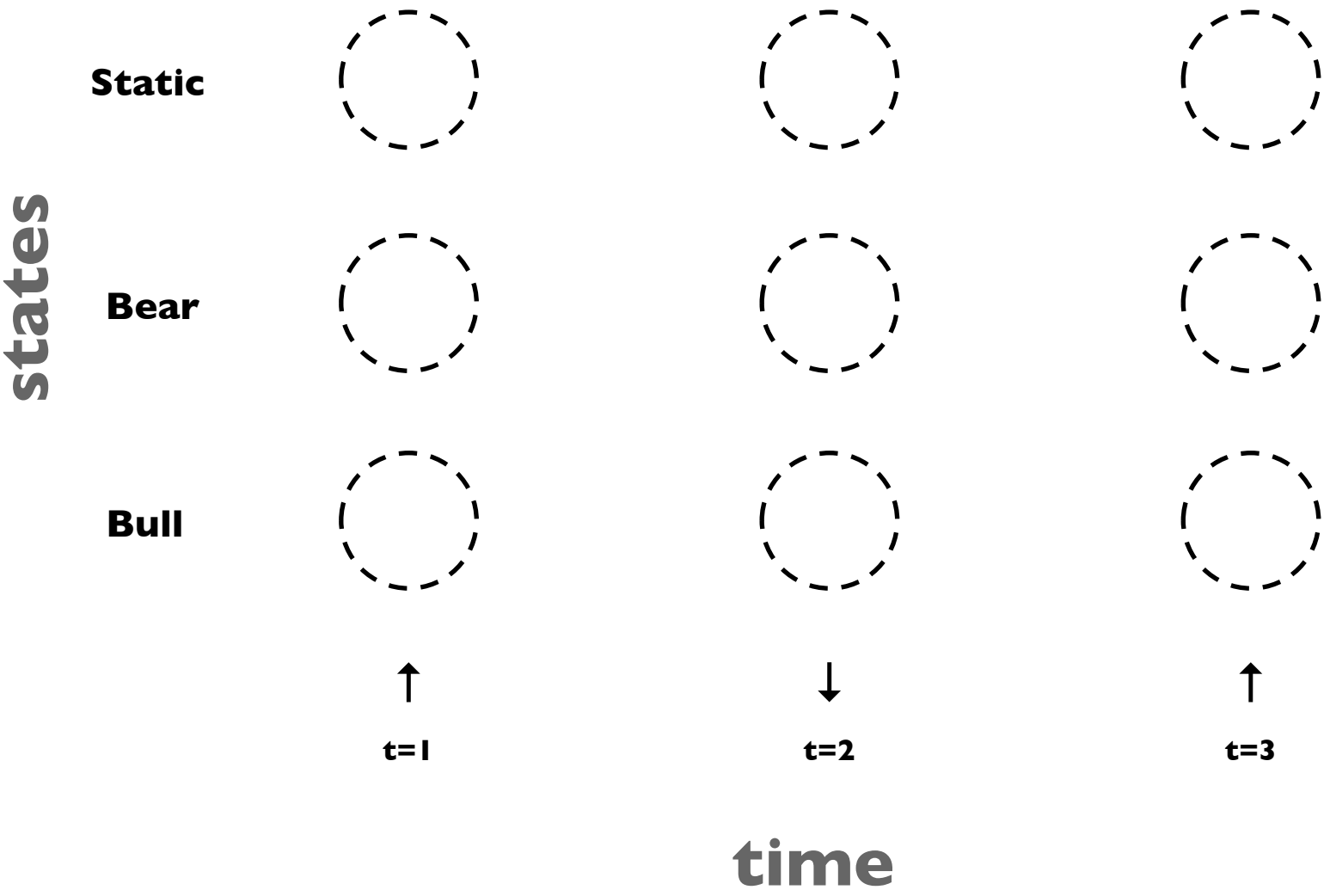


Forward Algorithm

$O = \uparrow \downarrow \uparrow$

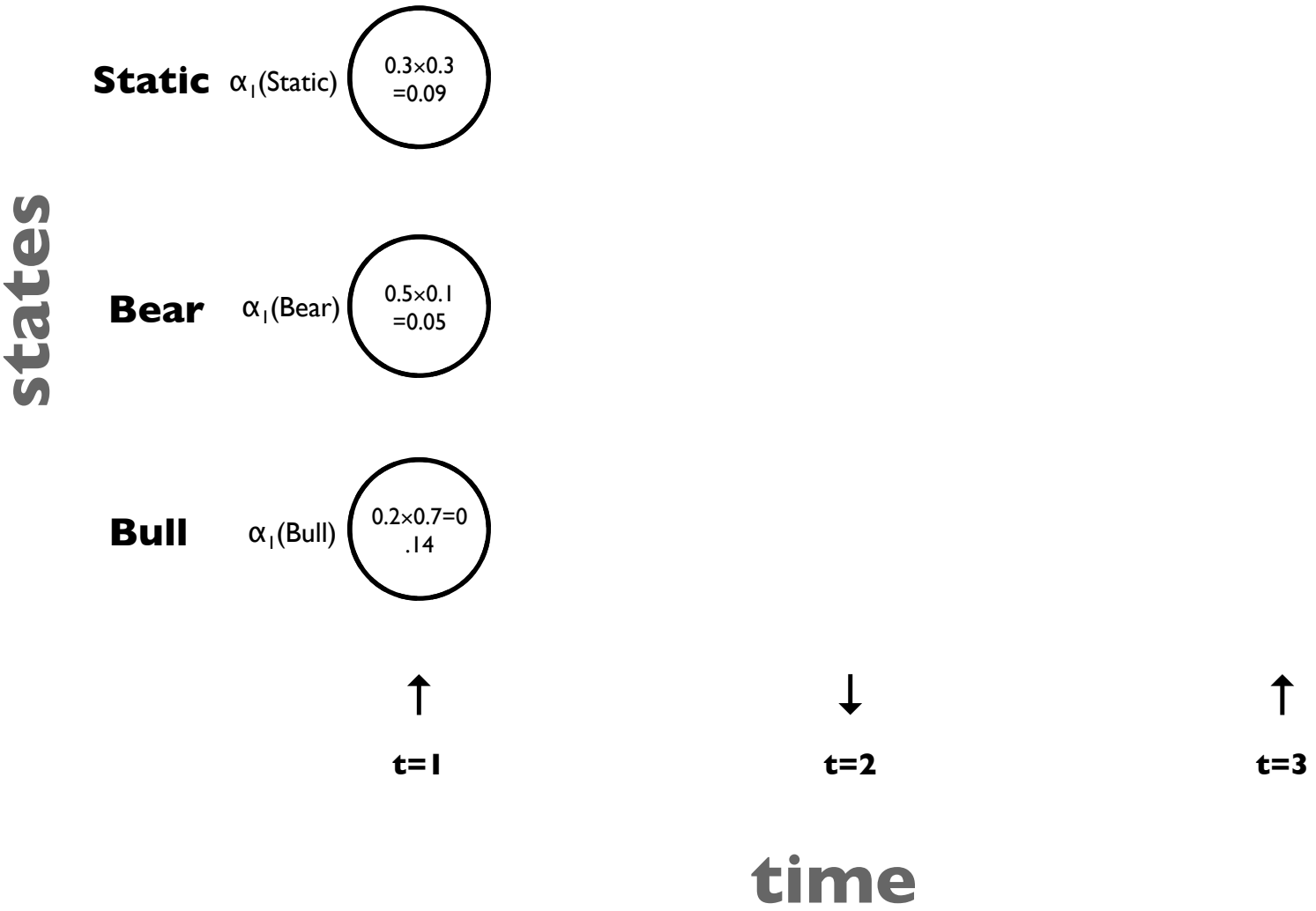
find $P(O|\lambda)$

Forward Algorithm



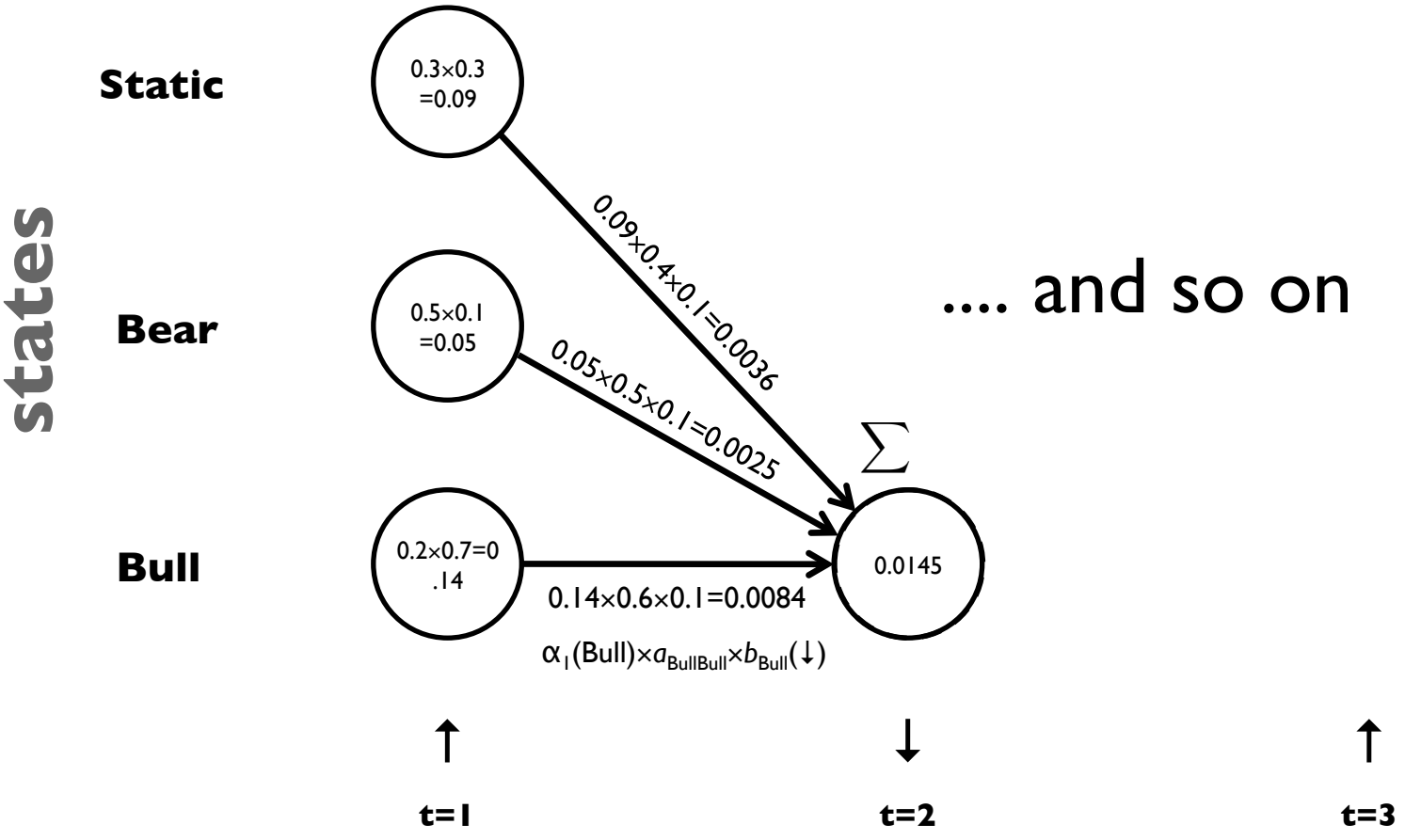
Forward Algorithm: Initialization

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



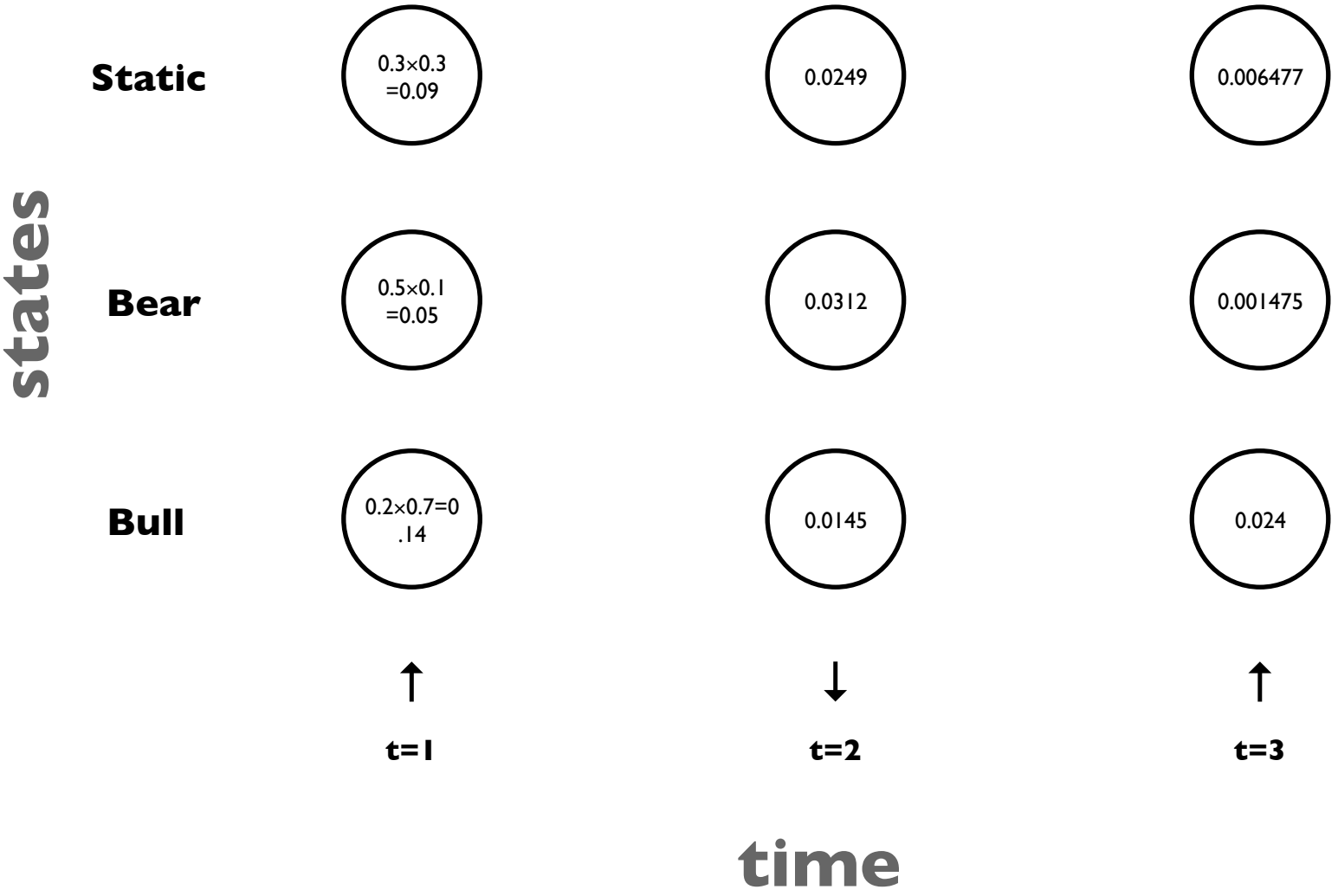
Forward Algorithm: Recursion

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



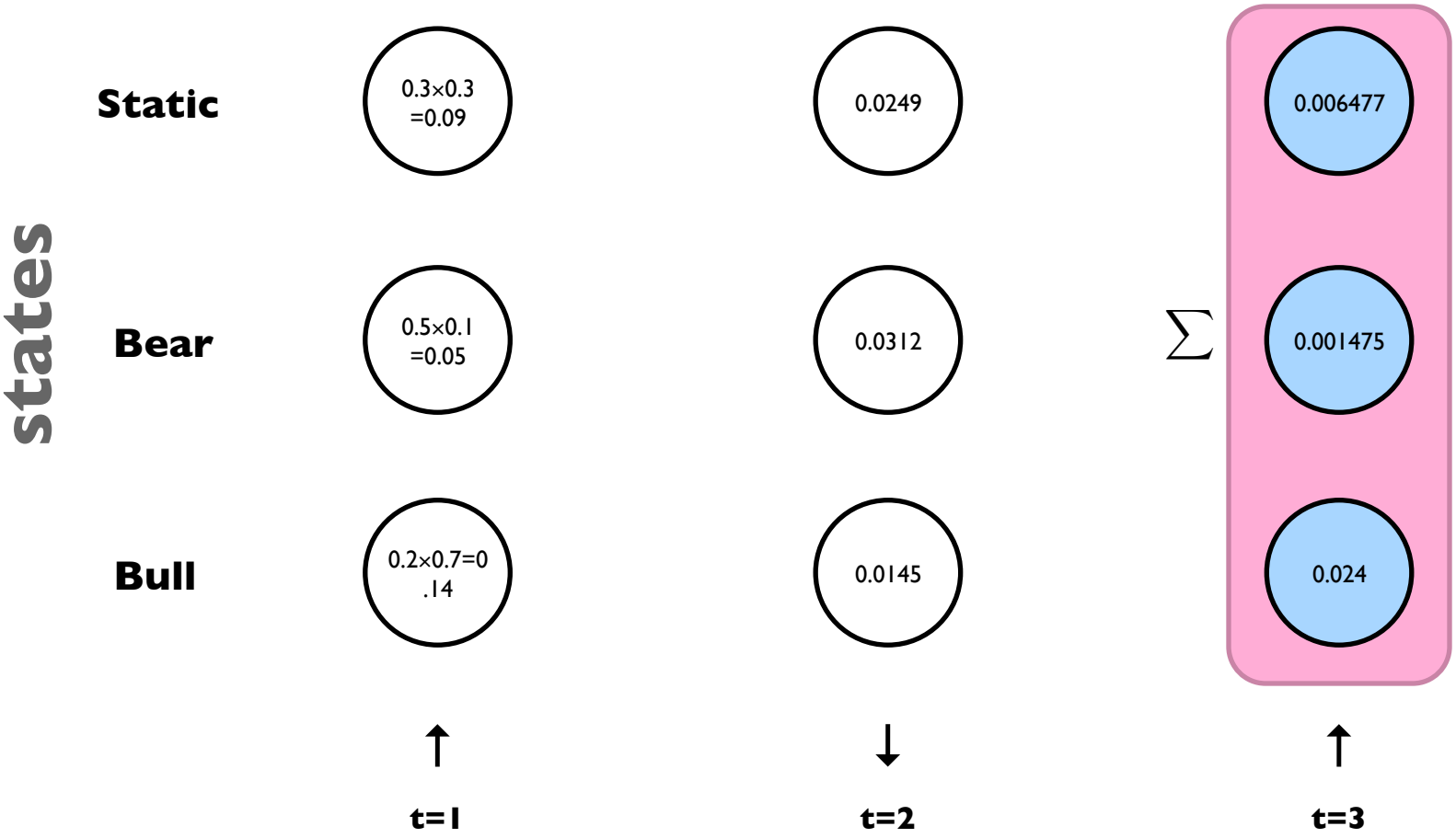
.... and so on

Forward Algorithm: Recursion



Forward Algorithm: Termination

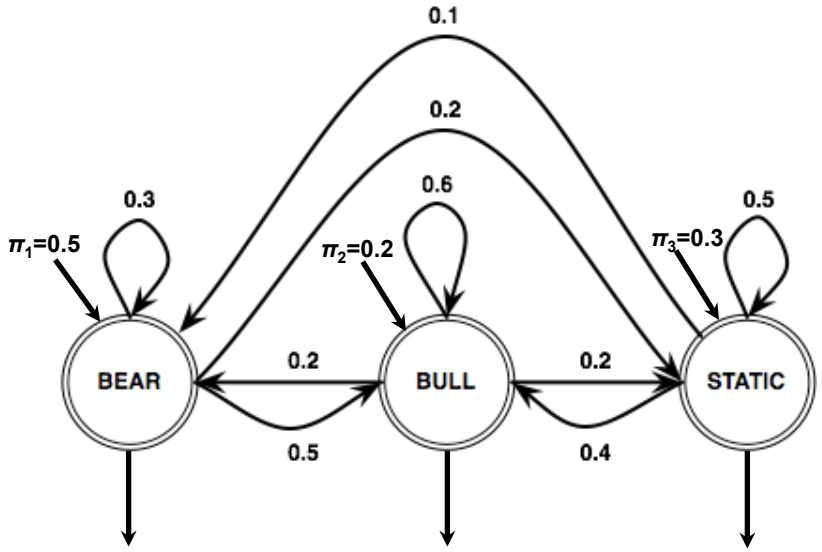
$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$



P(O) = 0.03195

HMM Problem #2: Decoding

Decoding



t	1	2	3	4	5	6
O	↑	↓	↔	↑	↓	↔

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

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

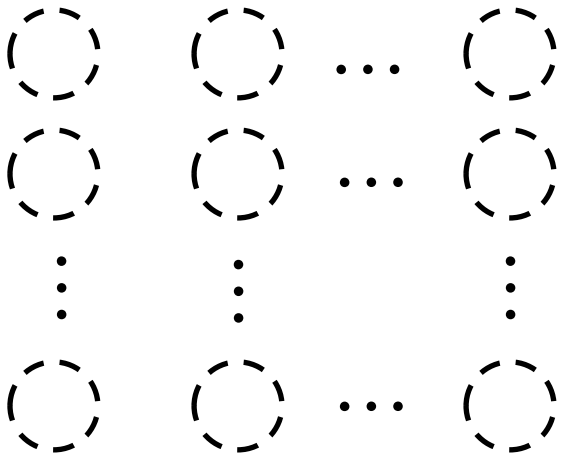
- "Decoding" = computing most likely state sequence
  - Another dynamic programming algorithm
  - Efficient: polynomial vs. exponential
- Same idea as the forward algorithm
  - Store intermediate computation results in a trellis
  - Build new cells from existing cells

Viterbi Algorithm

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

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

- 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

$$v_1(j) = \pi_i b_i(o_1) \qquad 1 \leq i \leq N$$
$$\text{BT}_1(j) = \emptyset \qquad 1 \leq i \leq N$$

- Recursion

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

- Termination

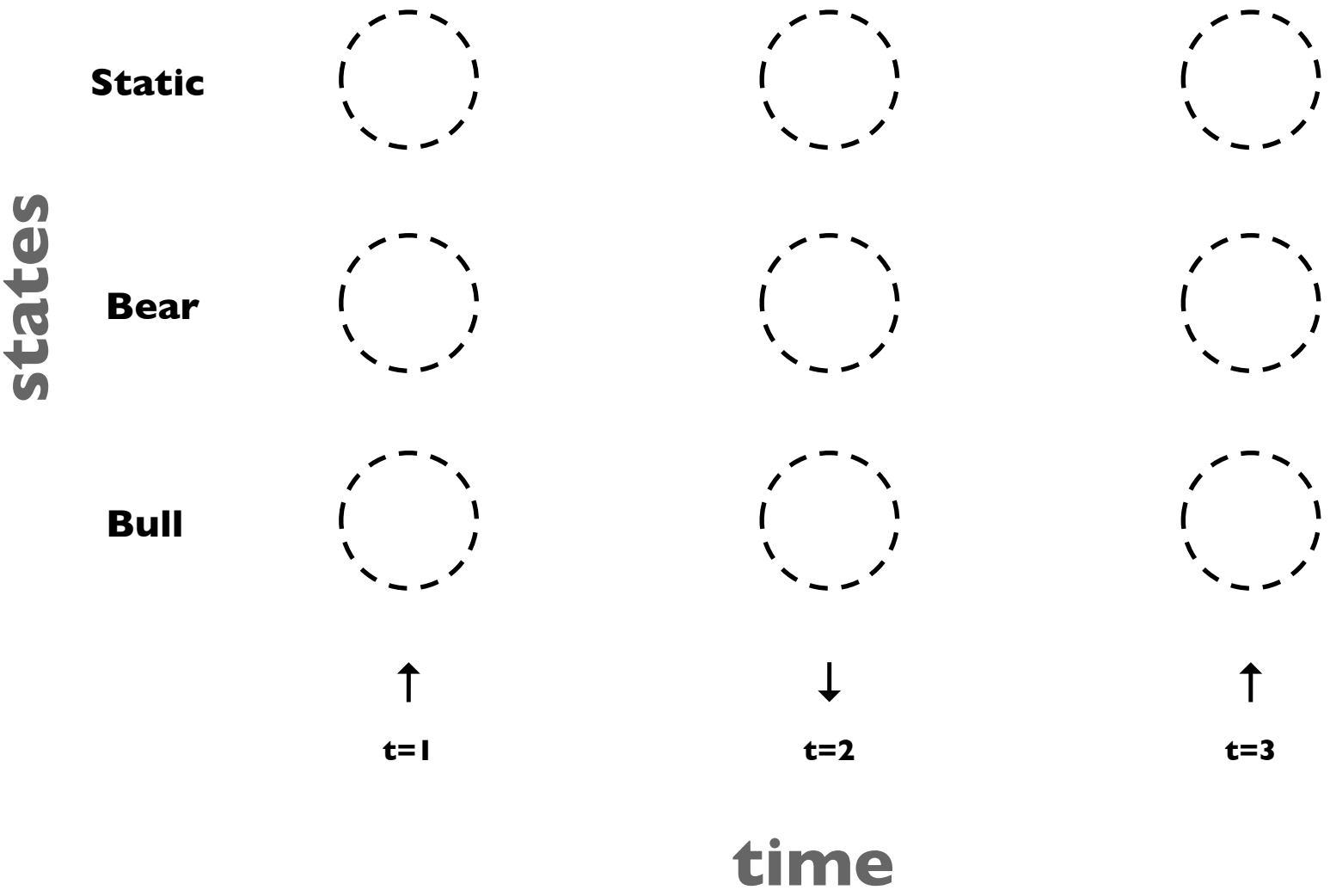
$$P^* = \max_{j=1}^{N} v_T(j)$$
$$q_T^* = \arg\max_{j=1}^{N} v_T(j)$$

Viterbi Algorithm

$O = \uparrow \downarrow \uparrow$

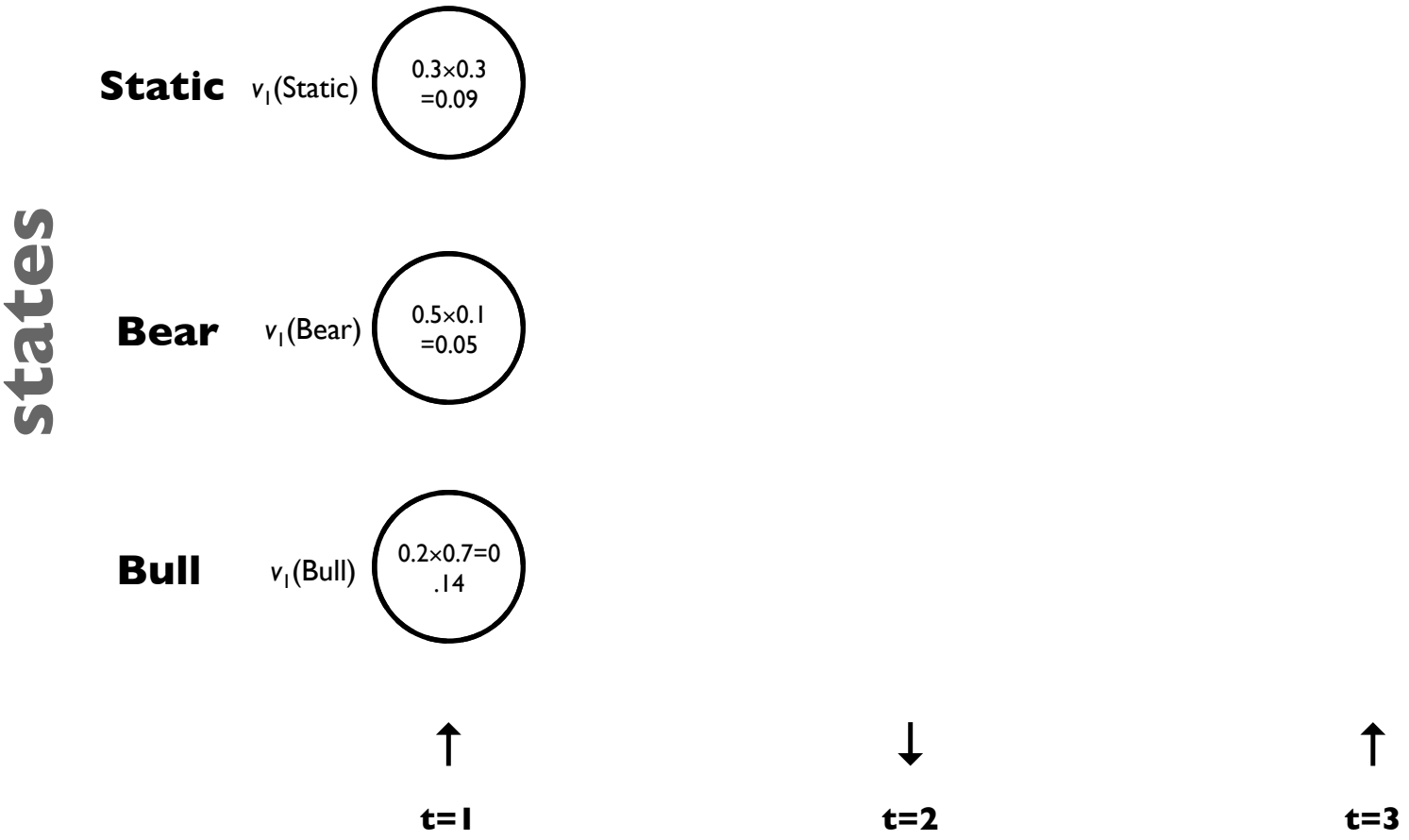
find most likely state sequence

Viterbi Algorithm



Viterbi Algorithm: Initialization

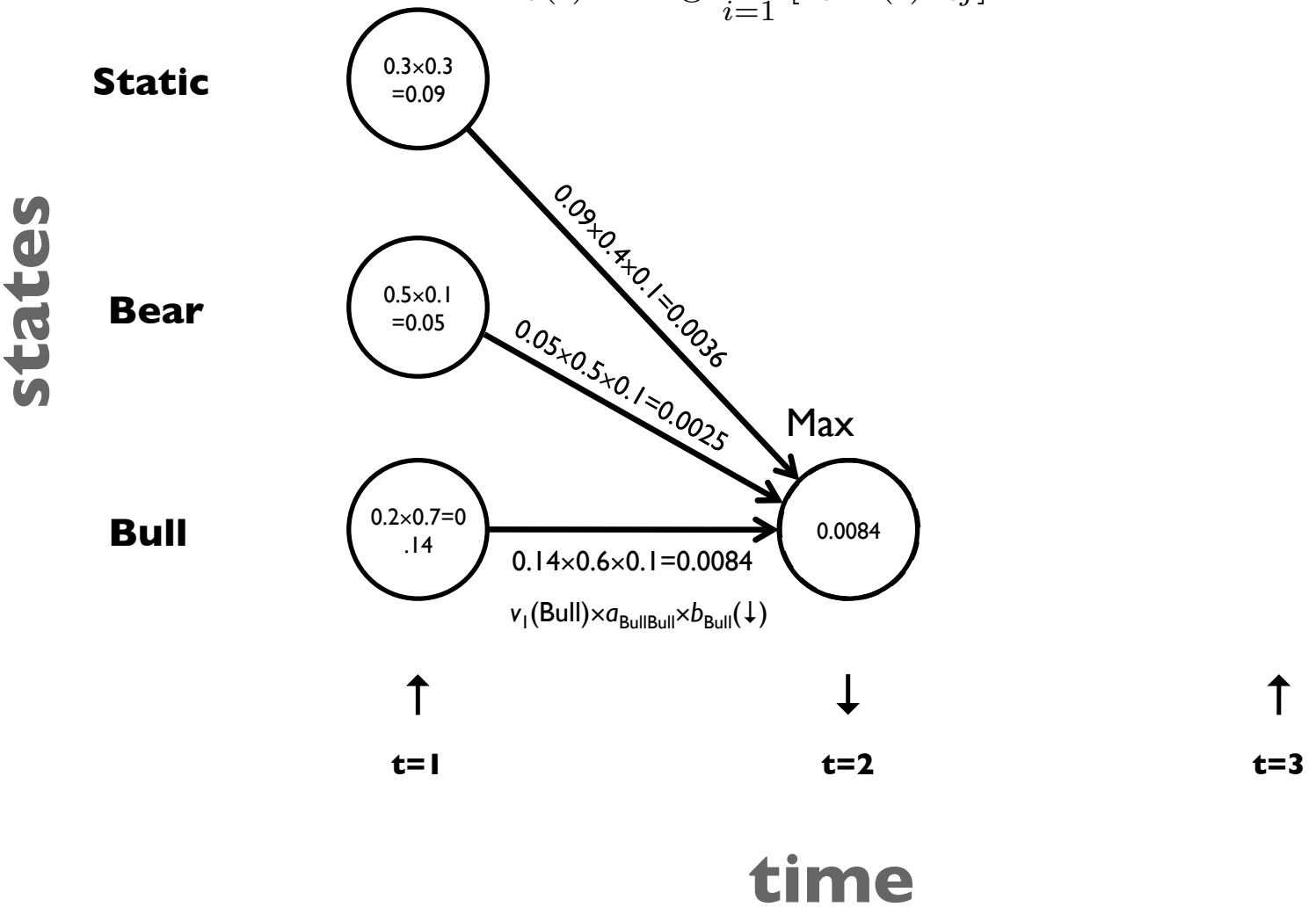
$$v_1(j) = \pi_i b_i(o_1) \qquad 1 \leq i \leq N$$
$$\text{BT}_1(j) = \emptyset \qquad 1 \leq i \leq N$$



Viterbi Algorithm: Recursion

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

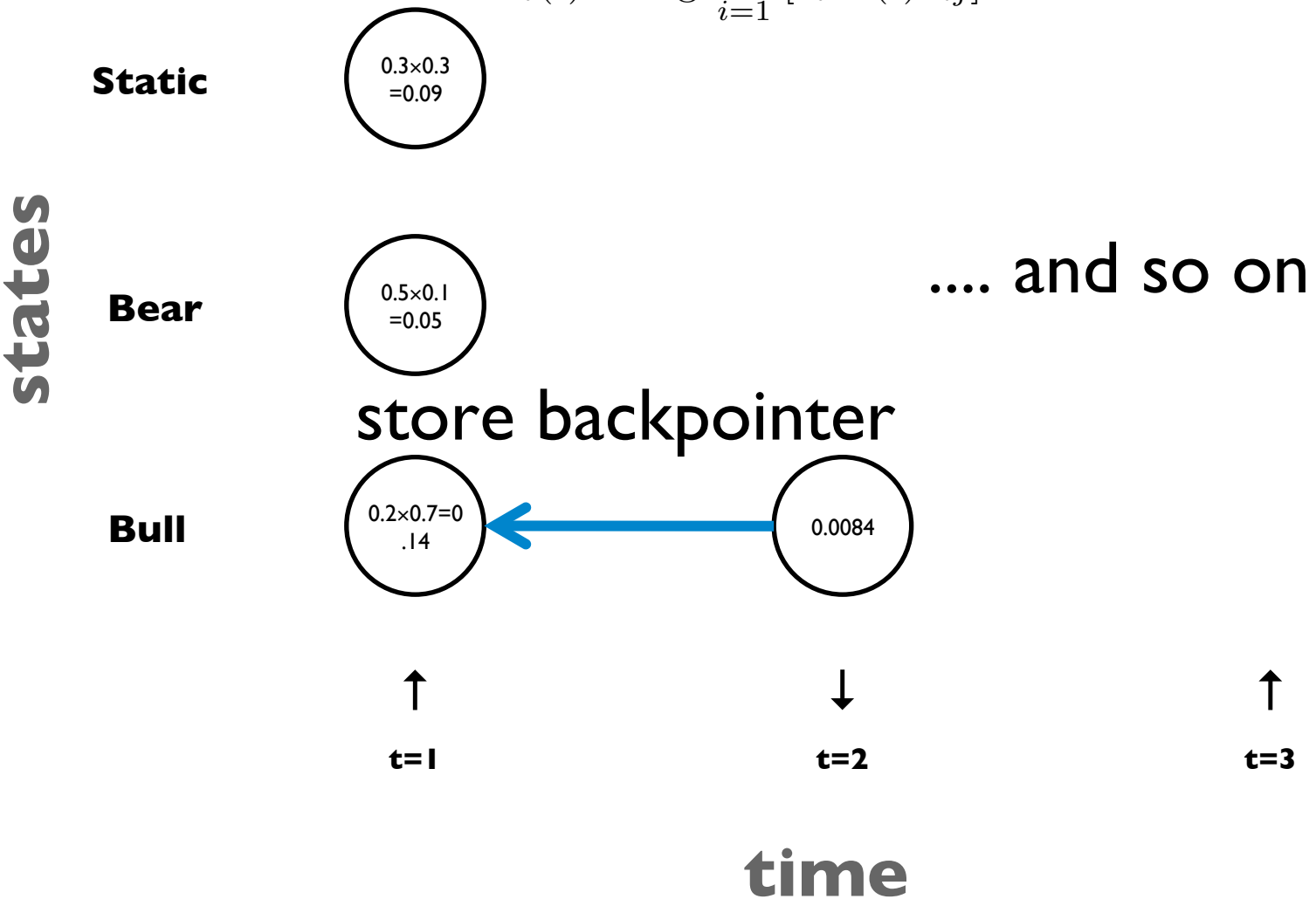
$$\text{BT}_t(i) = \arg\max_{i=1}^{N}[v_{t-1}(i)a_{ij}] \qquad 1 \le i \le N, 2 \le t \le T$$



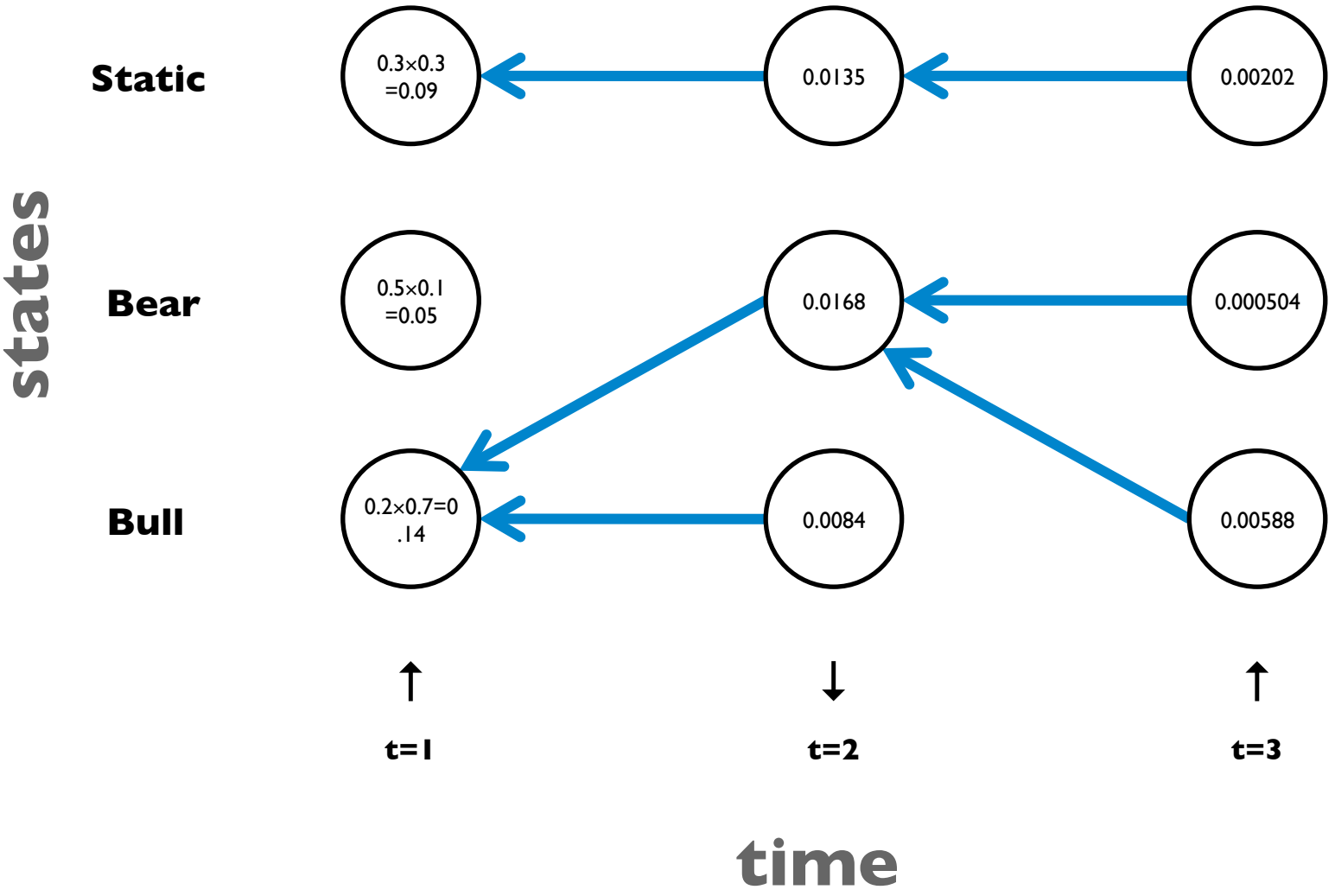
Viterbi Algorithm: Recursion

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

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



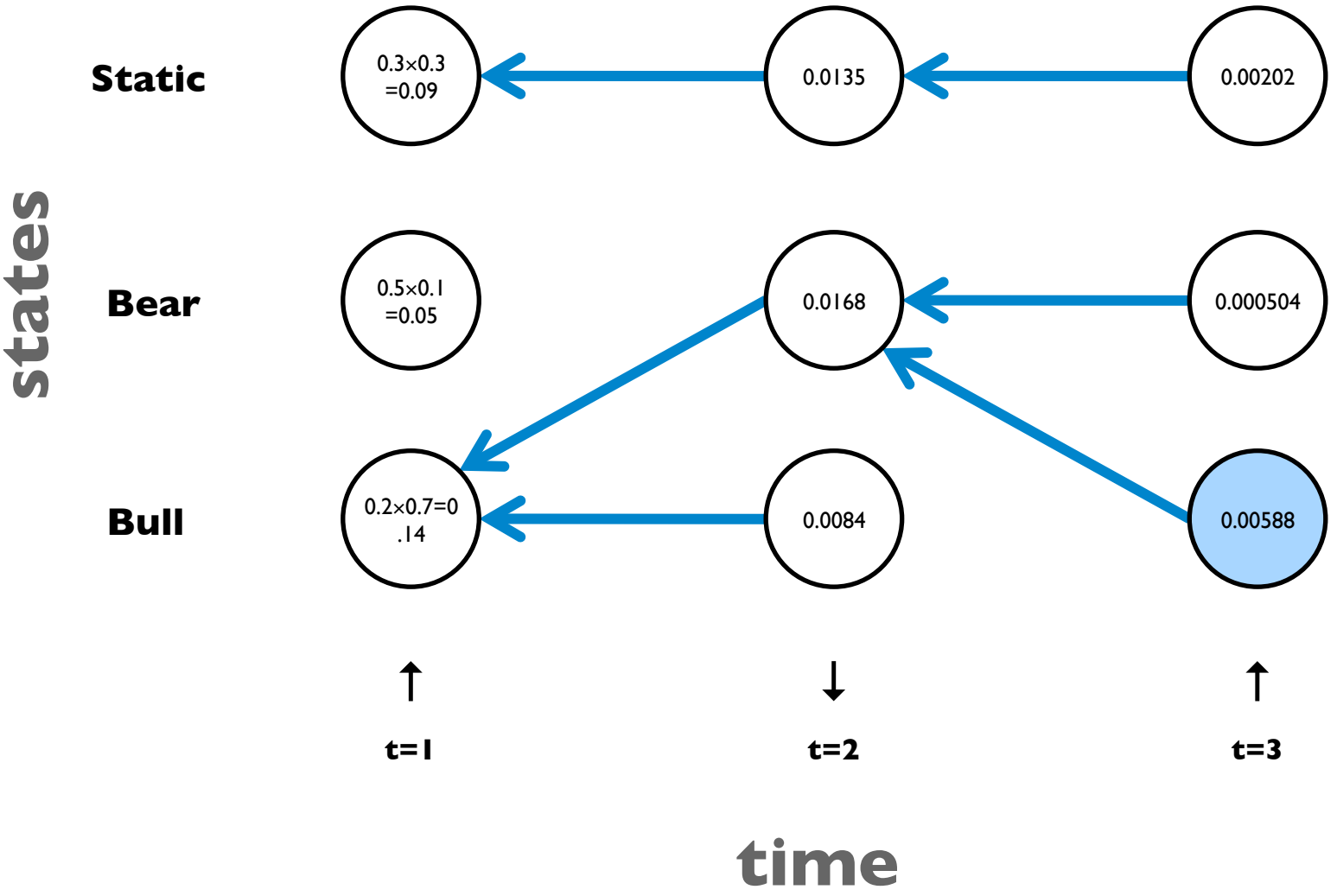
Viterbi Algorithm: Recursion



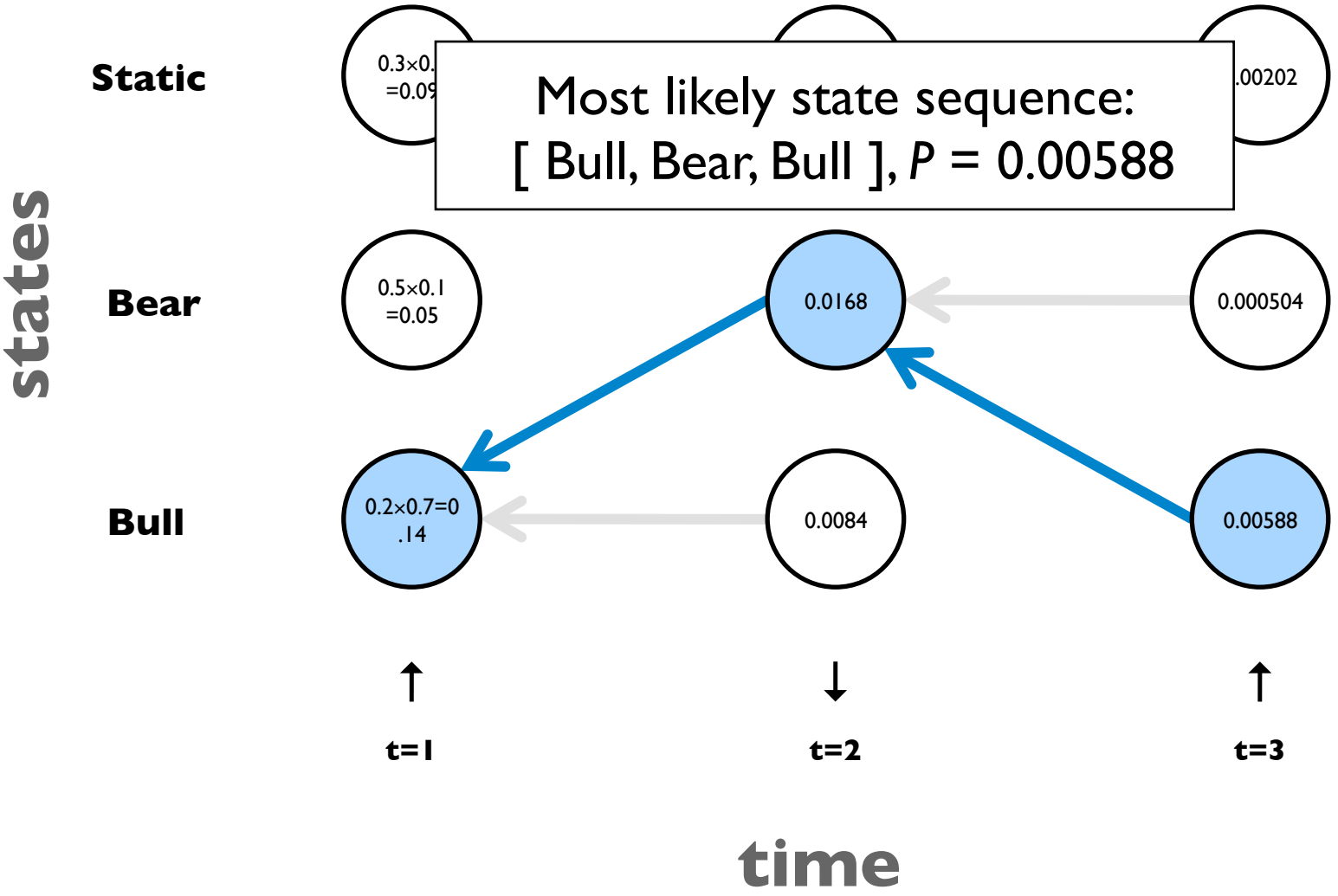
Viterbi Algorithm: Termination

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

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



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:

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

- 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

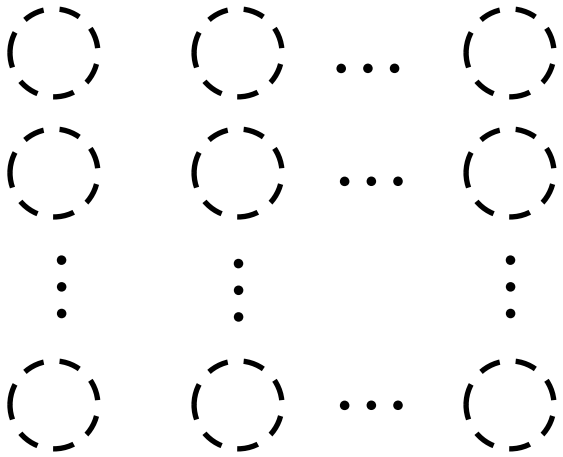
How does this work for HMMs?
First, we need one more definition…

Backward Algorithm

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

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

- Guess what? Build an *N* × *T* trellis



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

- Initialization

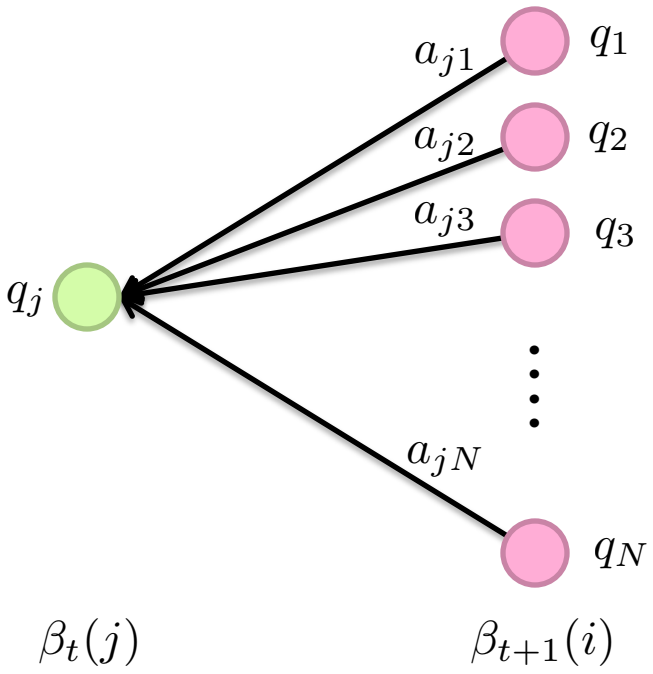
$$\beta_T(j) \quad 1 \leq j \leq N$$

- Recursion

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

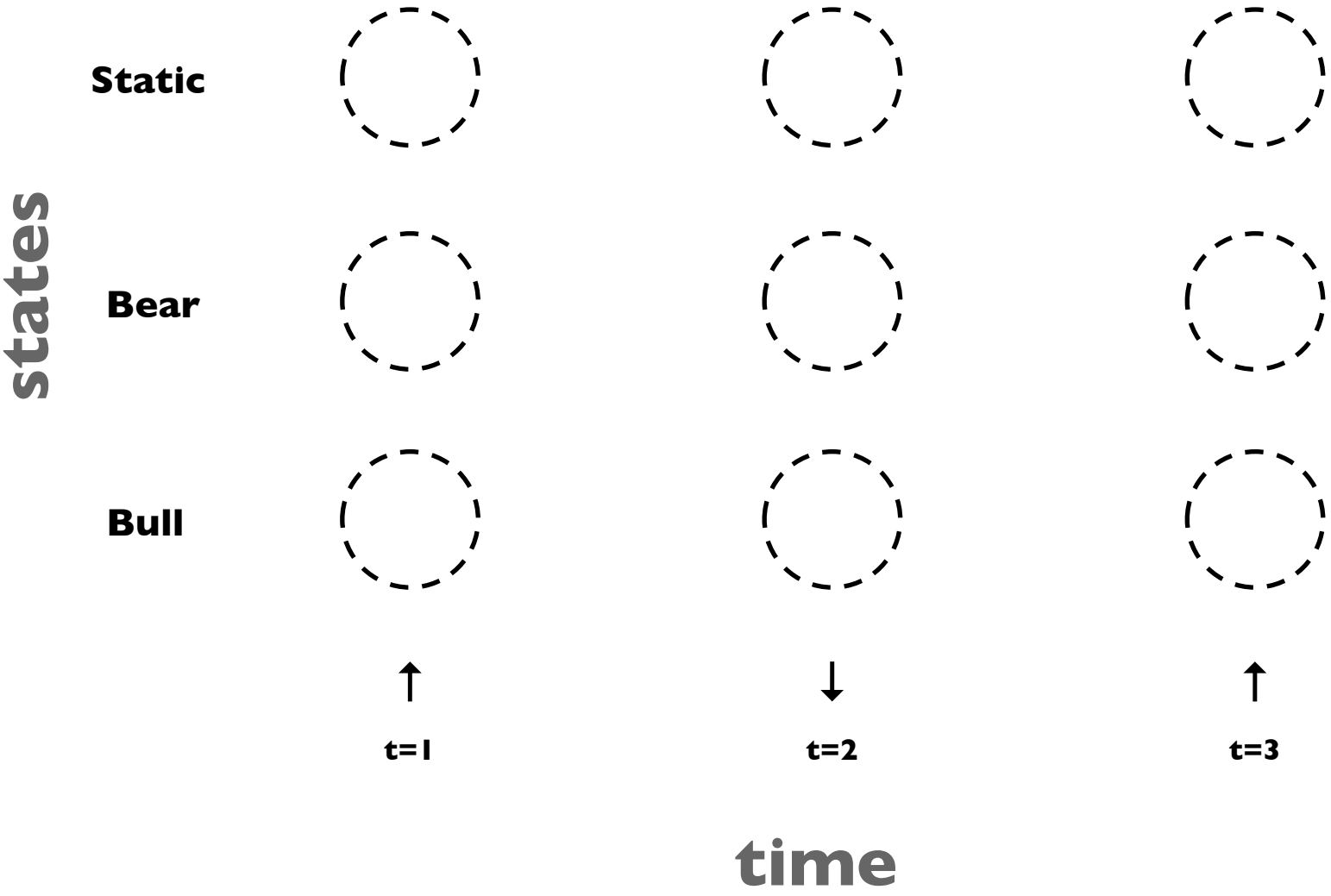
- Termination

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

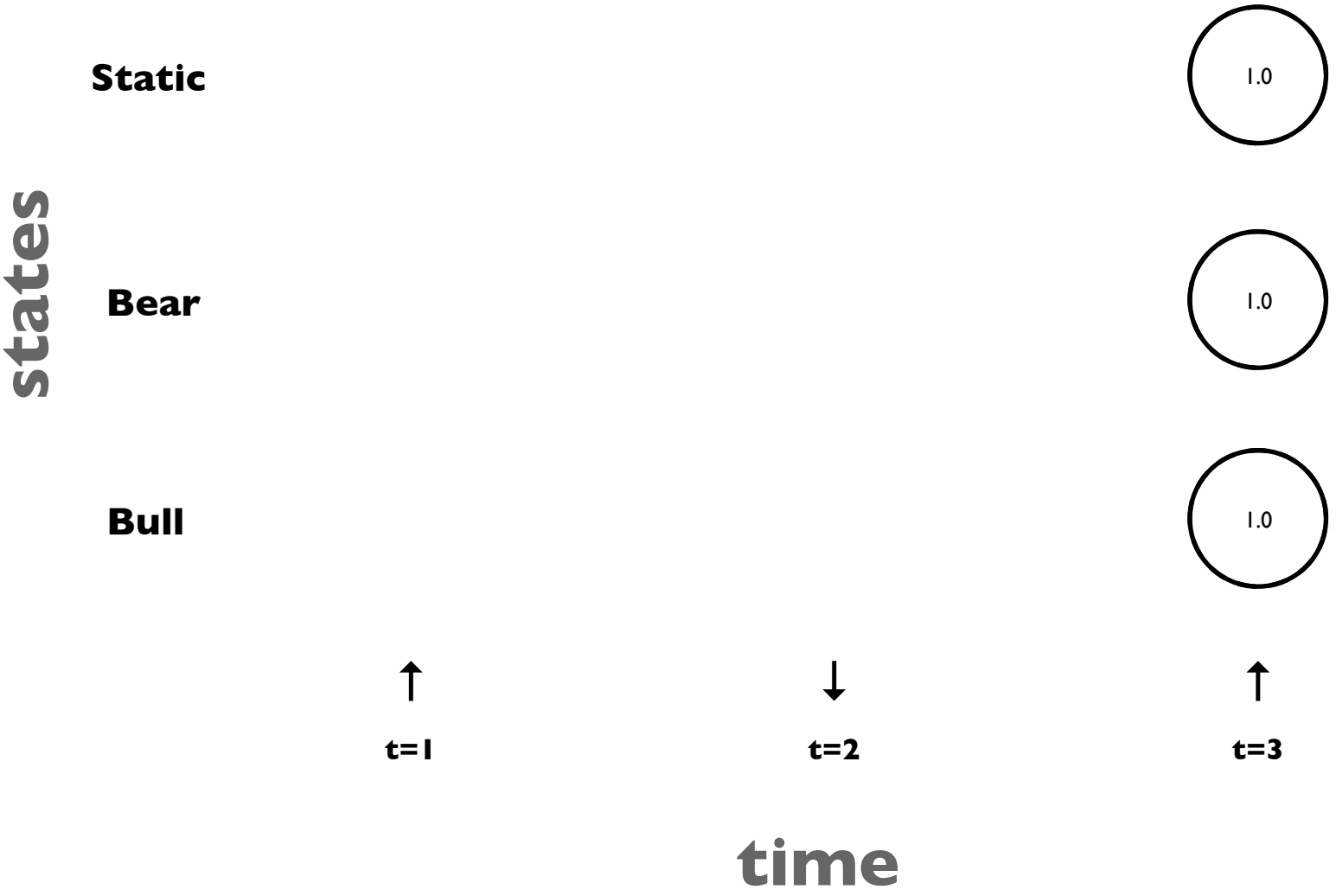


Backward Algorithm

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

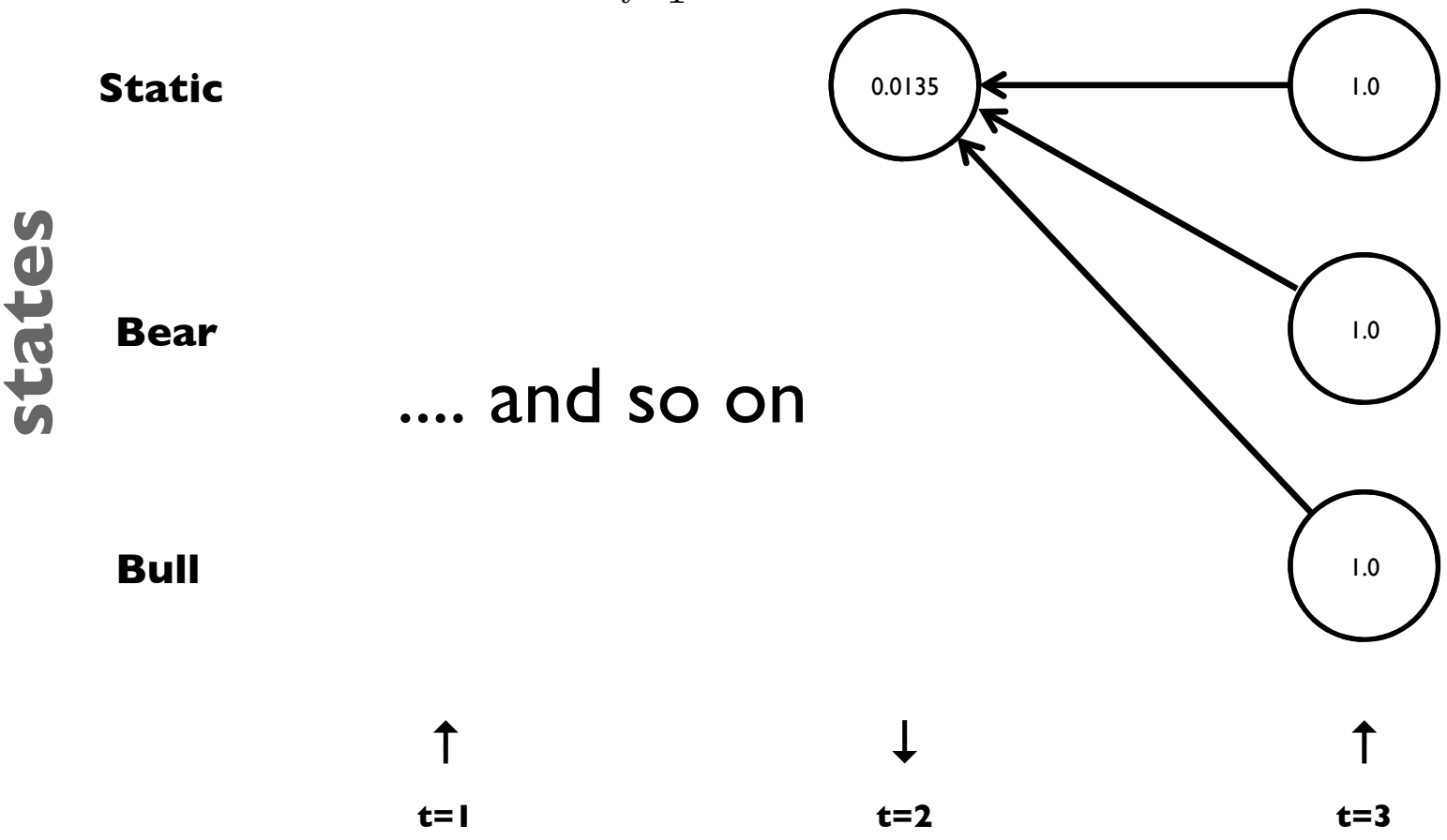


Backward Algorithm: Initialization

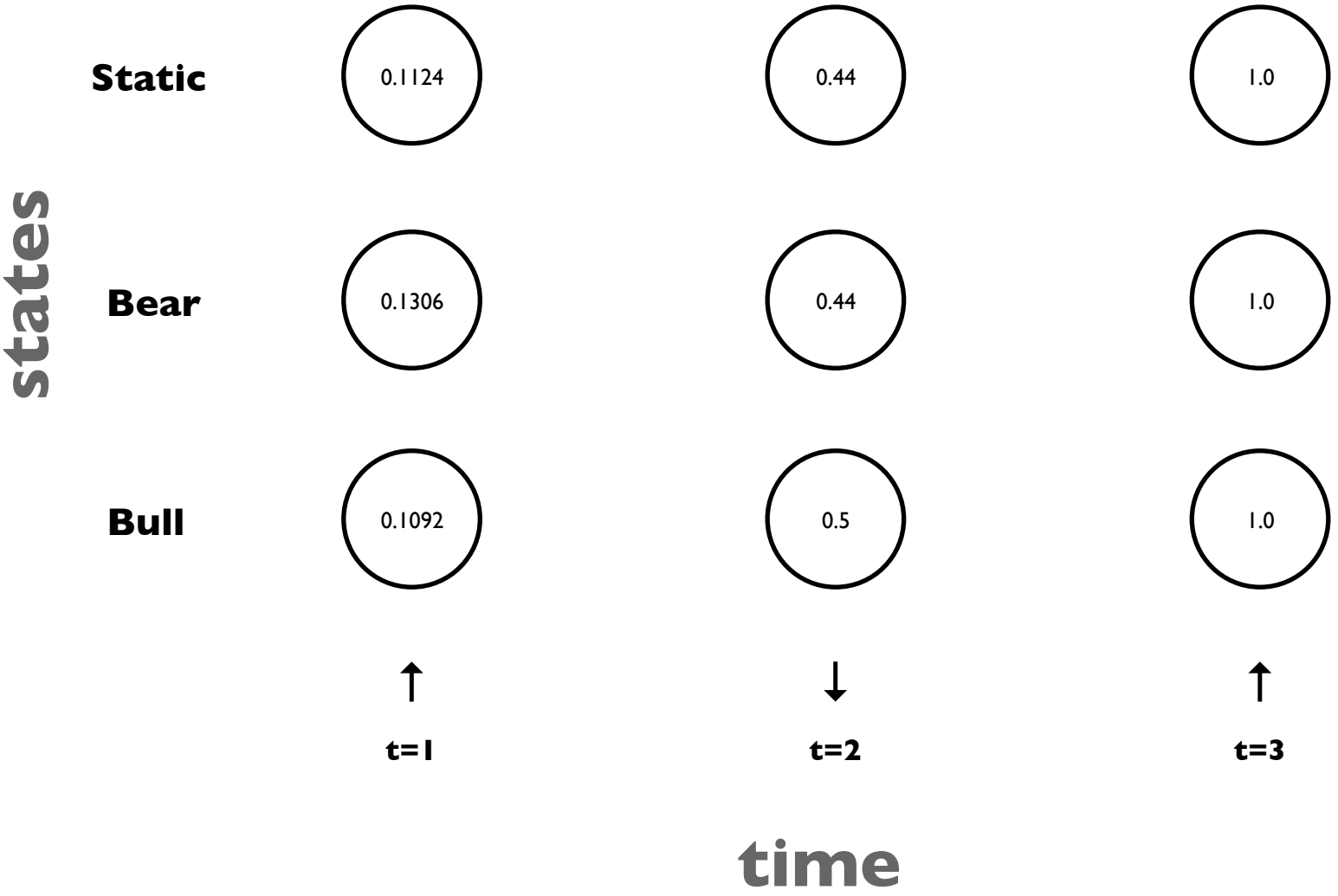


Backward Algorithm: Recursion

$$\beta_t(j) = \sum_{i=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1} \quad 1 \le j \le N, 1 \le t < T$$



Backward Algorithm: Recursion



Forward-Backward

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

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

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

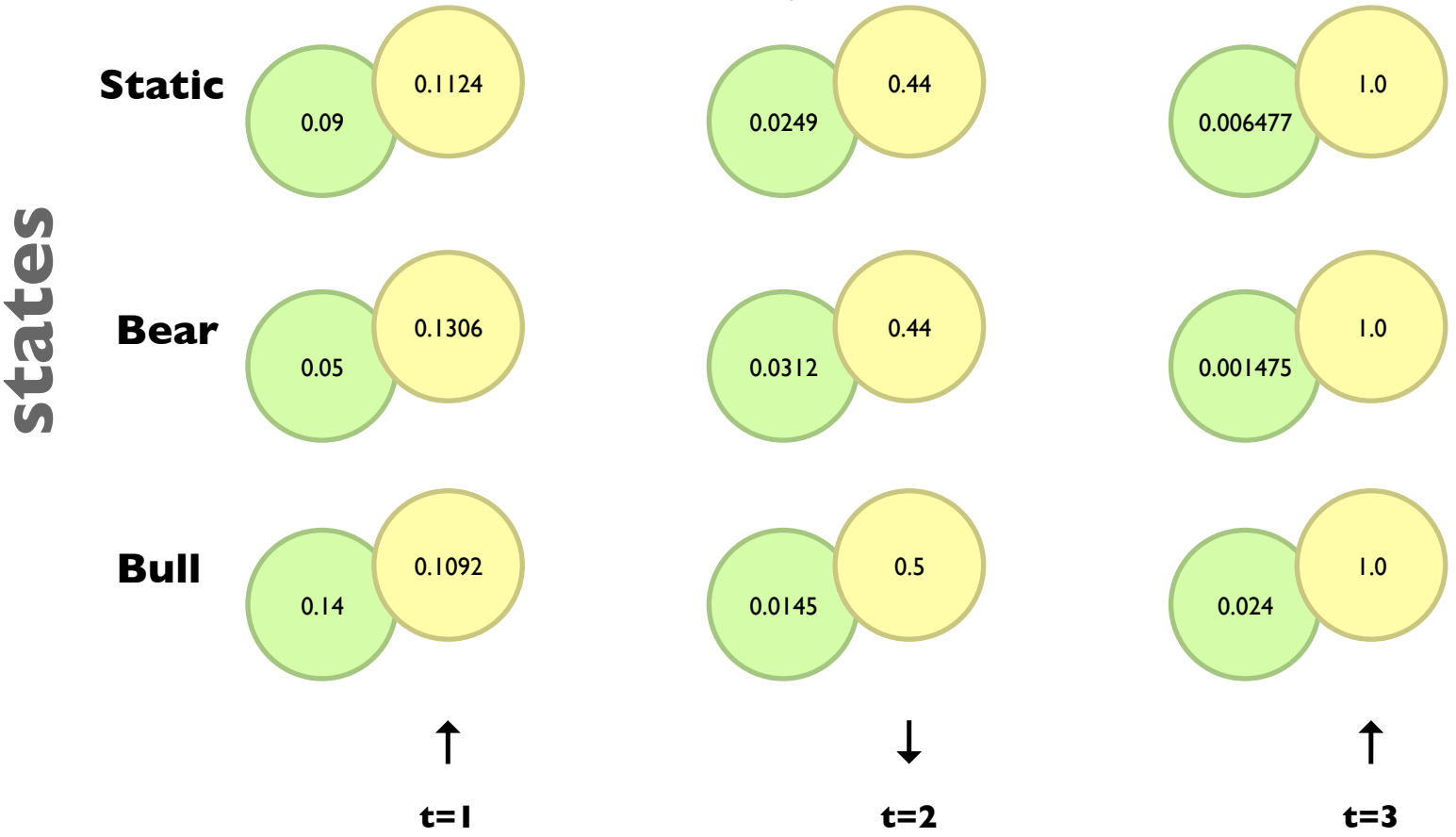
$$\beta_t(j) = P(o_{t+1}, o_{t+2} \ldots o_T|q_t = i, \lambda)$$

Neat Observation:

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

Forward-Backward

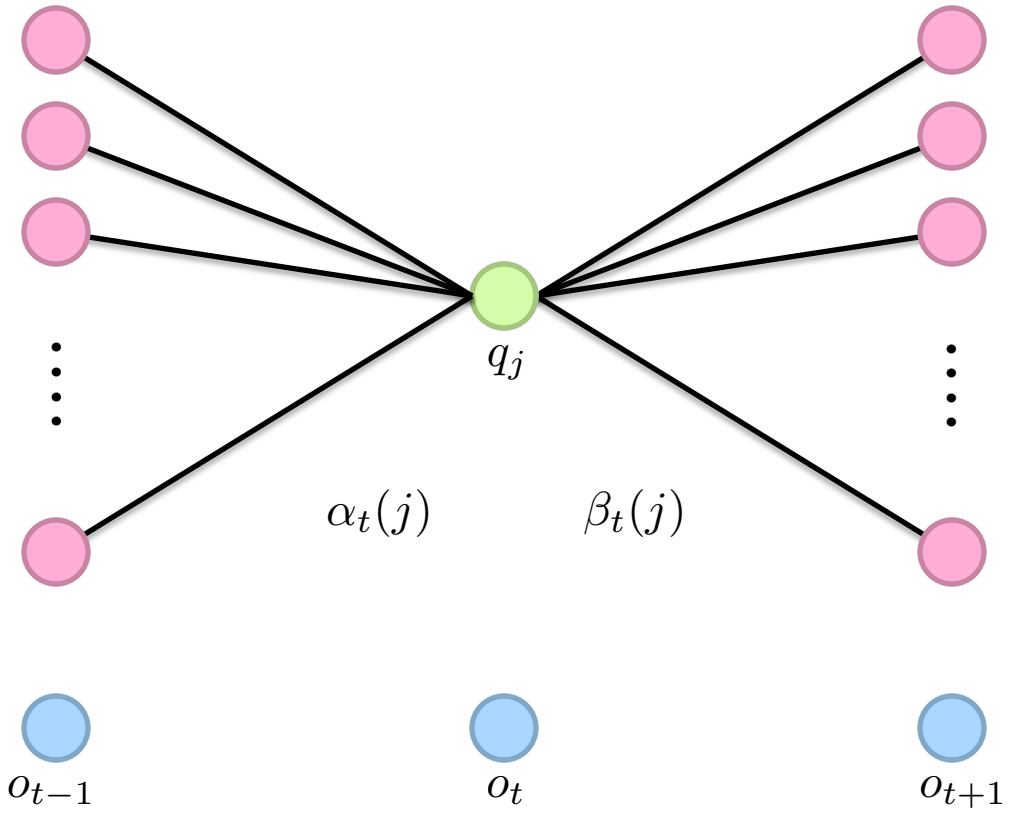
$$P(O|\lambda) = \sum_{j=1}^{N} \alpha_t(j)\beta_t(j)$$



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

Forward-Backward

$$\alpha_t(j) = P(o_1, o_2 \ldots o_t, q_t = j|\lambda) \qquad \beta_t(j) = P(o_{t+1}, o_{t+2} \ldots o_T|q_t = i, \lambda)$$



Estimating Emissions Probabilities

- Basic idea:

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

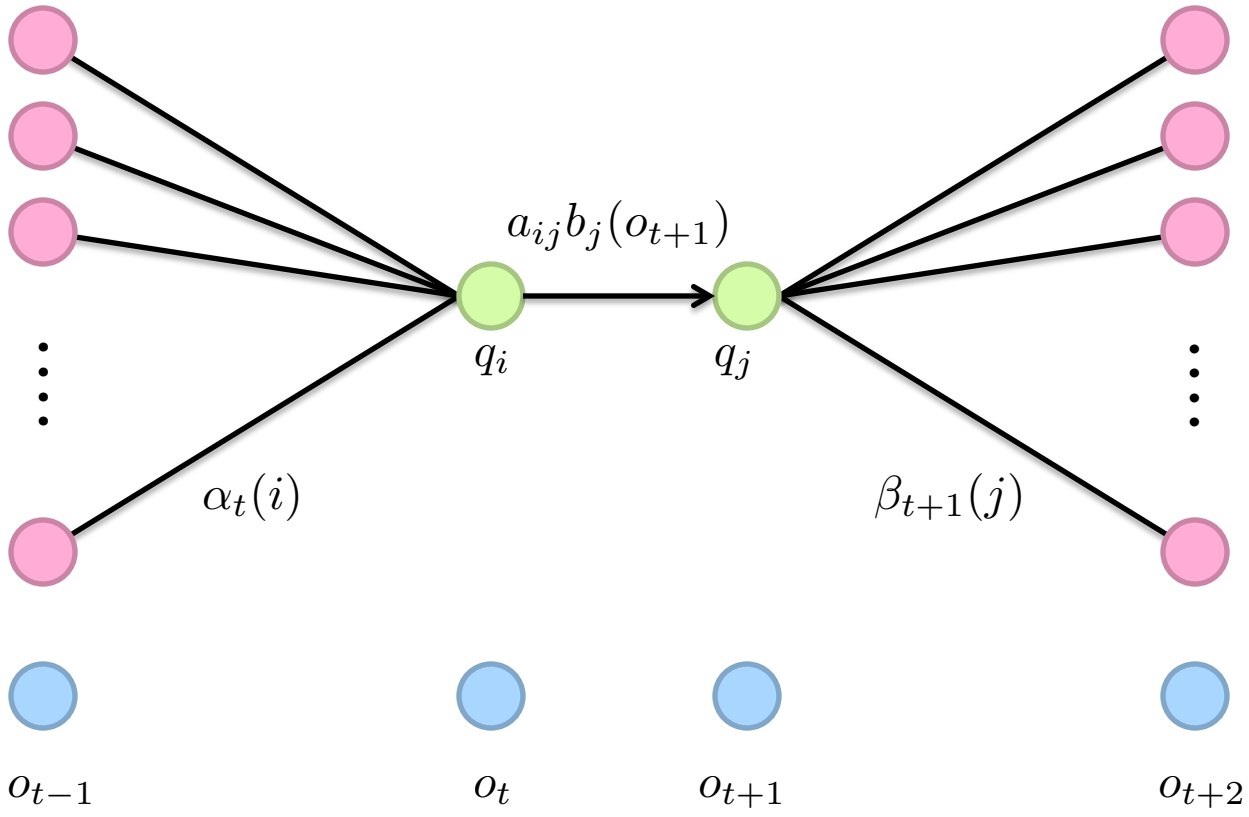
- Let's define:

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

- Thus:

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

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:         ⟨S, O⟩ ← ReadModel
4:         θ ← ⟨A, B, π⟩ ← ReadModelParams(iteration)
5:     method Map(sample id, sequence x)
6:         α ← Forward(x, θ)
7:         β ← Backward(x, θ)
8:         I ← new AssociativeArray
9:         for all q ∈ S do
10:            I{q} ← α_1(q) · β_1(q)
11:        O ← new AssociativeArray of AssociativeArray
12:        for t = 1 to |x| do
13:            for all q ∈ S do
14:                O{q}{x_t} ← O{q}{x_t} + α_t(q) · β_t(q)
15:            t ← t + 1
16:        T ← new AssociativeArray of AssociativeArray
17:        for t = 1 to |x| − 1 do
18:            for all q ∈ S do
19:                for all r ∈ S do
20:                    T{q}{r} ← T{q}{r} + α_t(q) · A_q(r) · B_r(x_{t+1}) · β_{t+1}(r)
21:            t ← t + 1
22:        Emit(string 'initial', stripe I)
23:        for all q ∈ S do
24:            Emit(string 'emit from ' + q, stripe O{q})
25:            Emit(string 'transit from ' + q, stripe T{q})
```

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

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

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

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

MapReduce Implementation: Reducer

```
1: class Combiner
2:     method Combine(string t, stripes [C_1, C_2, ...])
3:         C_f ← new AssociativeArray
4:         for all stripe C ∈ stripes [C_1, C_2, ...] do
5:             Sum(C_f, C)
6:         Emit(string t, stripe C_f)
```

```
1: class Reducer
2:     method Reduce(string t, stripes [C_1, C_2, ...])
3:         C_f ← new AssociativeArray
4:         for all stripe C ∈ stripes [C_1, C_2, ...] do
5:             Sum(C_f, C)
6:         z ← 0
7:         for all ⟨k, v⟩ ∈ C_f do
8:             z ← z + v
9:         P_f ← new AssociativeArray
10:        for all ⟨k, v⟩ ∈ C_f do
11:            P_f{k} ← v/z
12:        Emit(string t, stripe P_f)
```

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

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

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

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

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

Source: Wikipedia (Japanese rock garden)

Midterm

Final Projects

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