CMSC 723/LING 723
Computational Linguistics I

Hidden Markov Models
Expectation Maximization

Lecture 6
October 3, 2007

Professor : Bonnie Dorr
Co-instructor : Nitin Madnani
TAs : Hamid Shahri, Alexandros Tzannes

Administrivia

- Asking questions
  - Use the forum (http://forum.cs.umd.edu)
  - Use existing FAQ thread
  - Set up daily notifications for forum
    Forum Tools → Subscribe
  - Email should only be used as a last resort
    Please CC all 4 of us !
- Assignment 3 out; Assignment 2 due !
  http://www.umiacs.umd.edu/~nmadnani/teaching/723/assignments/3/assignment3.zip

Lecture Organization

- Different from book order
- More intuitive
  - Markov Chains, HMMs & algorithms first (Sections 6.1-6.4)
  - Applying HMMs to POS tagging (Section 5.5)
  - How to train HMMs (Section 6.5)
- Available (for now) at: http://www.umiacs.umd.edu/~nmadnani/teaching/723/lectures/Lecture6.pdf

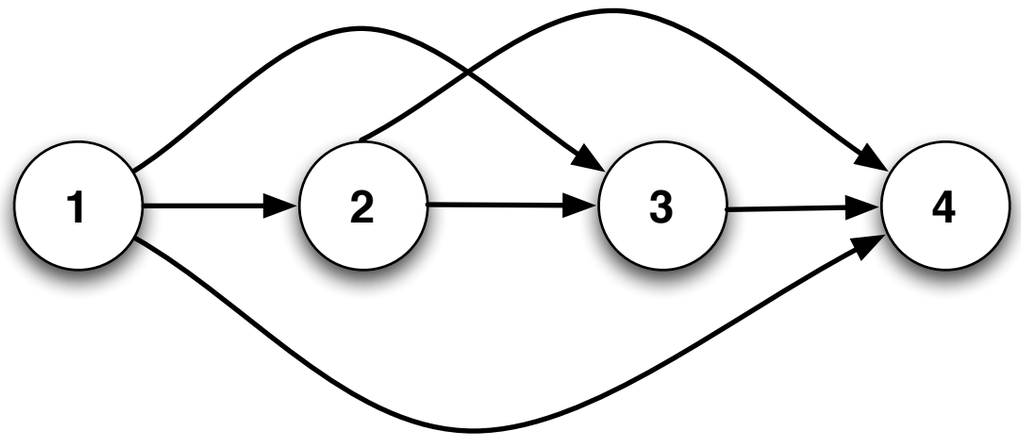
Recap: Finite State Machines

What do you need to specify an FSM formally ?

Finite State Machines

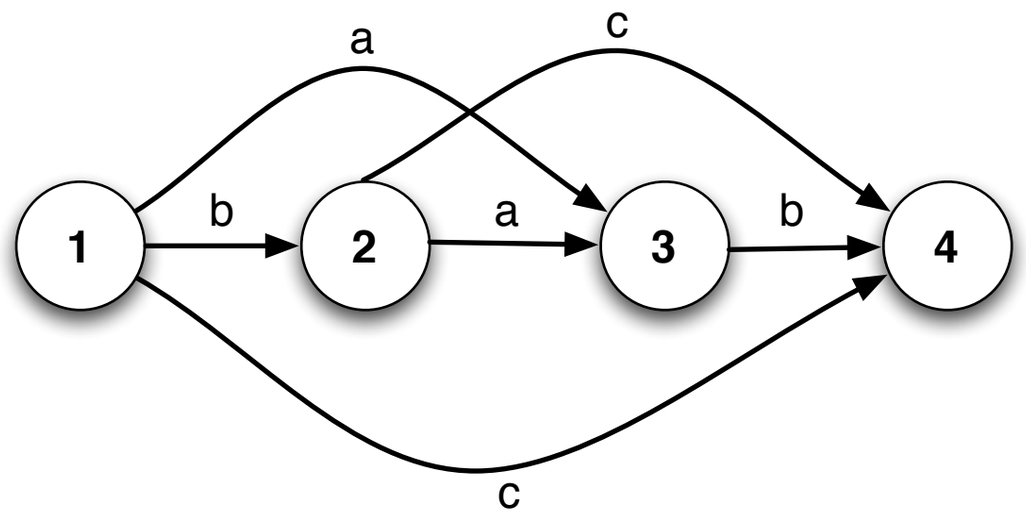
Finite number of states

Finite State Machines



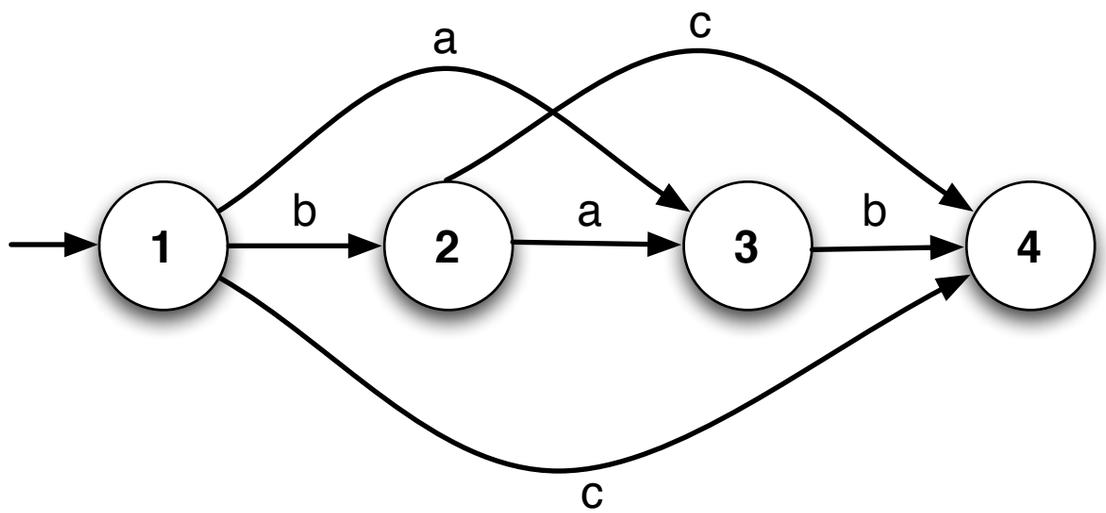
Transitions

Finite State Machines



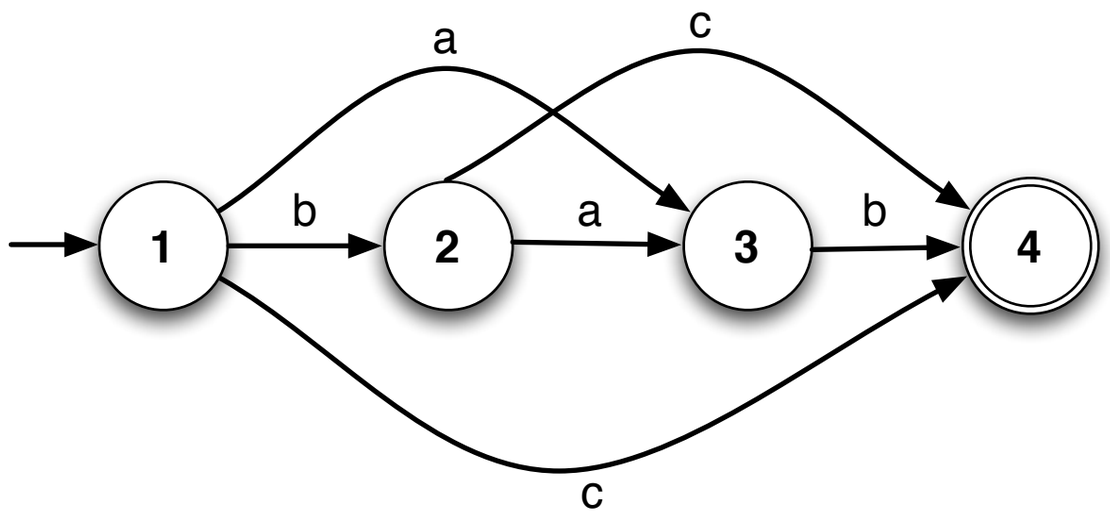
Input alphabet

Finite State Machines



Start state

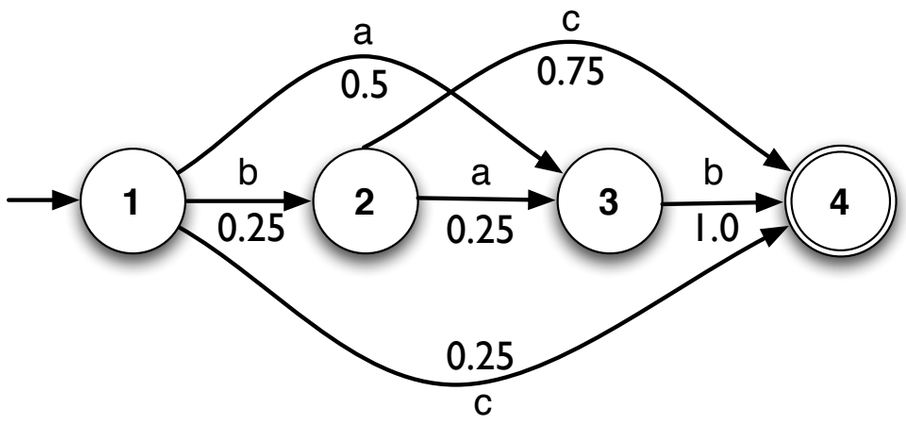
Finite State Machines



Final state(s)

Using Real-world Knowledge

Probabilistic FSMs



‘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’

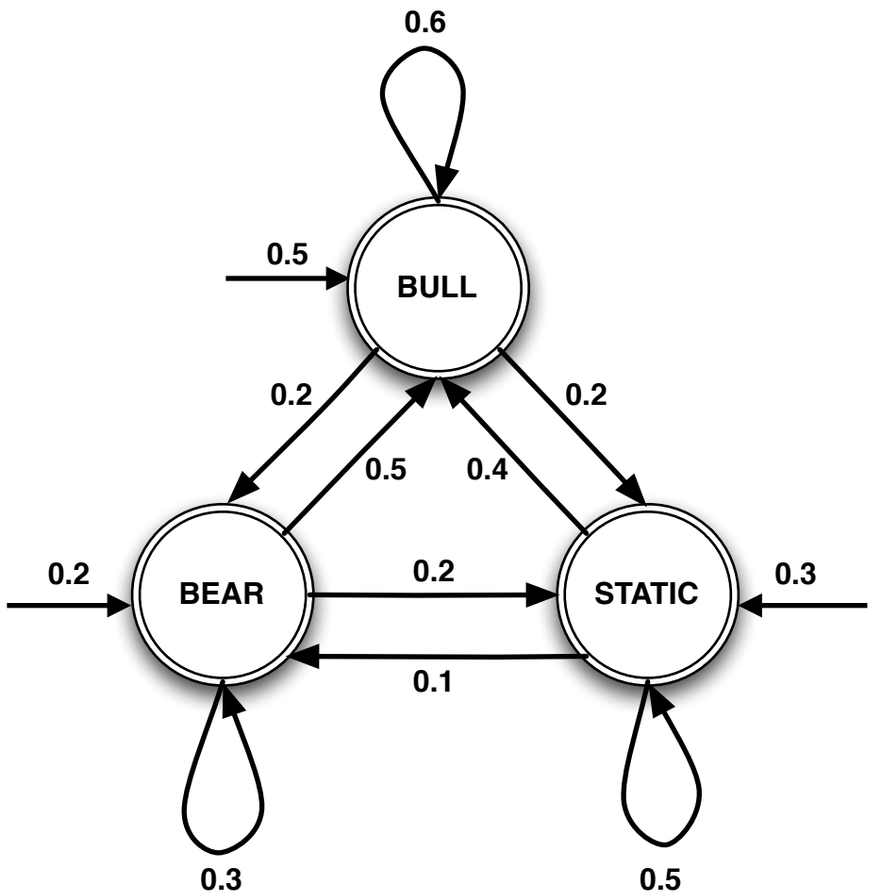
The prob. of all arcs going out of a state *must* sum to 1

What do we get out of it ?

P(‘ab’) = 0.50 * 1.00 = 0.5000

P(‘bc’) = 0.25 * 0.75 = 0.1875

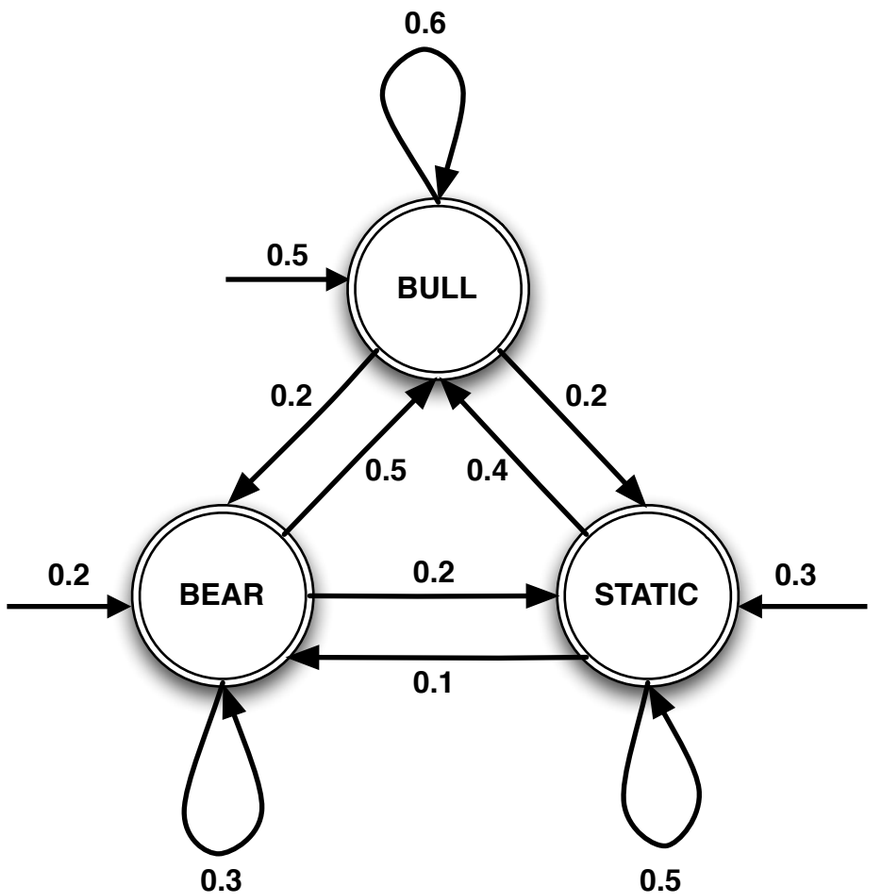
Markov Chains



The stock market

- Is this a valid prob. FSM ?
  No !
- What's missing ?
  The start state
- Can we pick one to start from ?
  No !
- So ?
  Use "prior probabilities"

Markov Chains



The stock market

- What's special about this FSM ?

  Prob. of any state ONLY depends on previous state

- The (1st order) Markov assumption

  $$P(q_i|q_1, q_2, \ldots, q_{i-1}) = P(q_i|q_{i-1})$$

- This extension of a prob. FSM is called a *Markov Chain* or an *Observed Markov Model*

- Each state corresponds to an observable physical event

Are states always observable ?

Day: 1 2 3 4 5 6

Not observable !

B_u B_e S B_e S B_u

Bu: Bull Market
Be: 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 are usually inadequate
- Need to model problems where observed events don't correspond to states directly
- Instead, observations = f_p(states) for some p.d.f *p*
- Solution: A Hidden Markov Model (HMM)
  - Assume two probabilistic processes
  - Underlying process is hidden (states = hidden events)
  - Second process produces sequence of observed events

Formalizing HMMs

- An HMM $\lambda = (A, B, \Pi)$ is characterized by:
  - Set of N states $\{q_1, q_2, ..., q_N\}$
  - N x N Transition probability matrix $A = [a_{ij}]$

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

  - Sequence of observations $o_1, o_2, ... o_T$, each drawn from a given set of symbols (vocabulary V)
  - N x |V| Emission probability matrix, $B = [b_{it}]$

$$b_{it} = b_i(o_t) = p(o_t|q_i)$$

  - N x 1 Prior probabilities vector $\Pi = \{ \Pi_1, \Pi_2, ..., \Pi_N \}$ (equivalent to J&M's explicit start state; how ?)

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

Things to know about HMMs

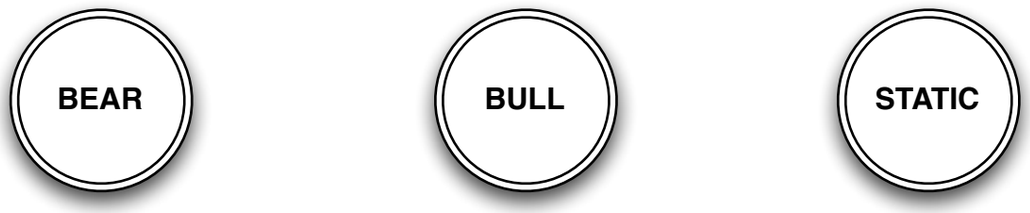
- The (first-order) Markov assumption holds
- The probability of an output symbol depends *only* on the state generating it

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

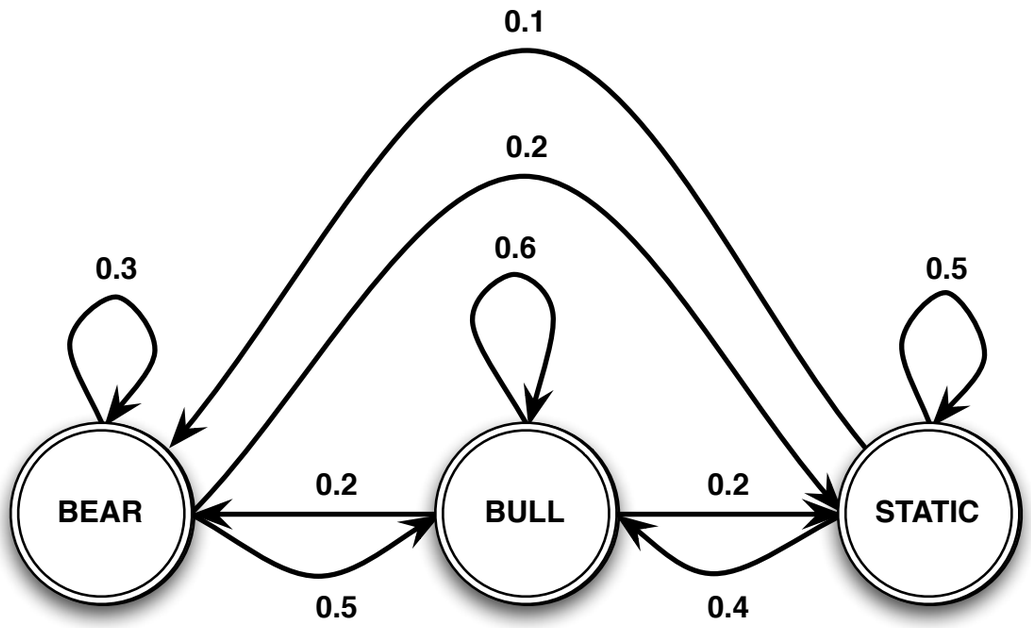
- The number of states (N) does not have to equal the number of observations (T)

Stock Market HMM

States ?



Stock Market HMM

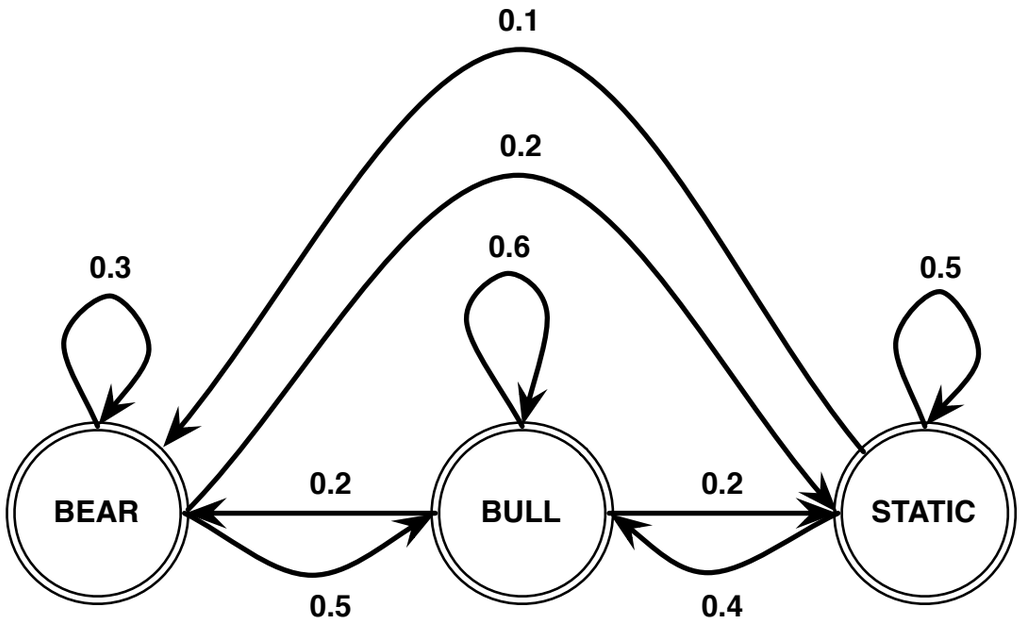


States ✓

Transitions ?

Valid ?

Stock Market HMM



States ✓

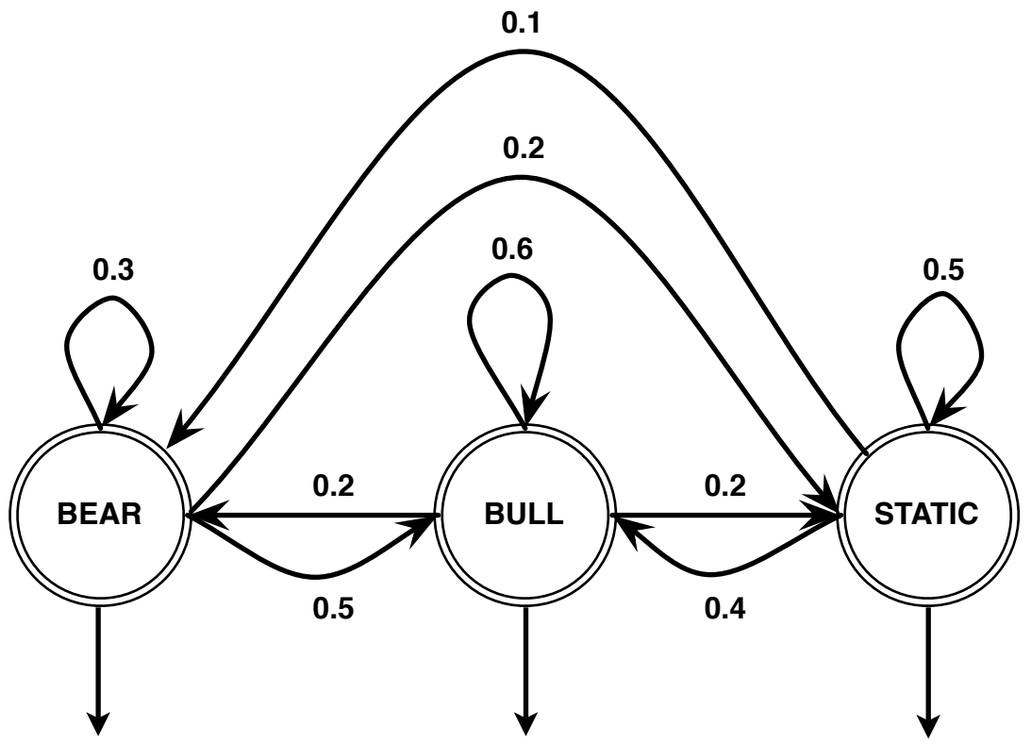
Transitions ✓

Valid ✓

Vocabulary ?

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

Stock Market HMM



States ✓

Transitions ✓

Valid ✓

Vocabulary ✓

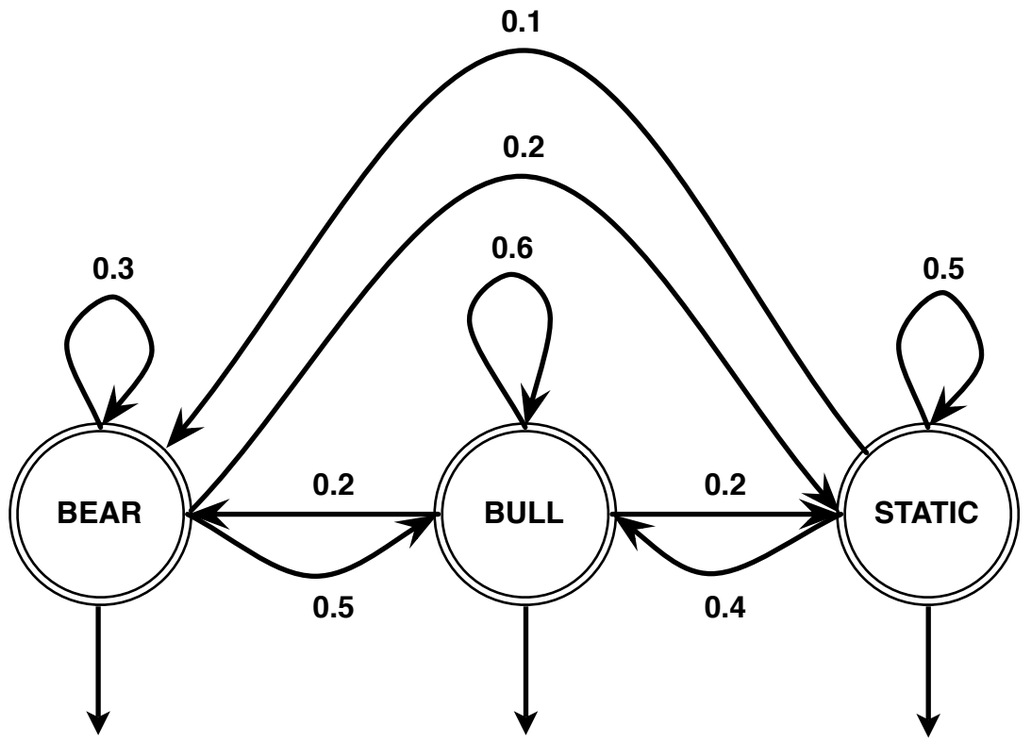
Emissions ?

Valid ?

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

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

Stock Market HMM



States ✓

Transitions ✓

Valid ✓

Vocabulary ✓

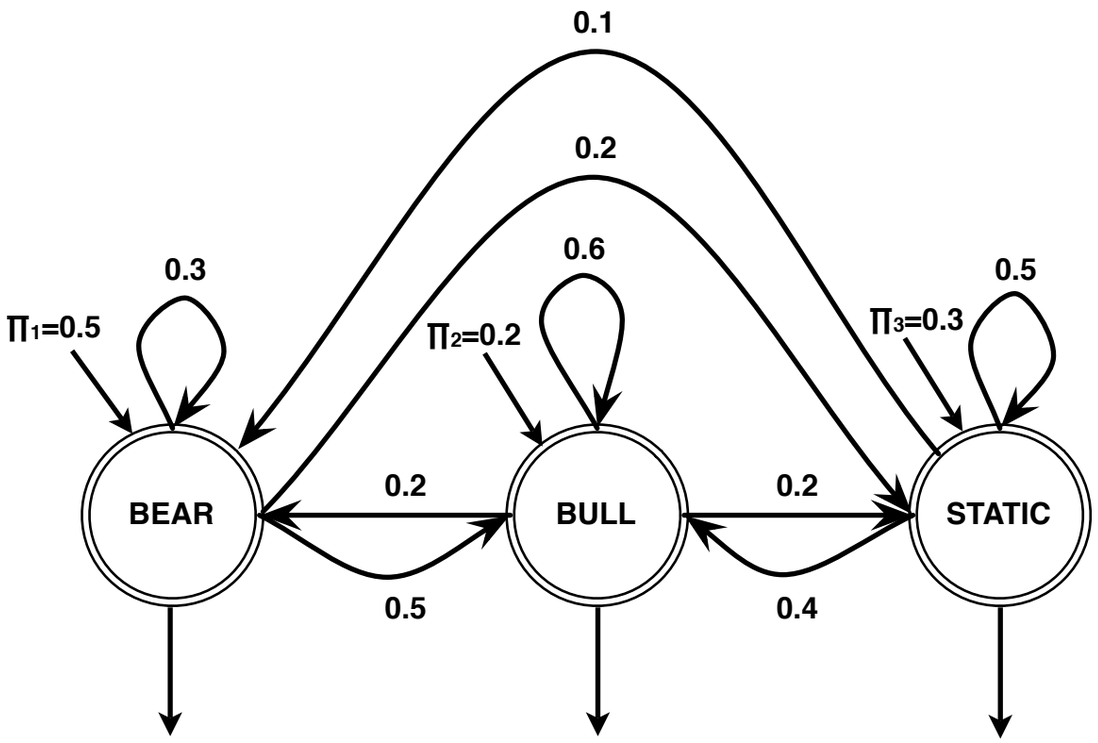
Emissions ✓

Valid ✓

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

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

Stock Market HMM



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

States ✓

Transitions ✓

Valid ✓

Vocabulary ✓

Emissions ✓

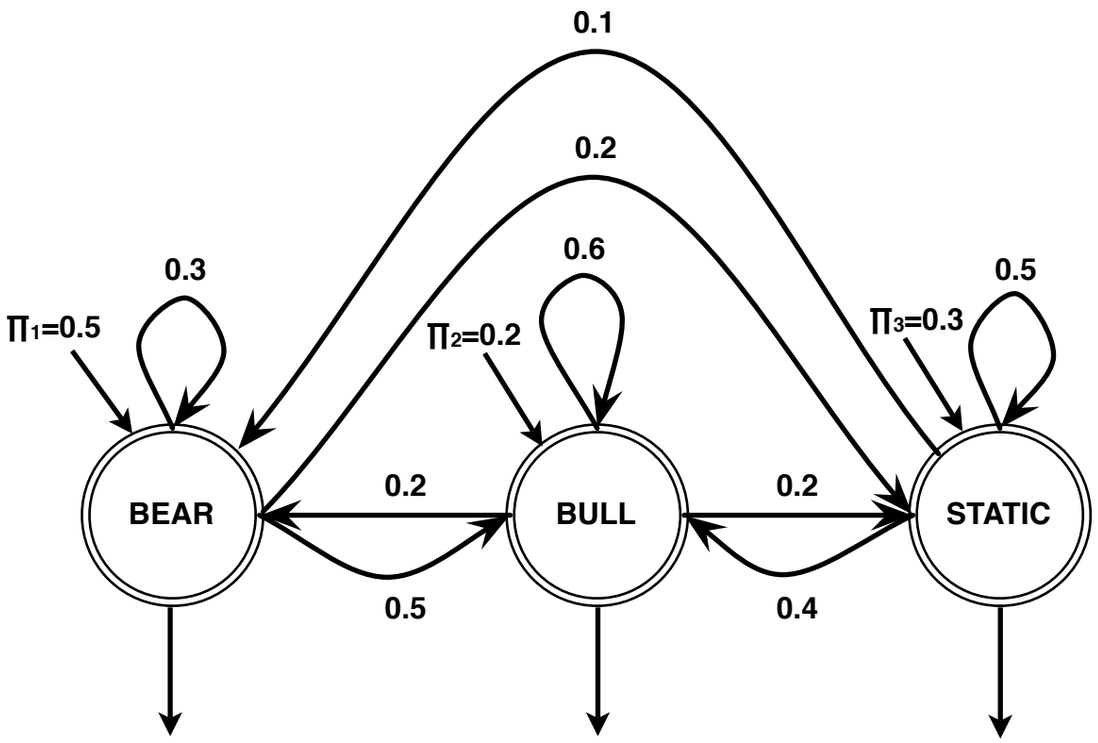
Valid ✓

Priors ?

Valid ?

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

Stock Market HMM



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

States ✓

Transitions ✓

Valid ✓

Vocabulary ✓

Emissions ✓

Valid ✓

Priors ✓

Valid ✓

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

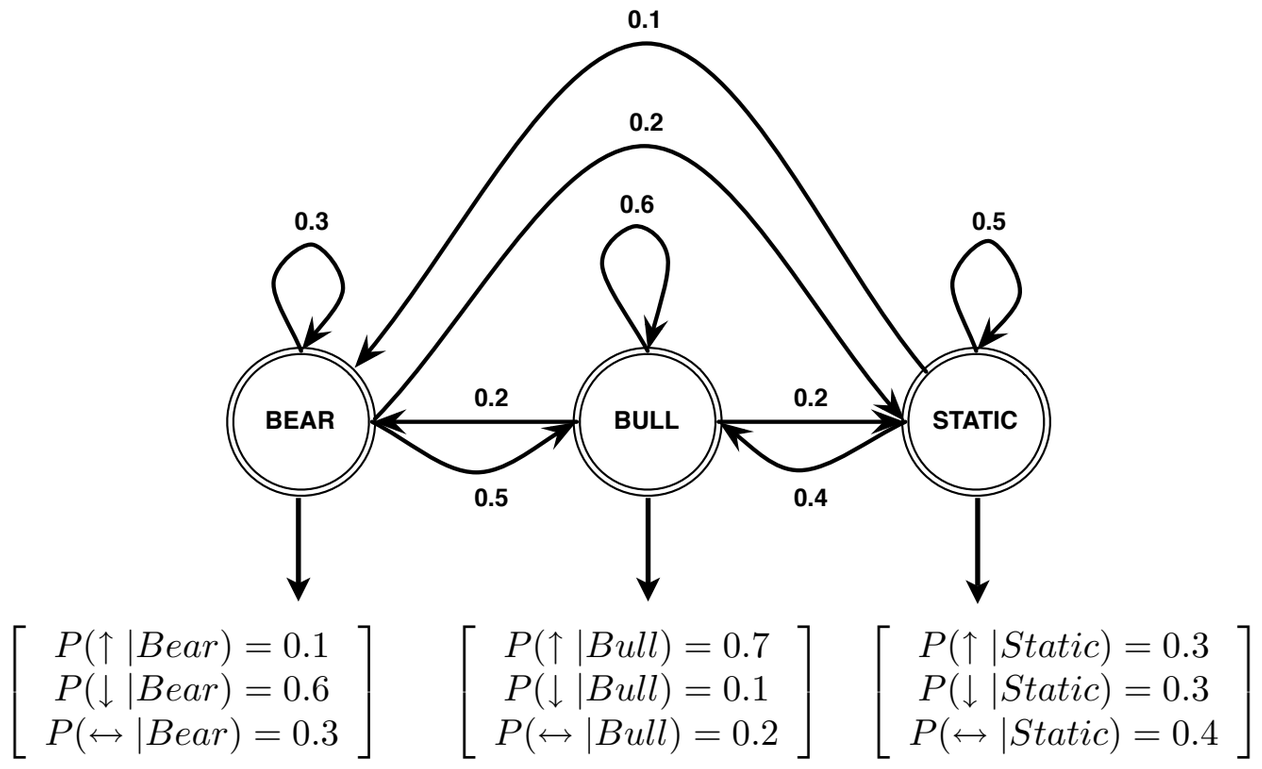
Applying HMMs

- 3 problems to solve before HMMs can be useful
  - Given an HMM $\lambda = (A, B, \prod)$, and a sequence of observed events O, find $P(O|\lambda)$ [***Likelihood***]
  - Given an HMM $\lambda = (A, B, \prod)$, and an observation sequence O, find the most likely (hidden) state sequence [***Decoding***]
  - Given an observation sequence and the set of states Q in λ, compute the parameters A an B. [***Training***]

Computing Likelihood

t : 1 2 3 4 5 6

O : ↑ ↓ ↔ ↑ ↓ ↔



λ_{stock}

Assuming λ_{stock} models the stock market, how likely is it that on day 1, the market is up, on day 2, it's down etc. ?

Markov Chain ?

Computing Likelihood

- Easy !
- Sum over all possible ways in which we could generate O from λ

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

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

Takes exponential ($\propto N^T$) time to compute !

Right idea, wrong algorithm !

Computing Likelihood

- What are we doing wrong ?
- State sequences may have a lot of overlap
  - We are recomputing the shared bits every time
  - Need to store intermediate computation results somehow so that they can be used
  - Requires a *Dynamic Programming* algorithm

Forward Algorithm

- Use an N x T *trellis* or chart [α_{tj}]
- 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 = $\sum_N$ extensions of all paths from other cells

$$\alpha_t(j) = \sum_{i=1} \alpha_{t-1}(i) a_{ij} b_j(o_t)$$

  - $\alpha_{t-1}(i)$: 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*
- $P(O|\lambda) = \sum_i \alpha_T(i)$
- Polynomial time ($\propto N^2T$)

Forward Algorithm

- Formal Definition
- Initialization

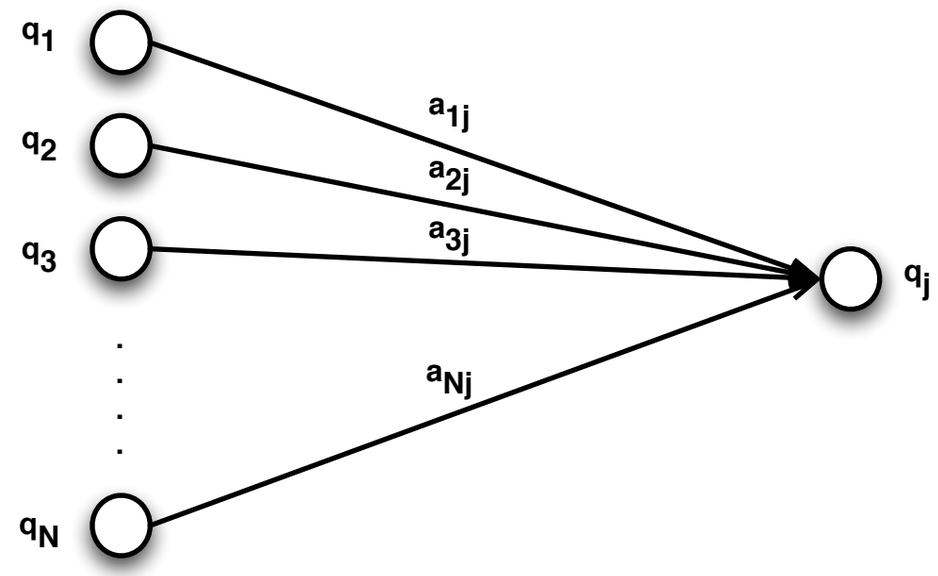
$$\alpha_1(j) = \pi_j b_j(o_1), 1 \leq j \leq N$$

- 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

Static

state

Bear

$O = \uparrow \downarrow \uparrow$

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

Bull

↑ t=1

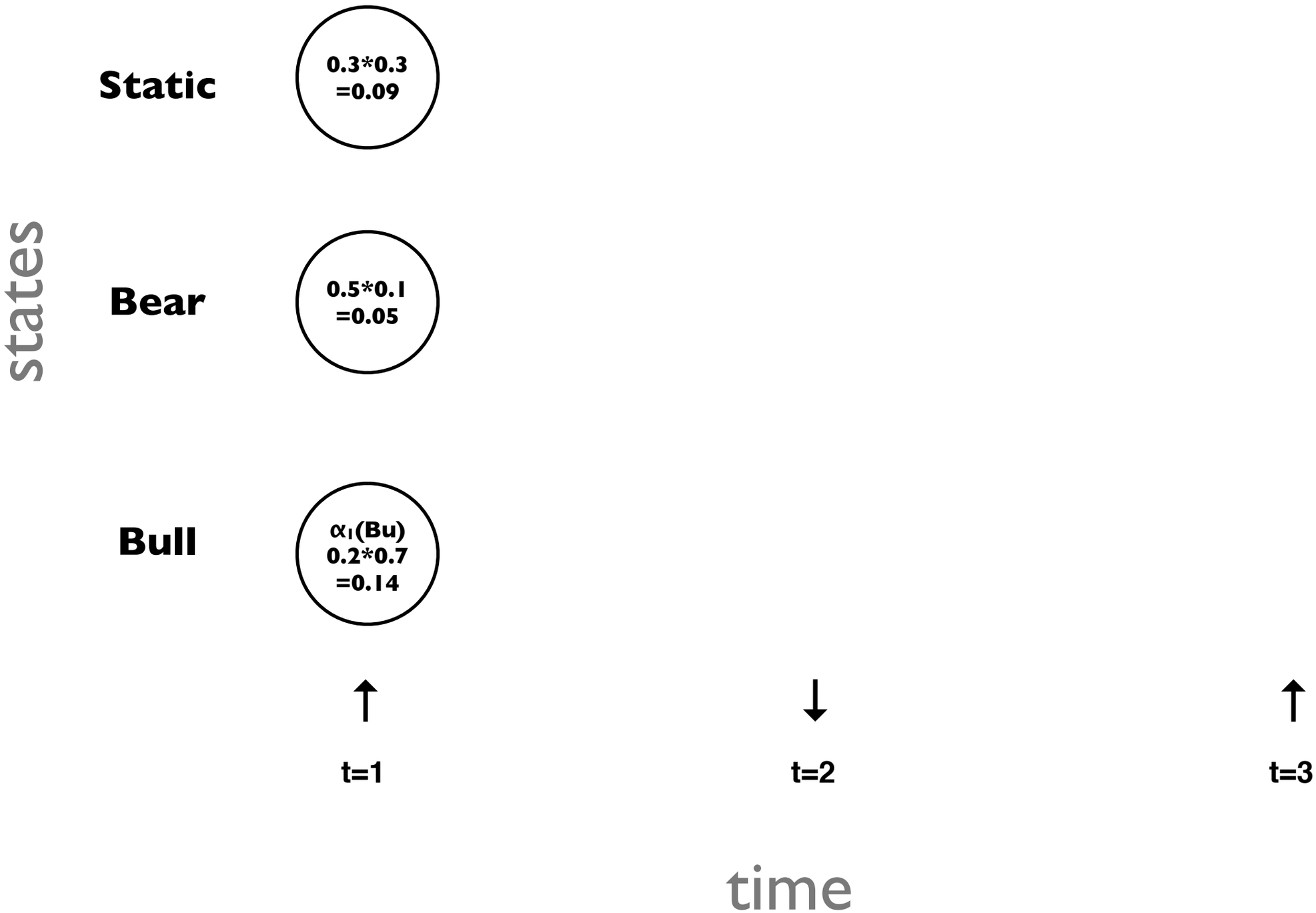
↓ t=2

↑ t=3

time

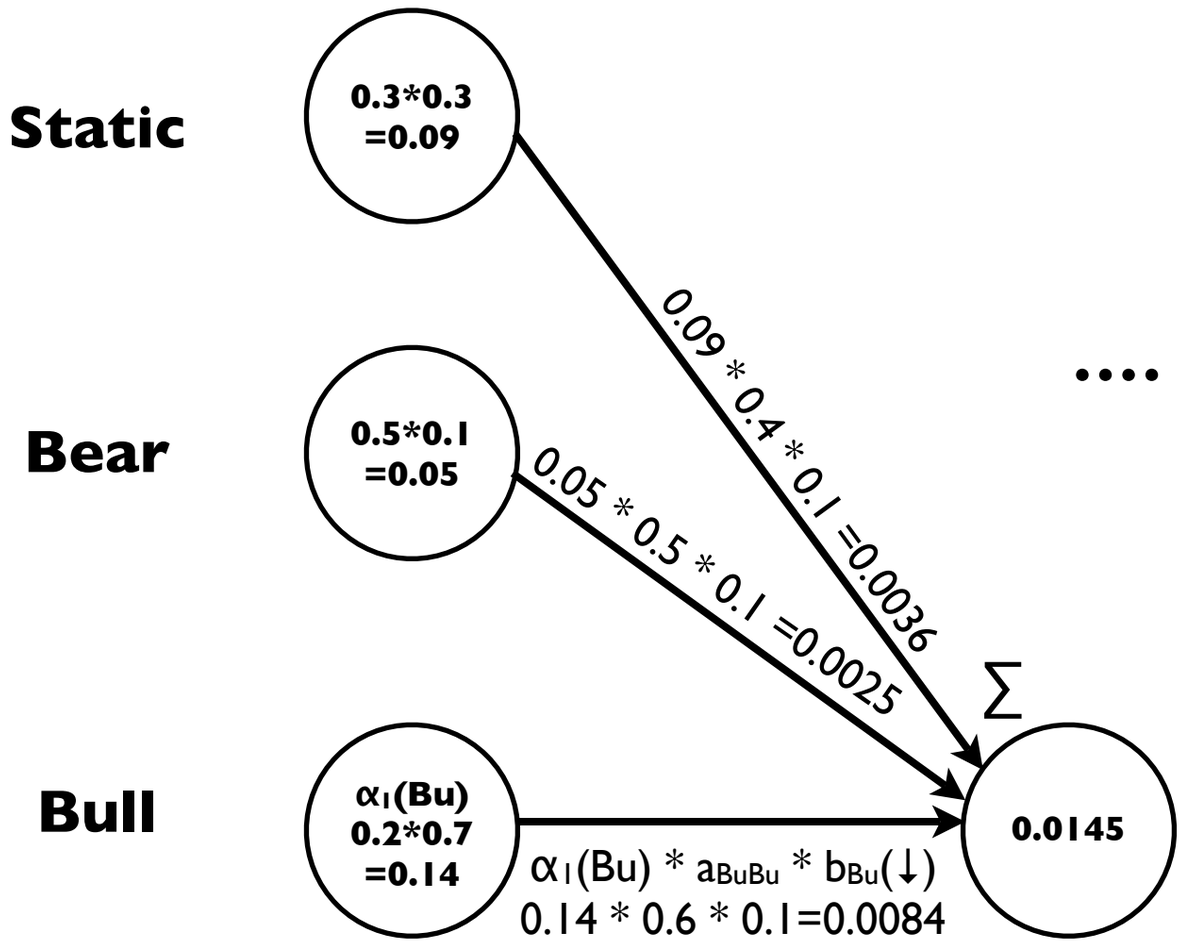
Forward Algorithm

Initialization



Forward Algorithm

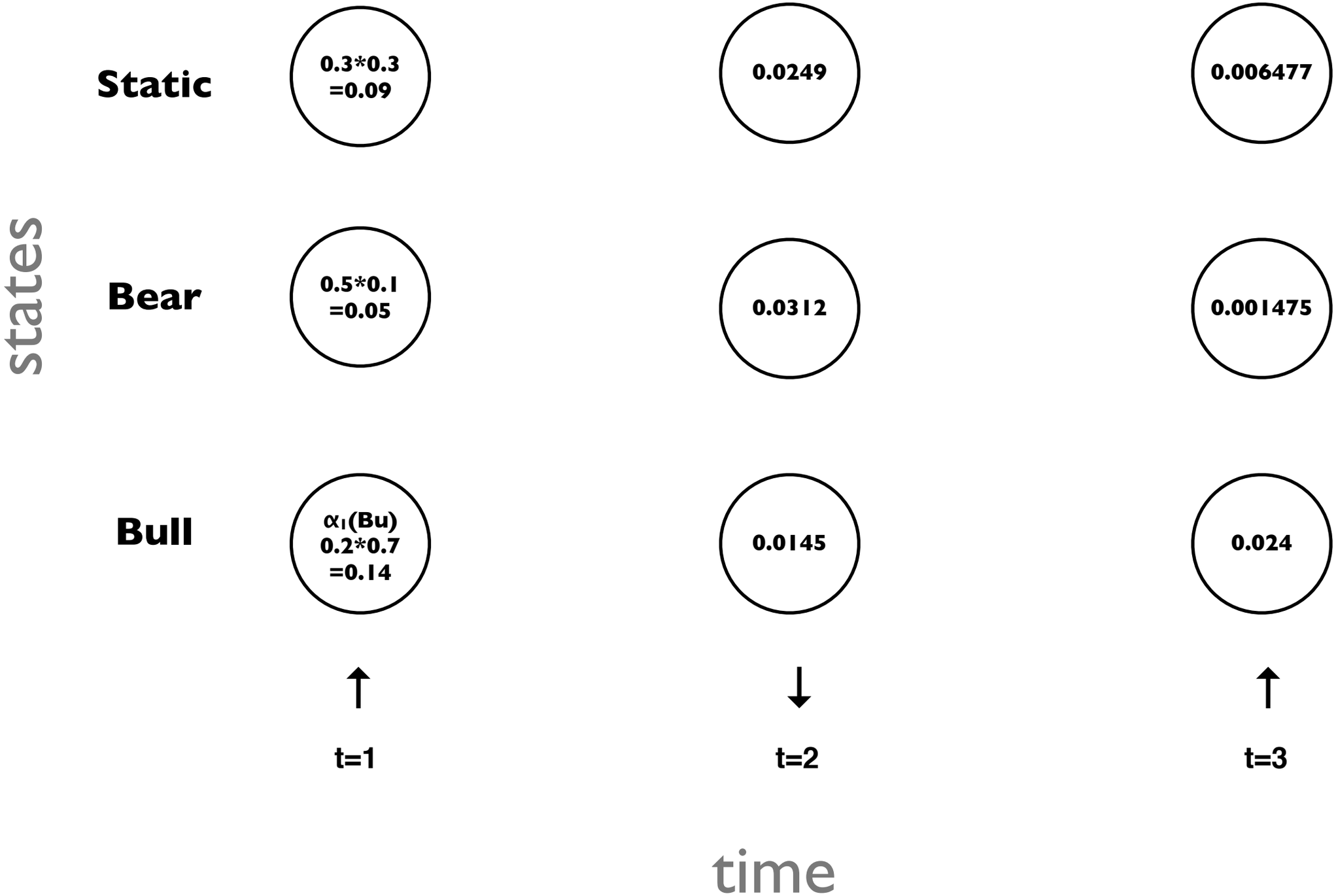
Recursion



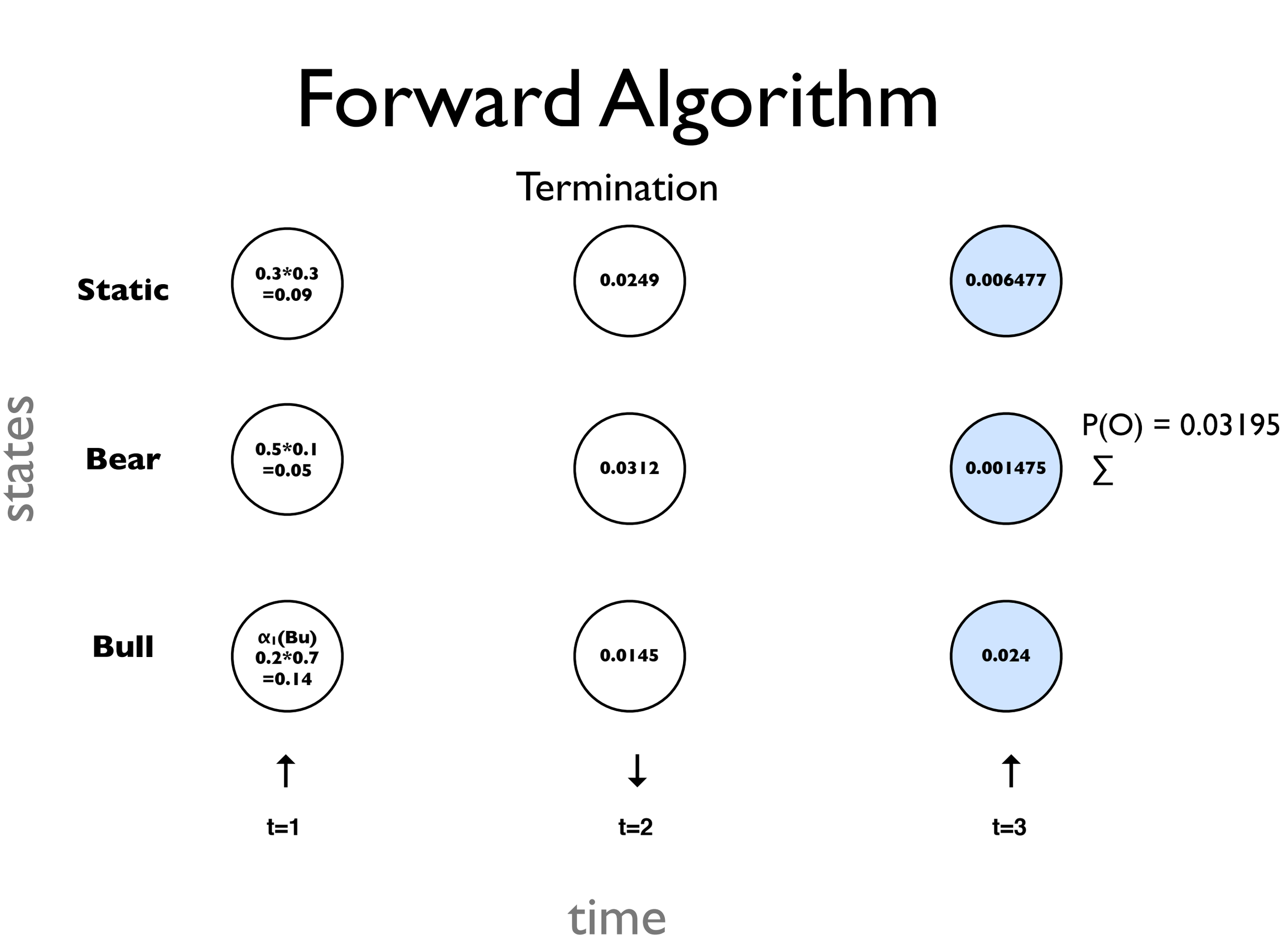
.... and so on

Forward Algorithm

Recursion



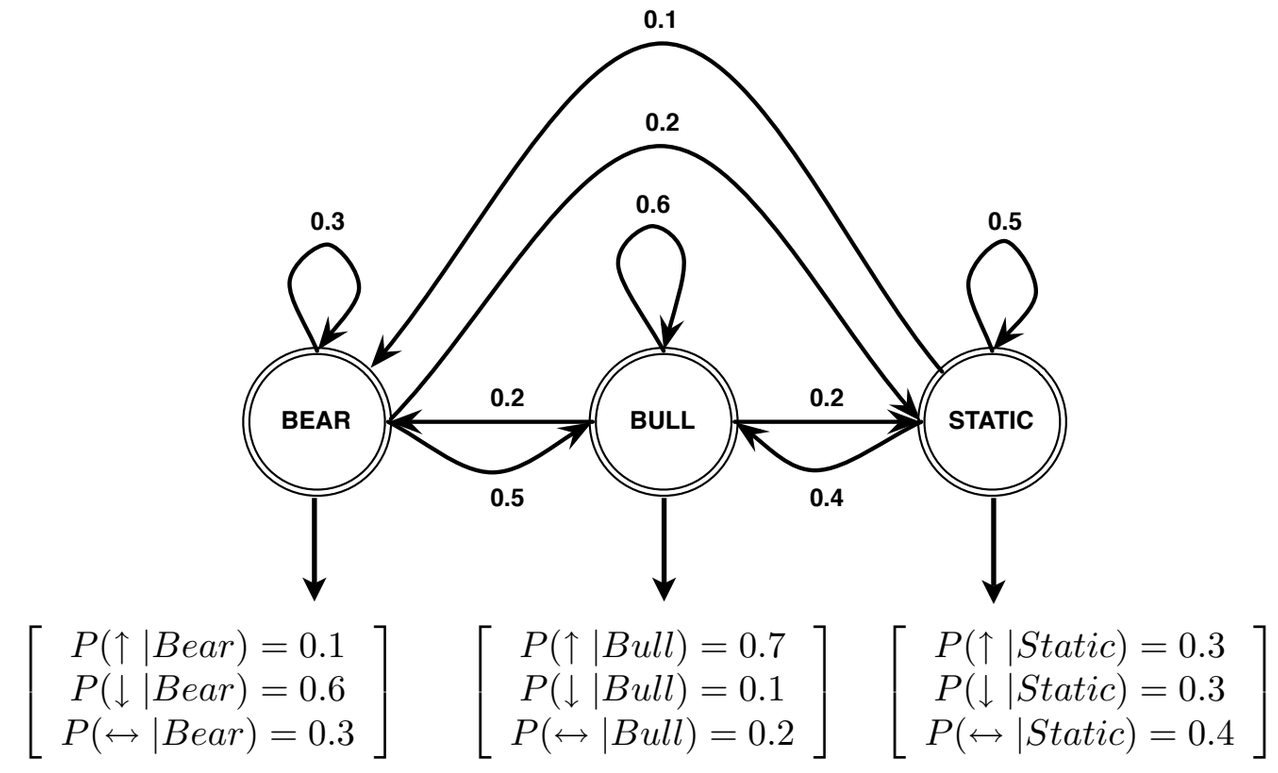
Forward Algorithm



Decoding

t : 1 2 3 4 5 6

O : ↑ ↓ ↔ ↑ ↓ ↔



λ_{stock}

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
- Easy !
  - For each possible hidden state sequence, compute P(O) using forward algorithm
  - Pick the one that gives the highest P(O)
  - Will this give the right answer ?
  - Is it practical ?

Viterbi Algorithm

- Another dynamic programming algorithm
- Same idea as the forward algorithm
  - Store intermediate computation results in a trellis
  - Build new cells from existing cells
- Efficient (polynomial vs. exponential)

Viterbi Algorithm

- Use an N x T *trellis* [v_{tj}]
- v_{tj} or $v_t(j)$ = P(in state *j* after seeing *t* observations & passing through the most likely state sequence)
  = $p(q_1, q_2, ..., q_{t-1}, q_t=j, o_1, o_2, ... o_t)$
- Each cell = extension of most likely path from other cells

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

  - $\alpha_{t-1}(i)$: 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 = $\max_i v_T(i)$

Viterbi Algorithm

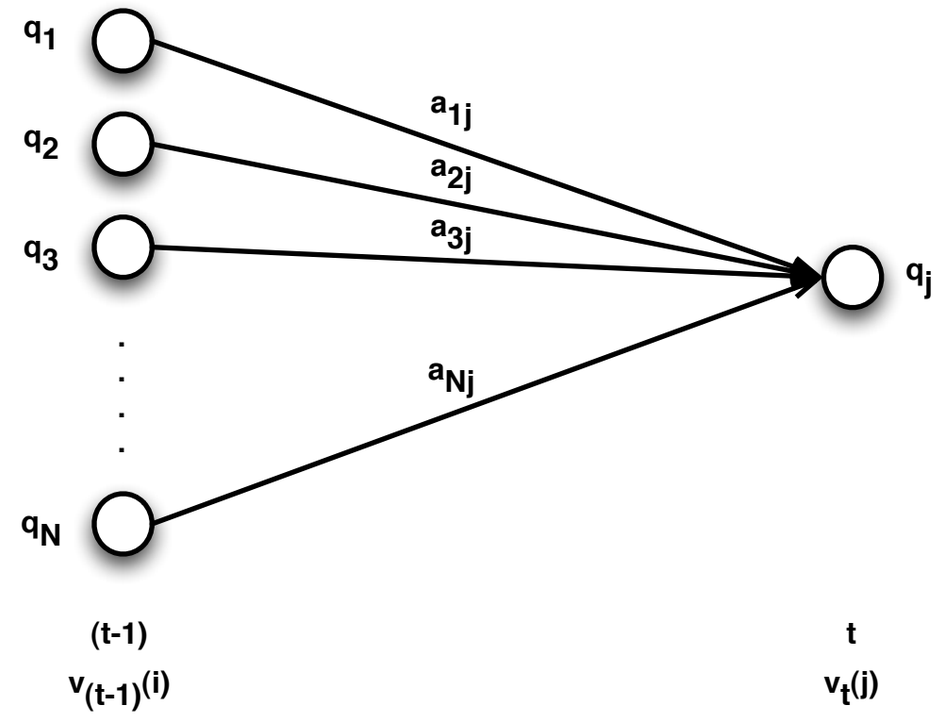
- Maximization instead of summation over previous paths
- This algorithm is still missing something !
- Unlike forward alg., we need something else in addition to the probability !
  - Need to keep track which previous cell we chose
  - At the end, follow the chain of backpointers and we have the most likely state sequence too !
  - $q_T^* = \text{argmax}_i\, v_T(i)$; q_t^* = the state q_{t+1}^* points to

Viterbi Algorithm

- Formal Definition
- Initialization

$$v_1(j) = \pi_i b_i(o_1); 1 \leq i \leq N$$

$$BT_1(i) = 0$$



- Recursion

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

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

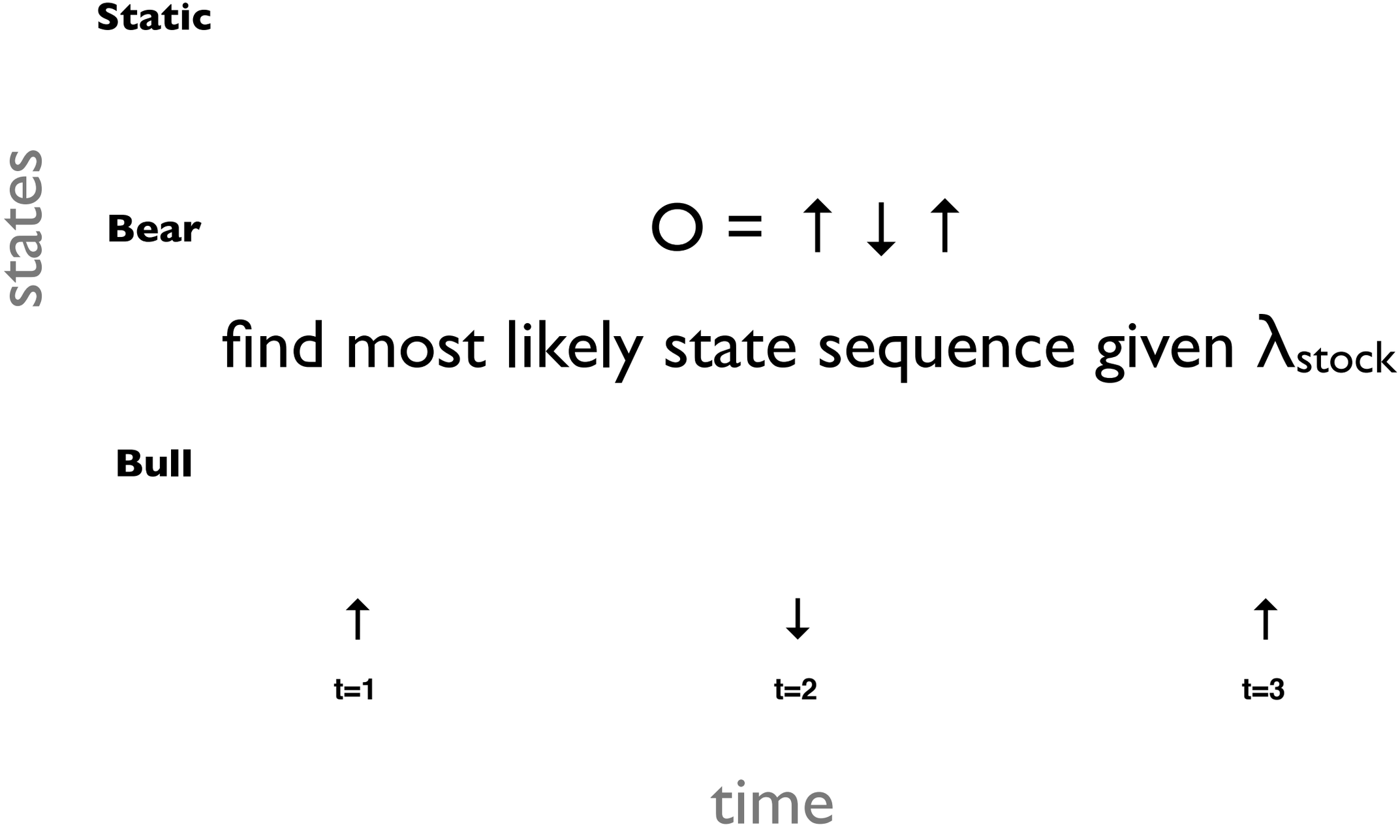
Why no b() ?

- Termination

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

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

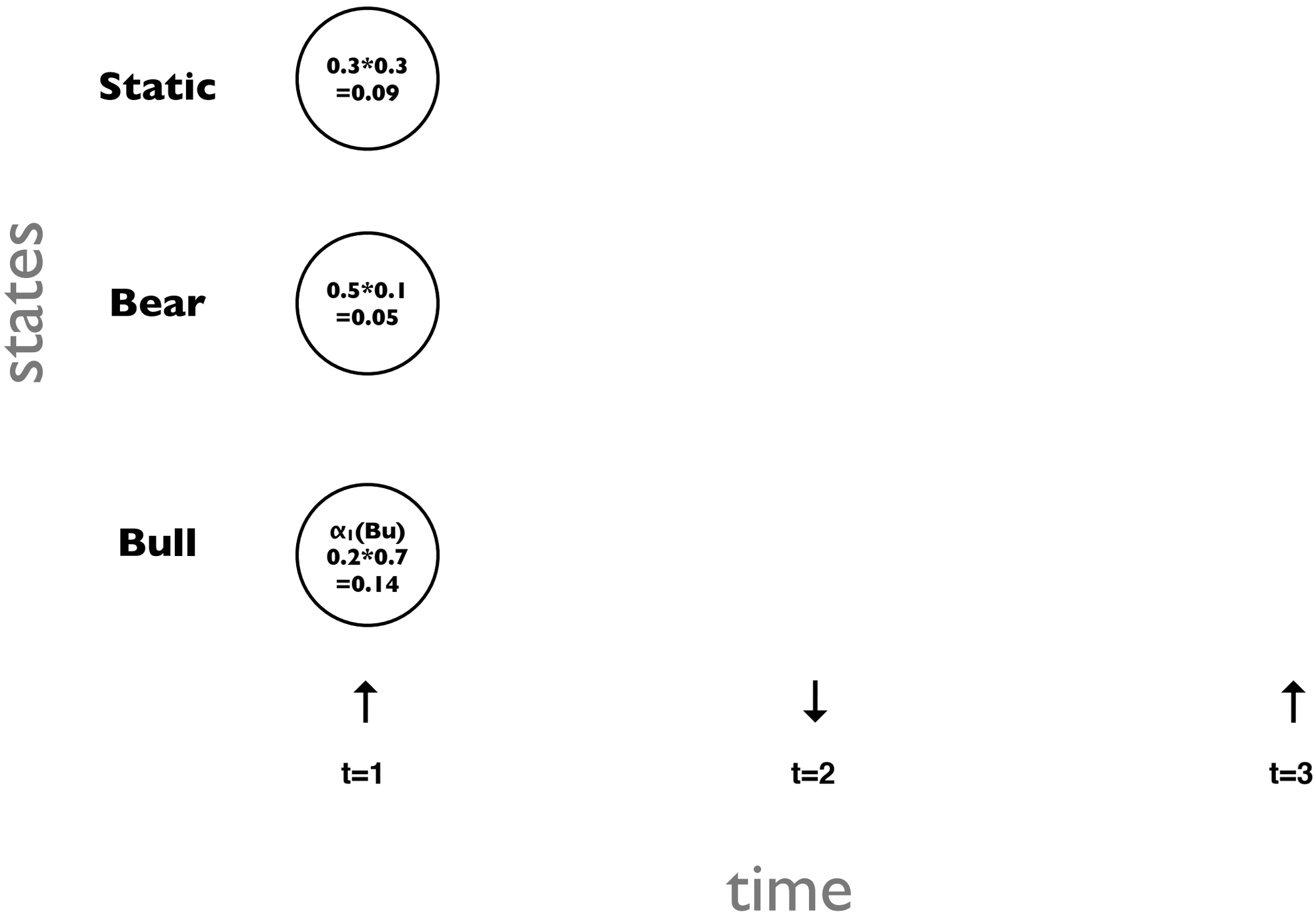
Viterbi Algorithm



find most likely state sequence given λ_{stock}

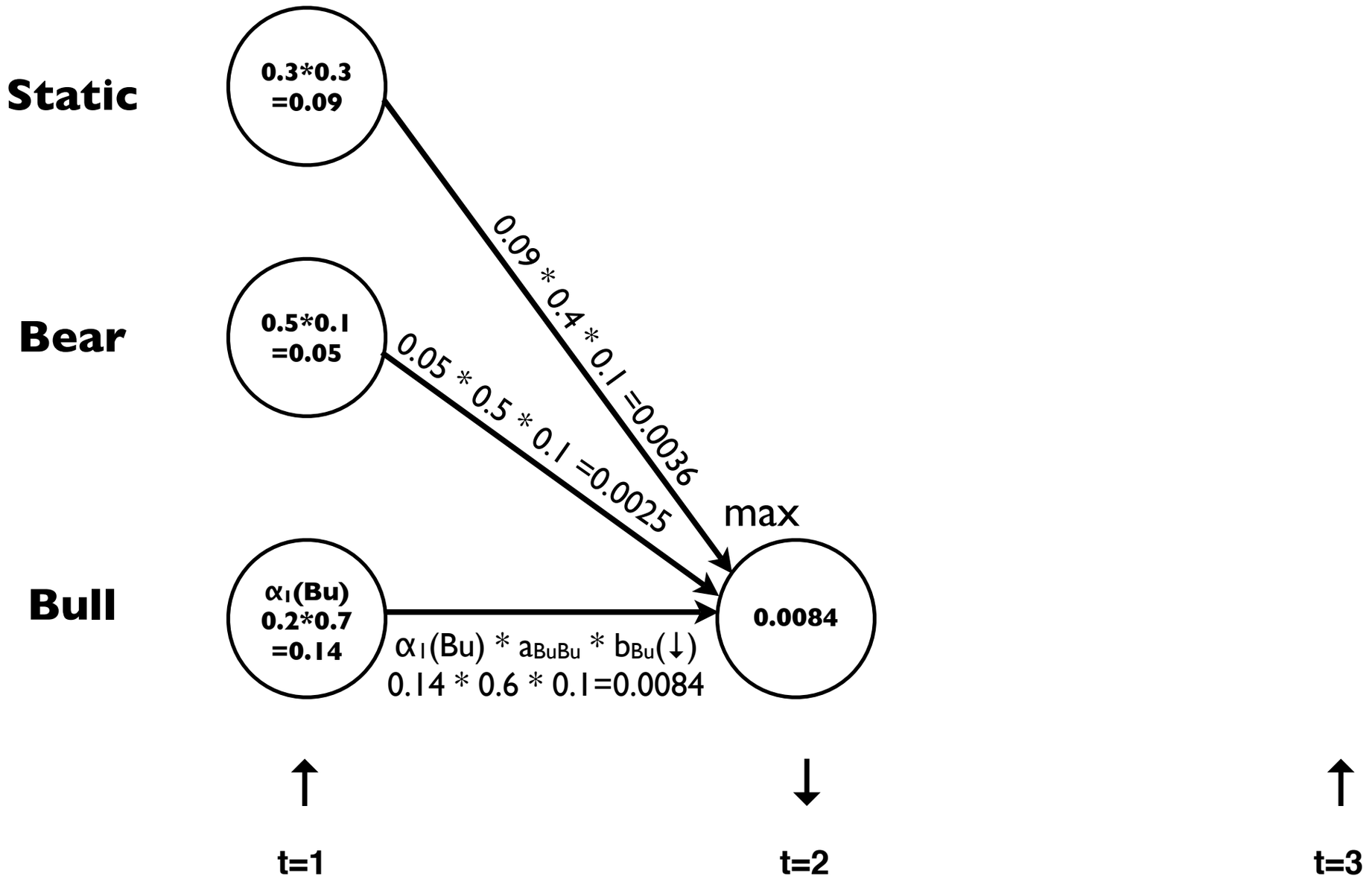
Viterbi Algorithm

Initialization



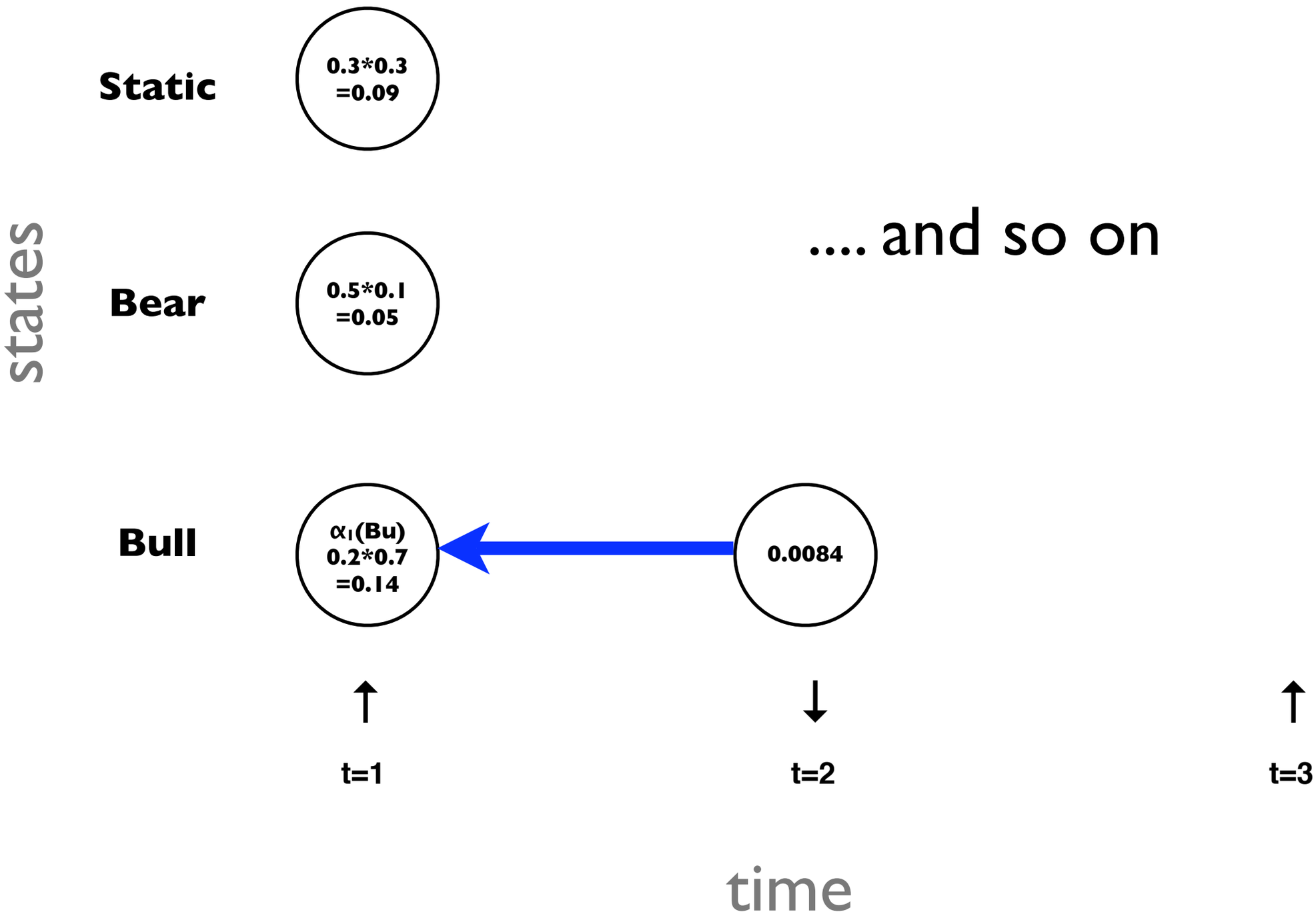
Viterbi Algorithm

Recursion



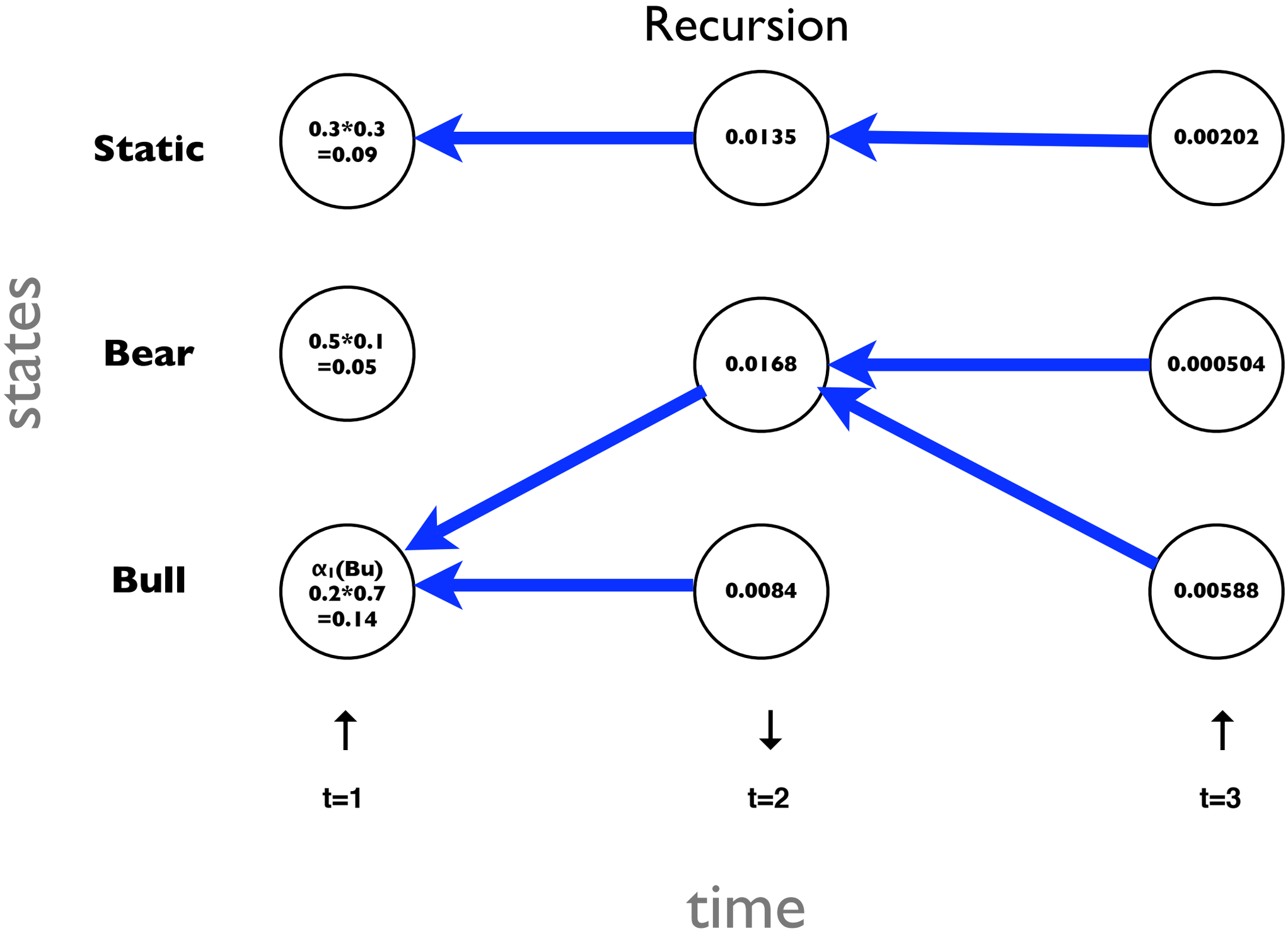
Viterbi Algorithm

Recursion

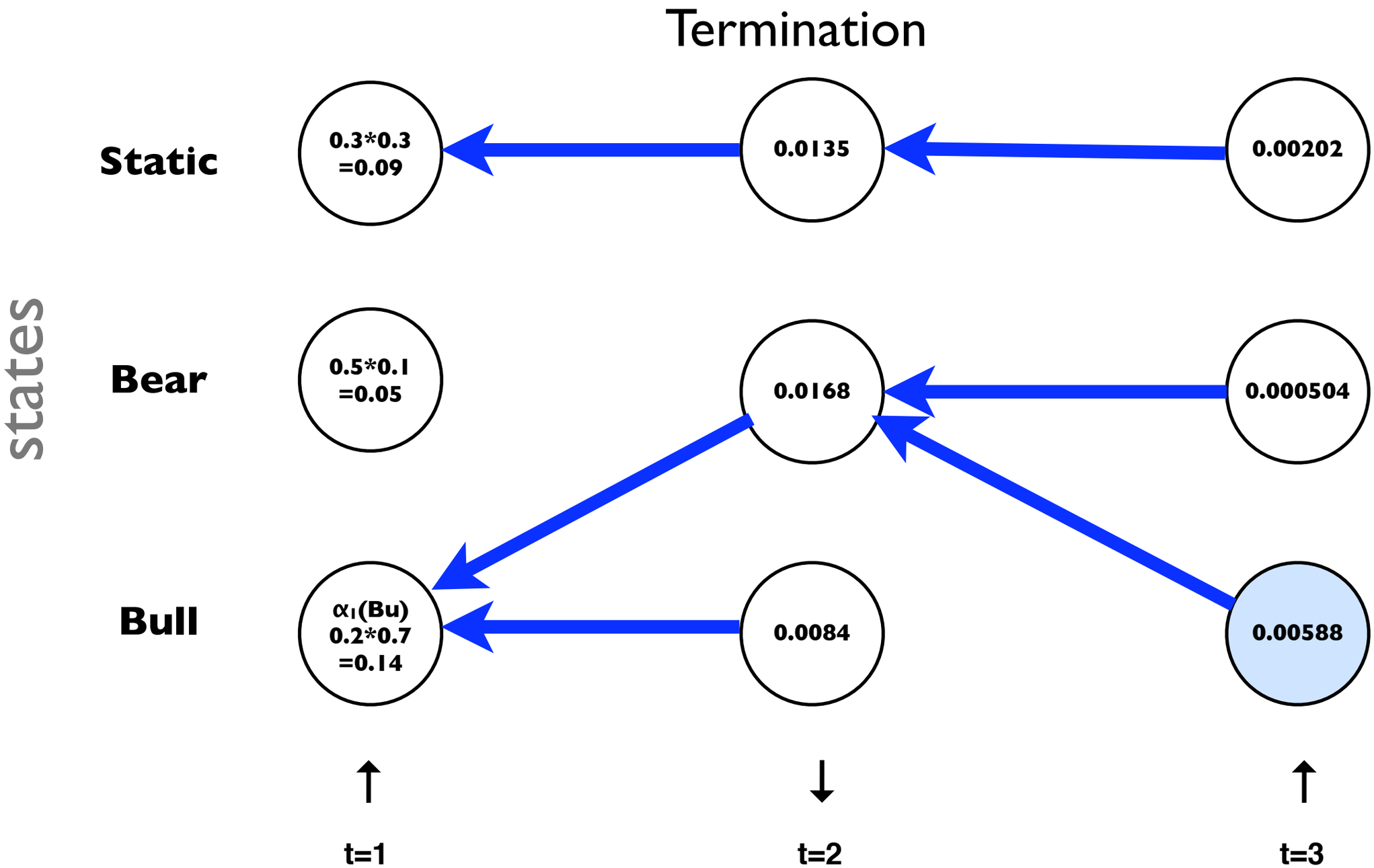


.... and so on

Viterbi Algorithm

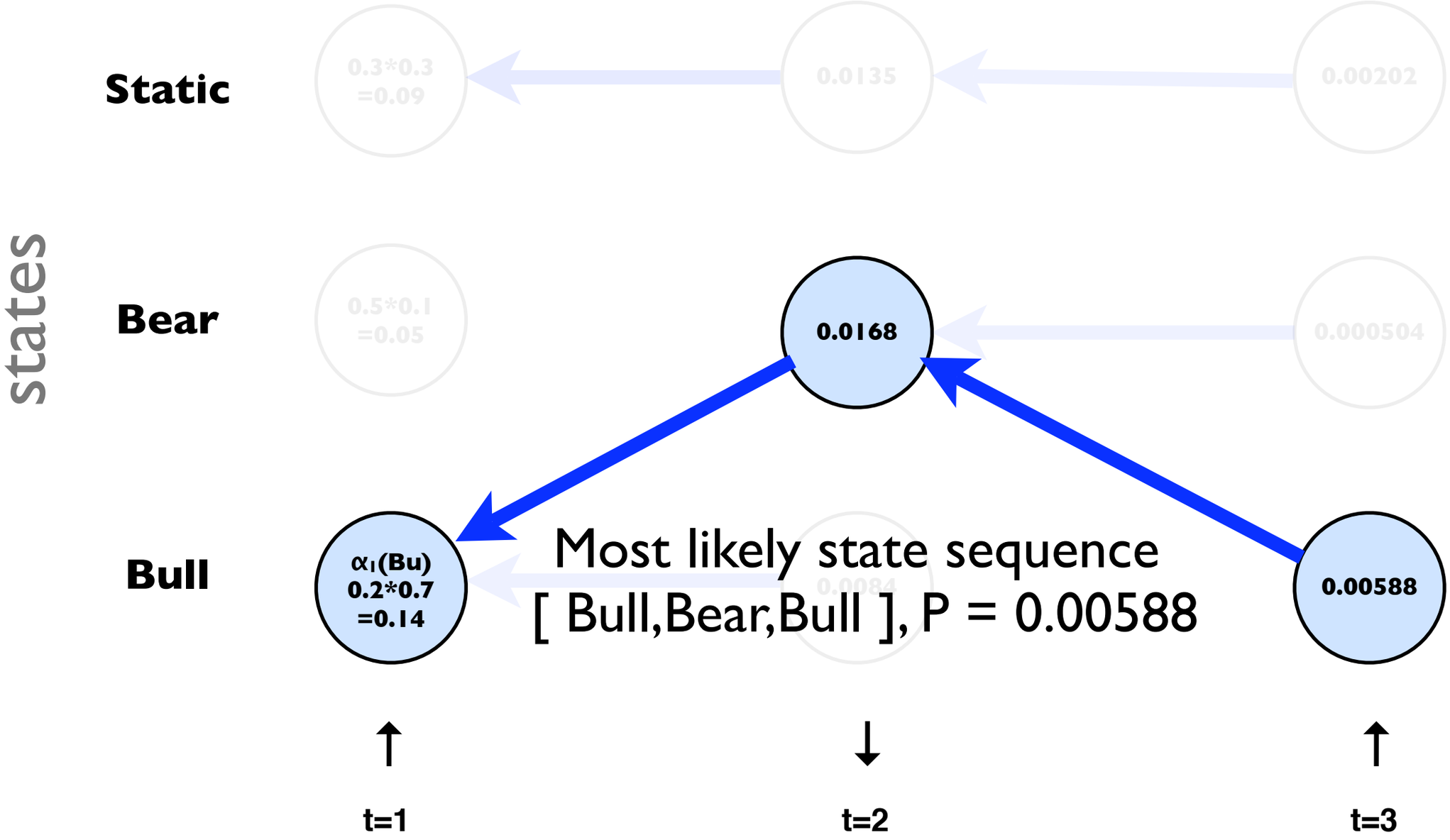


Viterbi Algorithm



Viterbi Algorithm

Termination



HMM-based POS Tagging

Modeling the problem

- What should the HMM look like ?
  - States: Part-of-Speech Tags (t_1, t_2, ..., t_N)
  - Output symbols: Words (w_1, w_2, ..., $w_{|V|}$)
    - Can an HMM find the most likely tags for a given sentence ?
      - Yes ! Viterbi Decoding
    - The HMM machinery gives us (almost) everything we need to solve the problem

HMM Training

- *almost* everything ?
- Before HMMs can decode, they must be *trained*, i.e., (A, B, ∏) must be computed
- Two training methods:
  - *Supervised training*: Use already tagged words; count stuff; estimate parameters
  - *Unsupervised training*: Use untagged words; bootstrap parameter estimates; improve estimates iteratively

Supervised Training

- We have *training data*, i.e., thousands of sentences with their words already tagged
- Given this data, we already have the set of states and symbols
- Next, compute *Maximum Likelihood Estimates (*MLEs*)* for the various parameters
- Those estimates of the parameters that 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
  - 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 $P(w_i|t_i) = C(w_i, t_i) / C(t_i)$, 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 $P(q_1 = t_i) = \prod_i = C(t_i)/N$, from the tagged data
  - For $\prod_{NN}$, count the number of times NN occurs and divide by the total number of tags in the data.

Unsupervised Training

- No labeled/tagged training data
- No way to compute MLEs directly
- 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)*

*Dempster & Laird (1977)

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) = 2μ, P(D) = 1/2 - 3μ
- Number of 'A's observed = 'a', 'b' number of 'B's etc.
- Compute MLE of μ given 'a', 'b', 'c' and 'd'

Original example from Andrew Moore's slides
http://www.autonlab.org/tutorials/gmm.html

Motivating Example

- Recall the definition of MLE
  "*.... maximizes likelihood of data given the model.*"
- P(Data | Model) = P(a,b,c,d|μ) = $(1/2)^a(\mu)^b(2\mu)^c(1/2 - 3\mu)^d$
- L = log-likelihood = log P(a,b,c,d|μ)
  = a log(1/2) + b log μ + c log 2μ + d log(1/2-3μ)
- 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

- P(A) = 1/2, P(B) = μ, P(C) = 2μ, P(D) = 1/2 - 3μ
- Number of ‘A’s and ‘B’s = h, c ‘C’s and d ‘D’s
- Part of the information is hidden

- Can we compute the MLE for μ now ?
- If we knew ‘b’ (and hence ‘a’), we could compute the MLE for μ. But we need to know 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

$b_t = [\mu_t/(1/2 + \mu_t)]h$

[**E-step**: Compute expected value of b given μ_t]

$\mu_t = (b_t + c)/6(b_t + c + d)$

[**M-step**: Compute MLE of μ given b_t]

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 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; Details in CL II (next sem.)
- Hidden information (the state sequence)
- Model Parameters: A, B & $\prod$
- Introduce two new observation statistics:
  - Number of transitions from q_i to q_j (ξ)
  - Number of times in state q_i (γ)
- The EM algorithm should now apply perfectly

Applying EM to HMMs

Start with initial guesses for A, B and Π

t = 1; Repeat

E-step: Compute expected values of ξ, γ using A_t, B_t, Π_t

M-step: Compute MLE of A, B and Π using ξ_t, γ_t

t = t + 1

Until some convergence criterion is met

The Forward-Backward Algorithm

Readings for next time

- New chapter 12 (Sections 12.1 - 12.6)

- New chapter 15