Data-Intensive Information Processing Applications — Session #8

Hidden Markov Models & EM

Nitin Madnani
University of Maryland

Tuesday, March 30, 2010

Source: Wikipedia (Japanese rock garden)

Today's Agenda

- Need to cover *lots* of background material
  - Introduction to Statistical Models
  - Hidden Markov Models
  - Part of Speech Tagging
  - Applying HMMs to POS tagging
  - Expectation-Maximization (EM) Algorithm
- Now on to the Map Reduce stuff
  - Training HMMs using MapReduce
    - Supervised training of HMMs
    - Rough conceptual sketch of unsupervised training using EM

Introduction to statistical models

- Until the 1990s, text processing relied on *rule-based* systems
- Advantages
  - More predictable
  - Easy to understand
  - Easy to identify errors and fix them
- Disadvantages
  - Extremely labor-intensive to create
  - Not robust to out of domain input
  - No partial output or analysis when failure occurs

Introduction to statistical models

- A better strategy is to use data-driven methods
- Basic idea: learn from a large corpus of examples of what we wish to model (*Training Data*)
- Advantages
  - More robust to the complexities of real-world input
  - Creating training data is usually cheaper than creating rules
    - Even easier today thanks to Amazon Mechanical Turk
    - Data may already exist for independent reasons
- Disadvantages
  - Systems often behave differently compared to expectations
  - Hard to understand the reasons for errors or debug errors

Introduction to statistical models

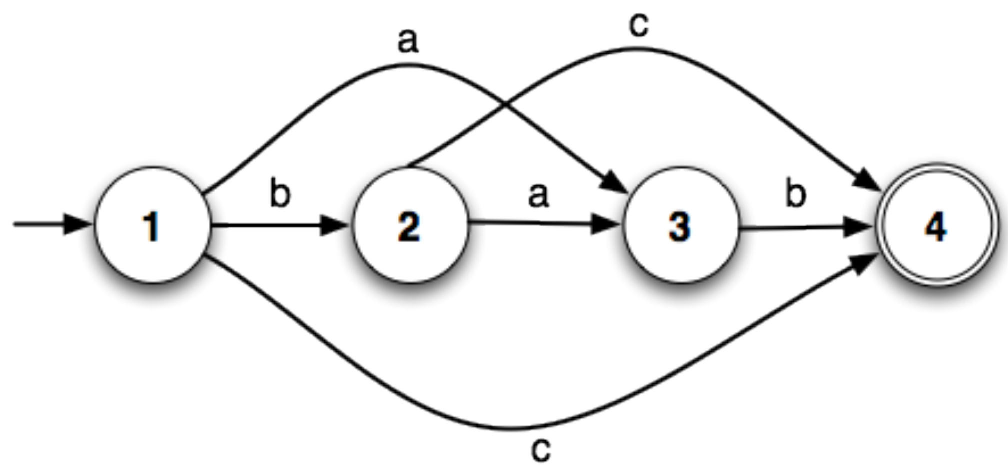
- Learning from training data usually means estimating the parameters of the statistical model
- Estimation usually carried out via machine learning
- Two kinds of machine learning algorithms
- Supervised learning
  - Training data consists of the inputs and respective outputs (labels)
  - Labels are usually created via expert annotation (expensive)
  - Difficult to annotate when predicting more complex outputs
- Unsupervised learning
  - Training data just consists of inputs. No labels.
  - One example of such an algorithm: Expectation Maximization

Hidden Markov Models (HMMs)

A very useful and popular statistical model

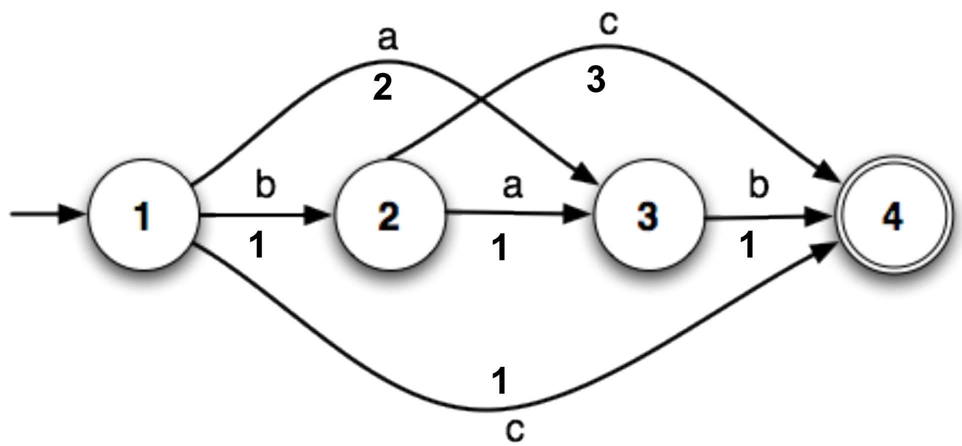
Finite State Machines

- What do we need to specify an FSM formally ?
  - Finite number of states
  - Transitions
  - Input alphabet
  - Start state
  - Final state(s)



Real World Knowledge

Weighted FSMs



‘a’ is twice as likely to be seen in state 1 as ‘b’ or ‘c’

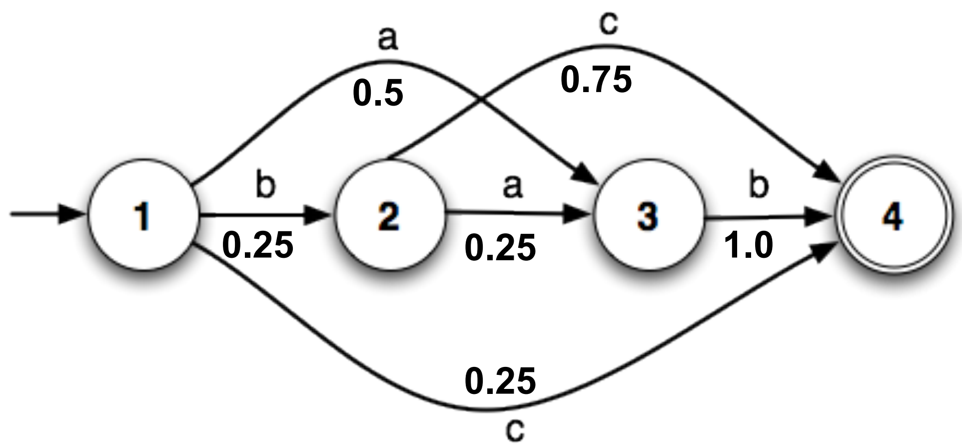
‘c’ is three times as likely to be seen in state 2 as ‘a’

What do we get out of it ?

score(‘ab’) = 2, score(‘bc’) = 3

Real World Knowledge

Probabilistic FSMs



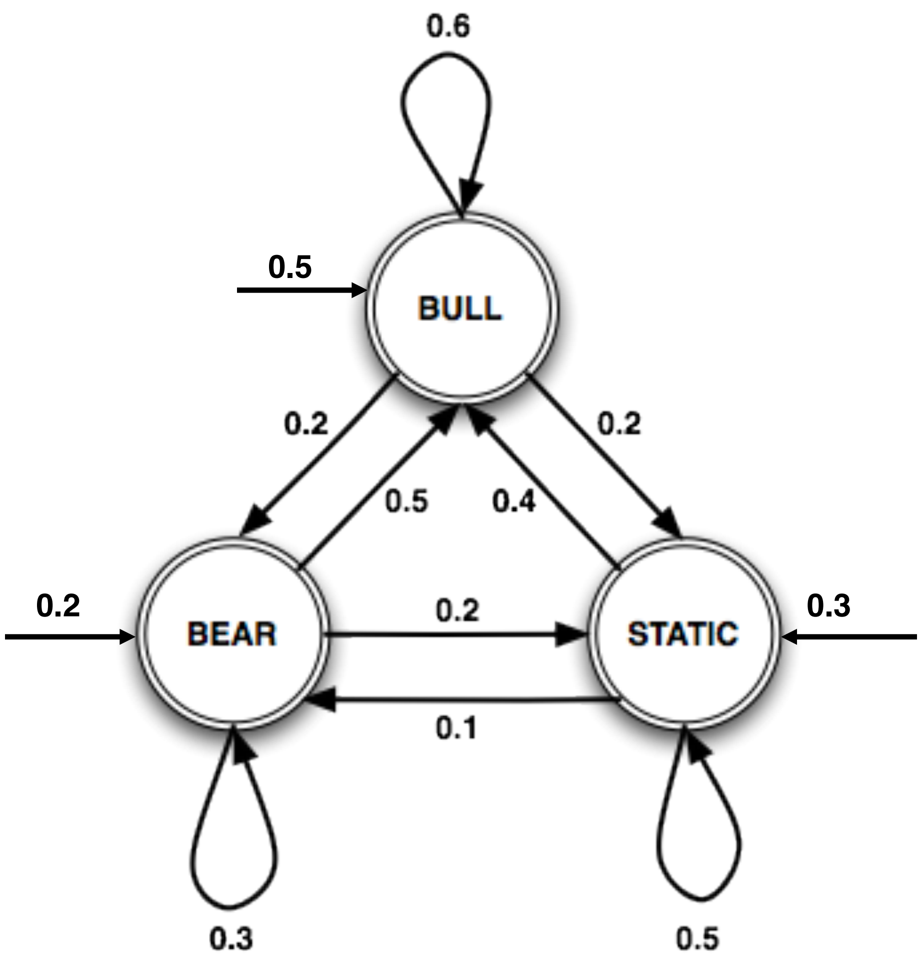
‘a’ is twice as likely to be seen in state 1 as ‘b’ or ‘c’

‘c’ is three times as likely to be seen in state 2 as ‘a’

What do we get out of it ?

P(‘ab’) = 0.50 * 1.00 = 0.5, P(‘bc’) = 0.25 * 0.75 = 0.1875

Markov Chains



- This not a valid prob. FSM!
  - No start states
- Use prior probabilities
- Note that prob. of being in any state ONLY depends on previous state ,i.e., the (1^{st} 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

Bu, Be, S, Be, S, Bu

Bu: Bull Market
Be: Bear Market
S : Static Market

Here's what you actually observe:

Day: 1, 2, 3, 4, 5, 6

↑ ↓ ↔ ↑ ↓ ↔

↑: 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 $\lambda = (A, B, \prod)$ 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 $\prod = \{ \prod_1, \prod_2, ..., \prod_N \}$

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

Things to know about HMMs

- The (first-order) Markov assumption holds

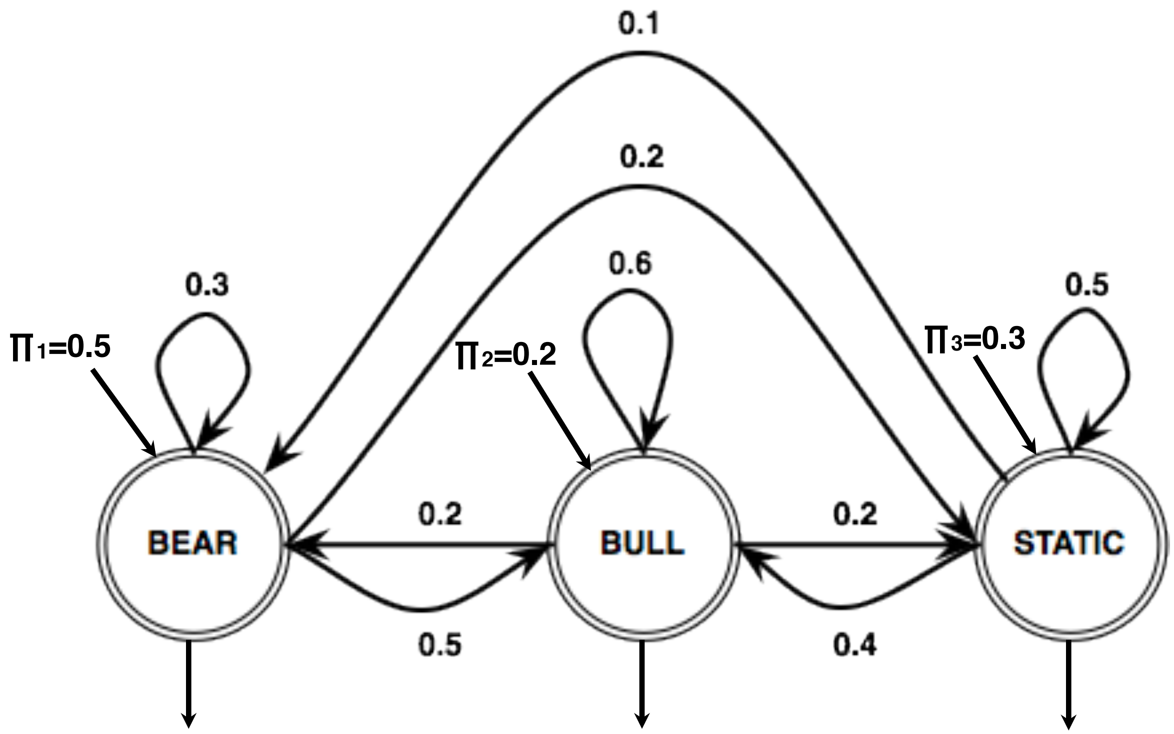
  $$P(q_i|q_1, q_2, \ldots, q_{i-1}) = P(q_i|q_{i-1})$$

- 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



$$\left[\begin{array}{c} P(\uparrow|Bear)=0.1 \\ P(\downarrow|Bear)=0.6 \\ P(\leftrightarrow|Bear)=0.3 \end{array}\right] \quad \left[\begin{array}{c} P(\uparrow|Bull)=0.7 \\ P(\downarrow|Bull)=0.1 \\ P(\leftrightarrow|Bull)=0.2 \end{array}\right] \quad \left[\begin{array}{c} P(\uparrow|Static)=0.3 \\ P(\downarrow|Static)=0.3 \\ P(\leftrightarrow|Static)=0.4 \end{array}\right]$$

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

States ✓

Transitions ✓

Valid ✓

Vocabulary ✓

Emissions ✓

Valid ✓

Priors ✓

Valid ✓

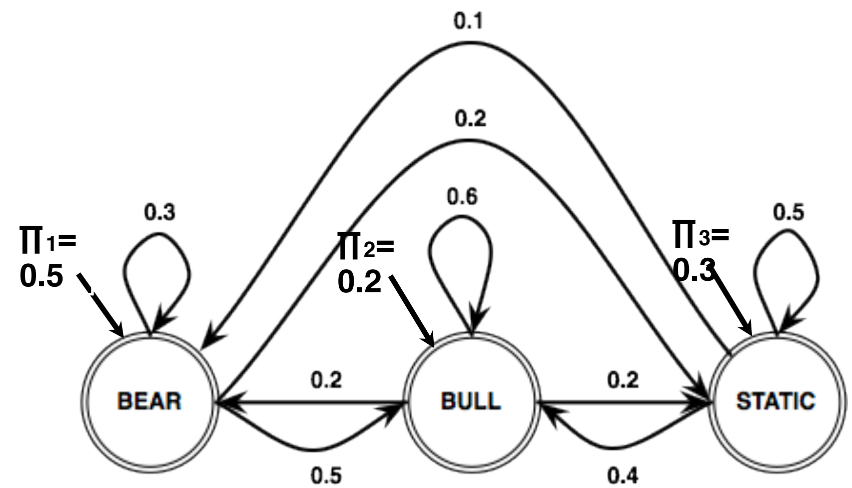
Applying HMMs

- 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 a set of observation sequences and the set of states Q in λ, compute the parameters A and B. [**Training**]

Computing Likelihood

t: 1 2 3 4 5 6

O: ↑ ↓ ↔ ↑ ↓ ↔



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

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

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

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

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

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

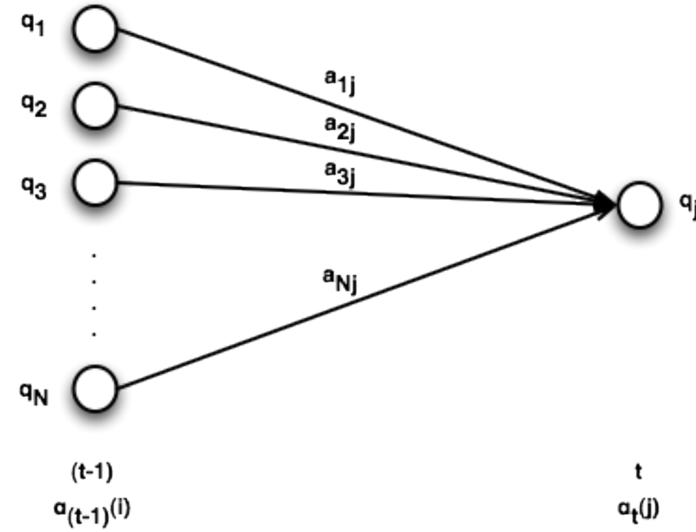
Forward Algorithm

- 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 = ∑ extensions of all paths from other cells

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

  - $\alpha_{t-1}(i)$: 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 ot 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

Bear

states

Bull

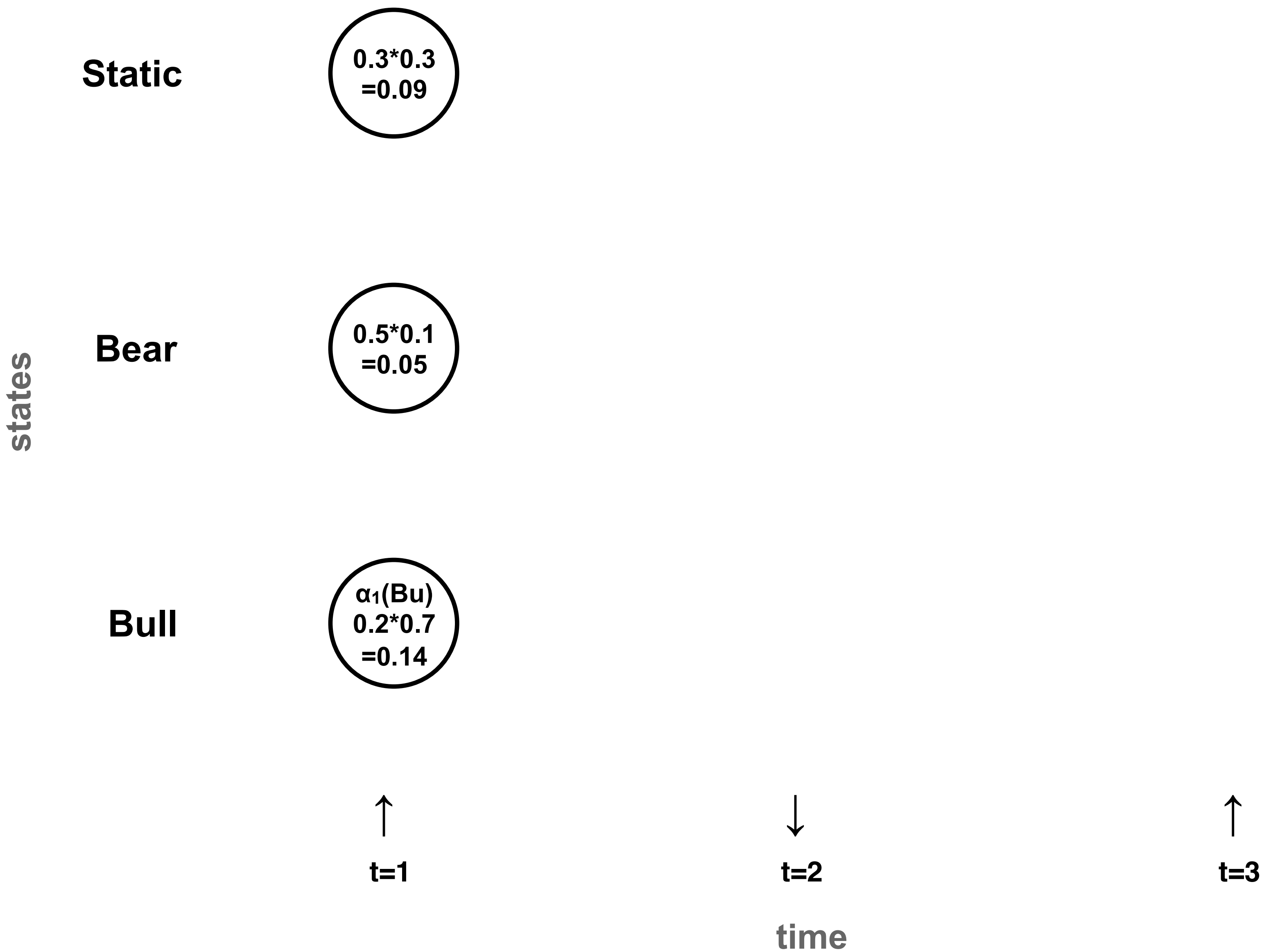
$$O = \uparrow \downarrow \uparrow$$

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

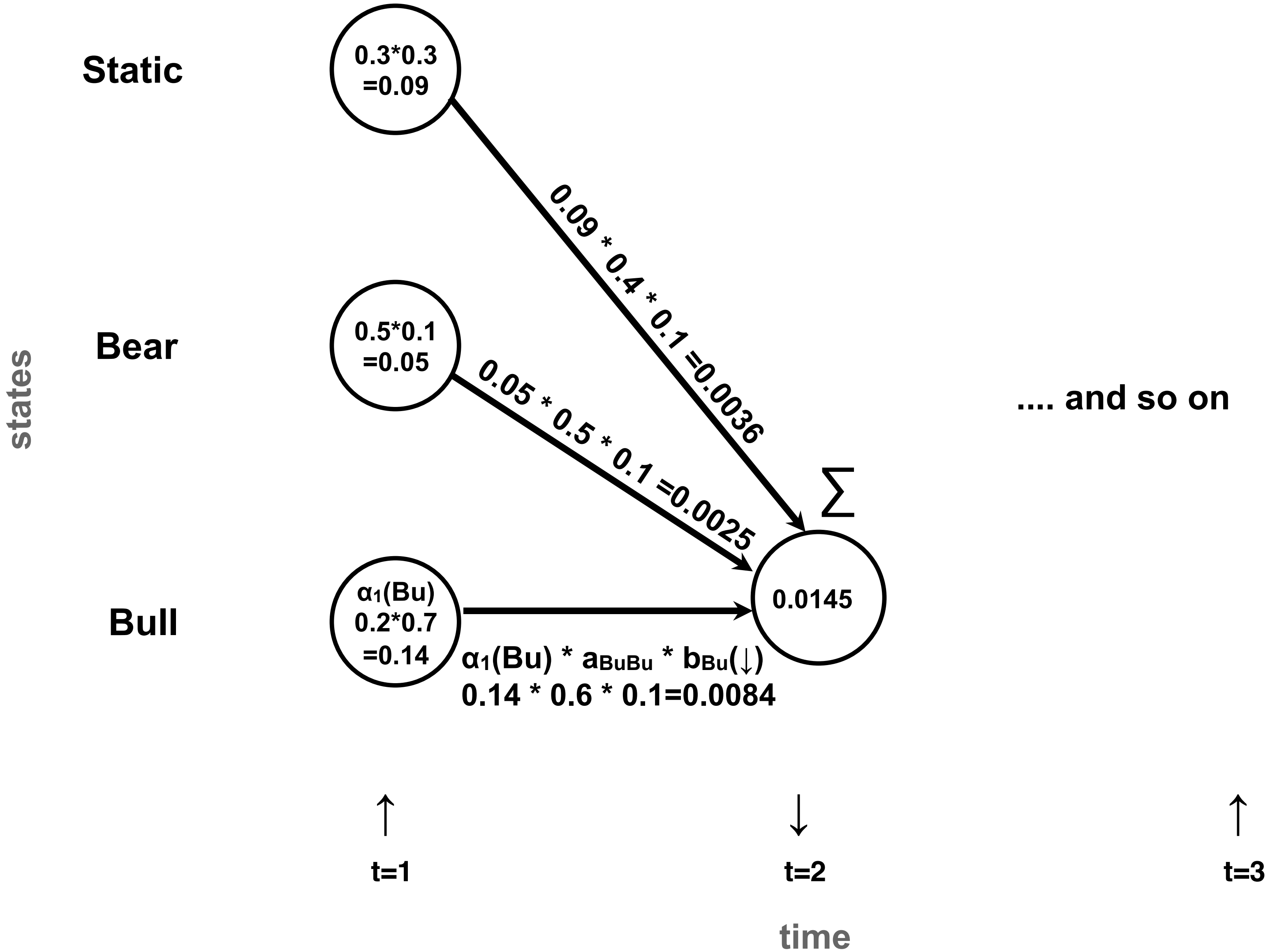
$\uparrow$	$\downarrow$	$\uparrow$
t=1	**t=2**	**t=3**

time

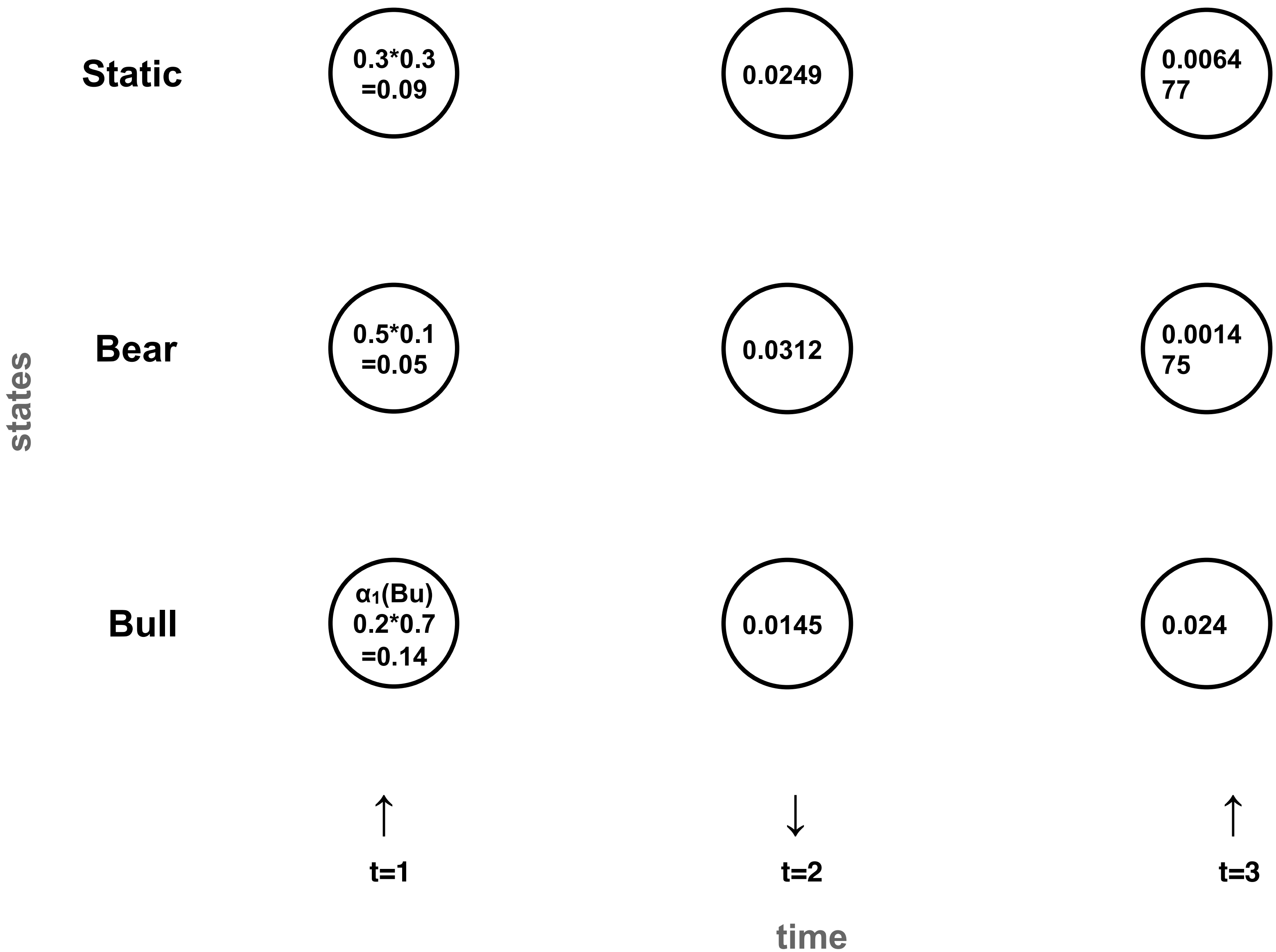
Forward Algorithm (Initialization)



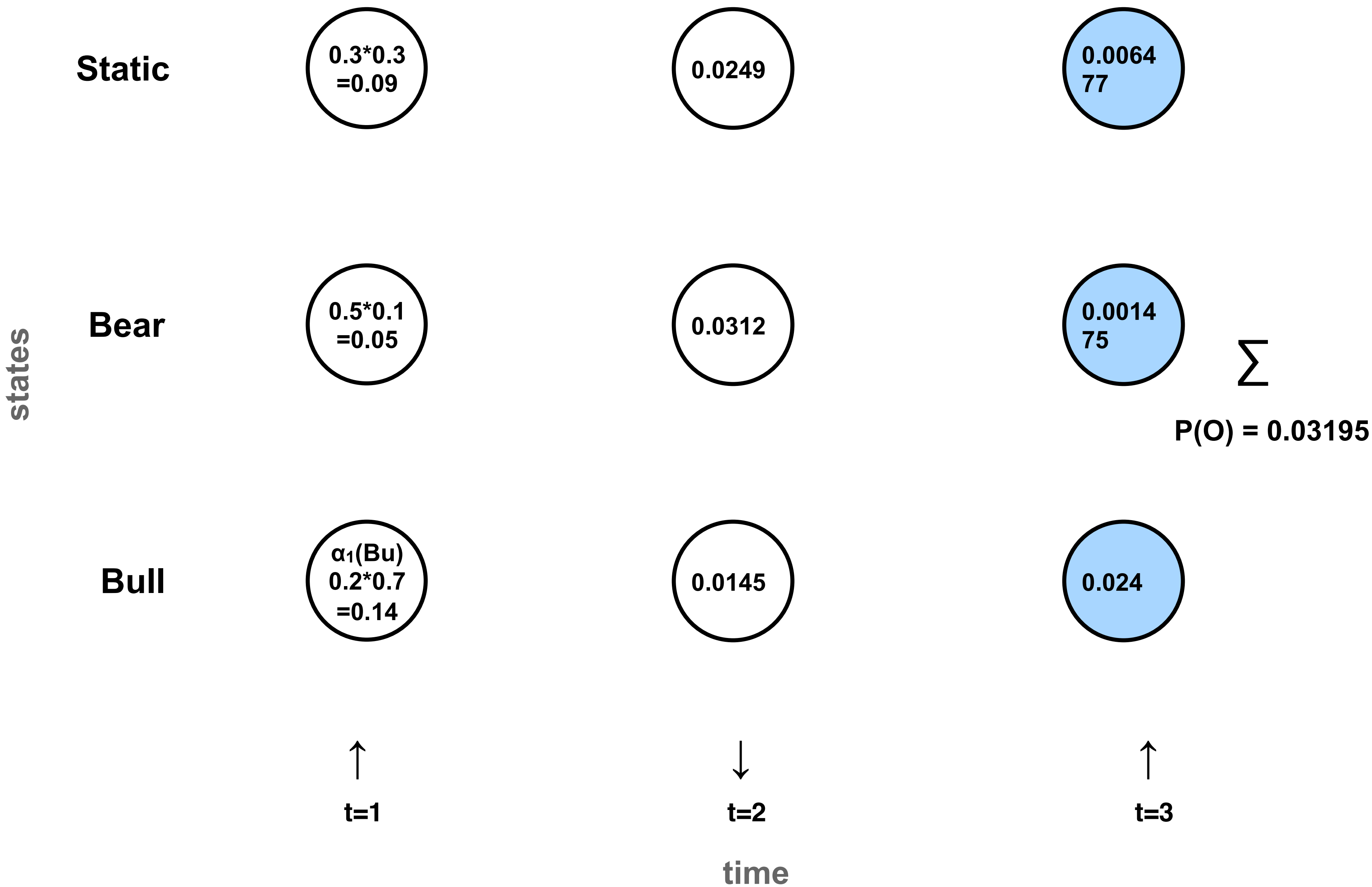
Forward Algorithm (Recursion)



Forward Algorithm (Recursion)



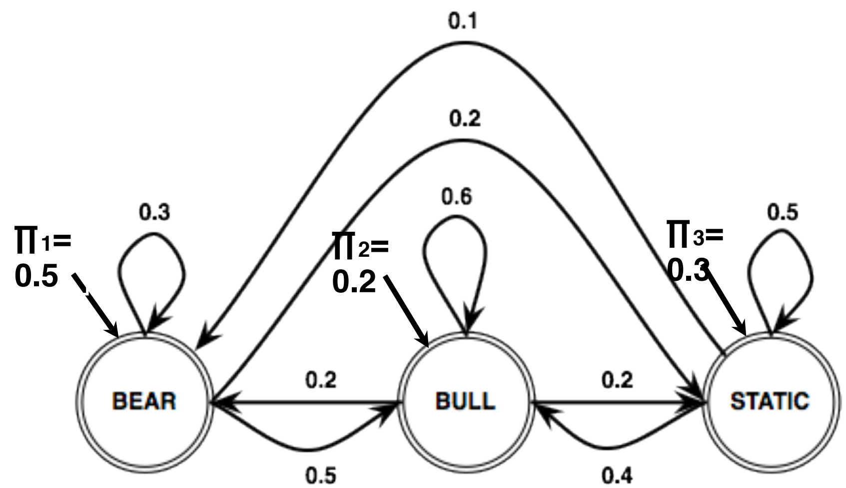
Forward Algorithm (Recursion)



Decoding

t: 1 2 3 4 5 6

O: ↑ ↓ ↔ ↑ ↓ ↔



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

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
- There’s a simple way to do it
  - 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 so far) $= p(q_1, q_2, ..., q_{t-1}, q_t{=}j, o_1, o_2, ... o_t)$
- Each cell = extension of most likely path from other cells

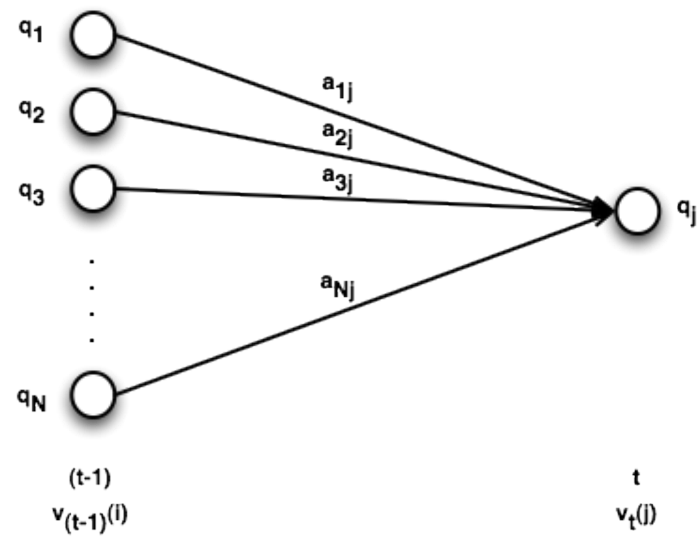
$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i) a_{ij} b_j(o_t)$$

  - $v_{t-1}(i)$: viterbi probability until time (t-1)
  - a_{ij} : transition probability of going from state i to j
  - $b_j(o_t)$: probability of emitting symbol ot 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

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

  - Recursion

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

Why no b() ?

  - Termination

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

Viterbi Algorithm

Static

Bear

states

Bull

O = ↑ ↓ ↑

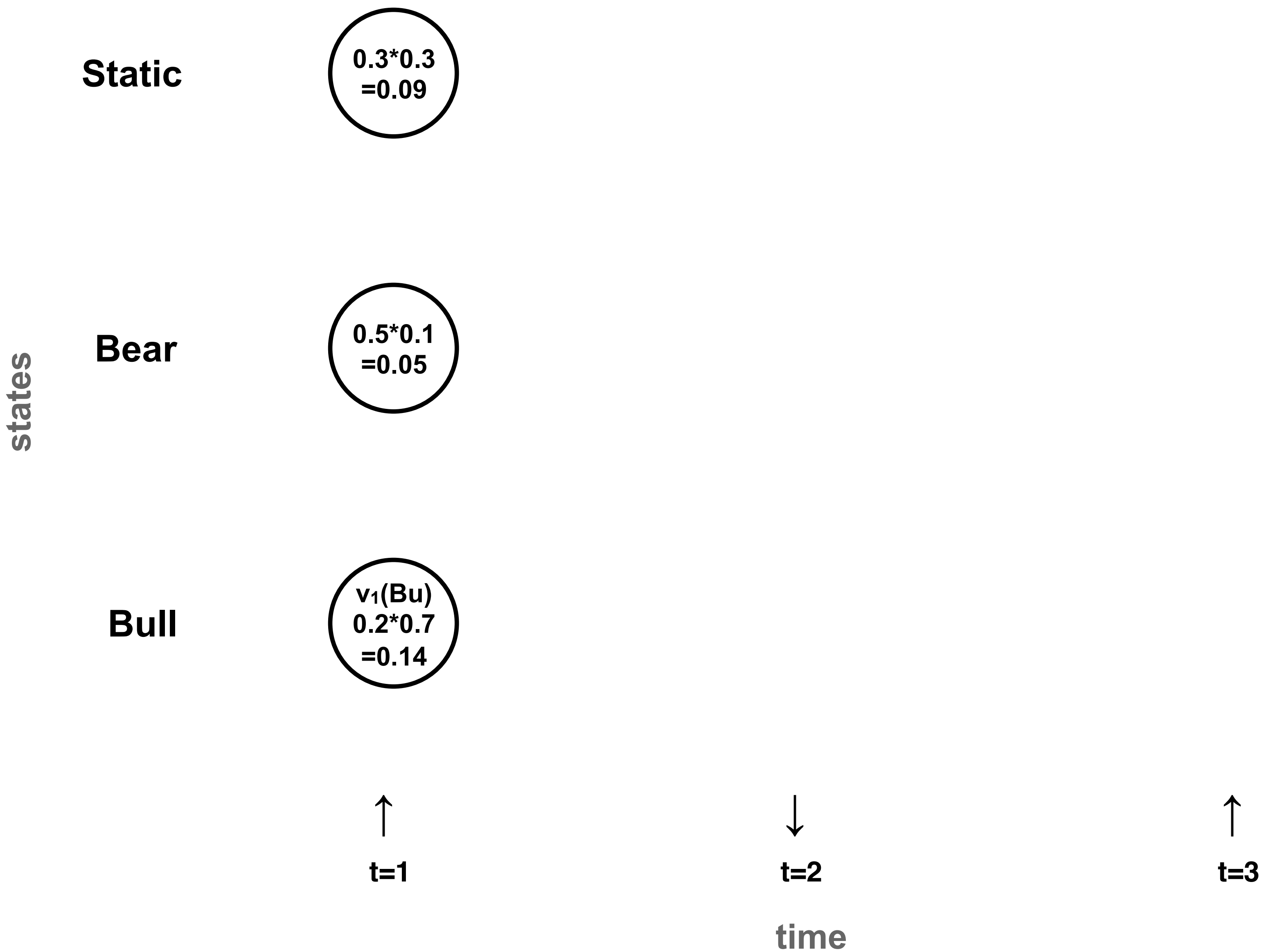
find most likely given state sequence

↑ ↓ ↑

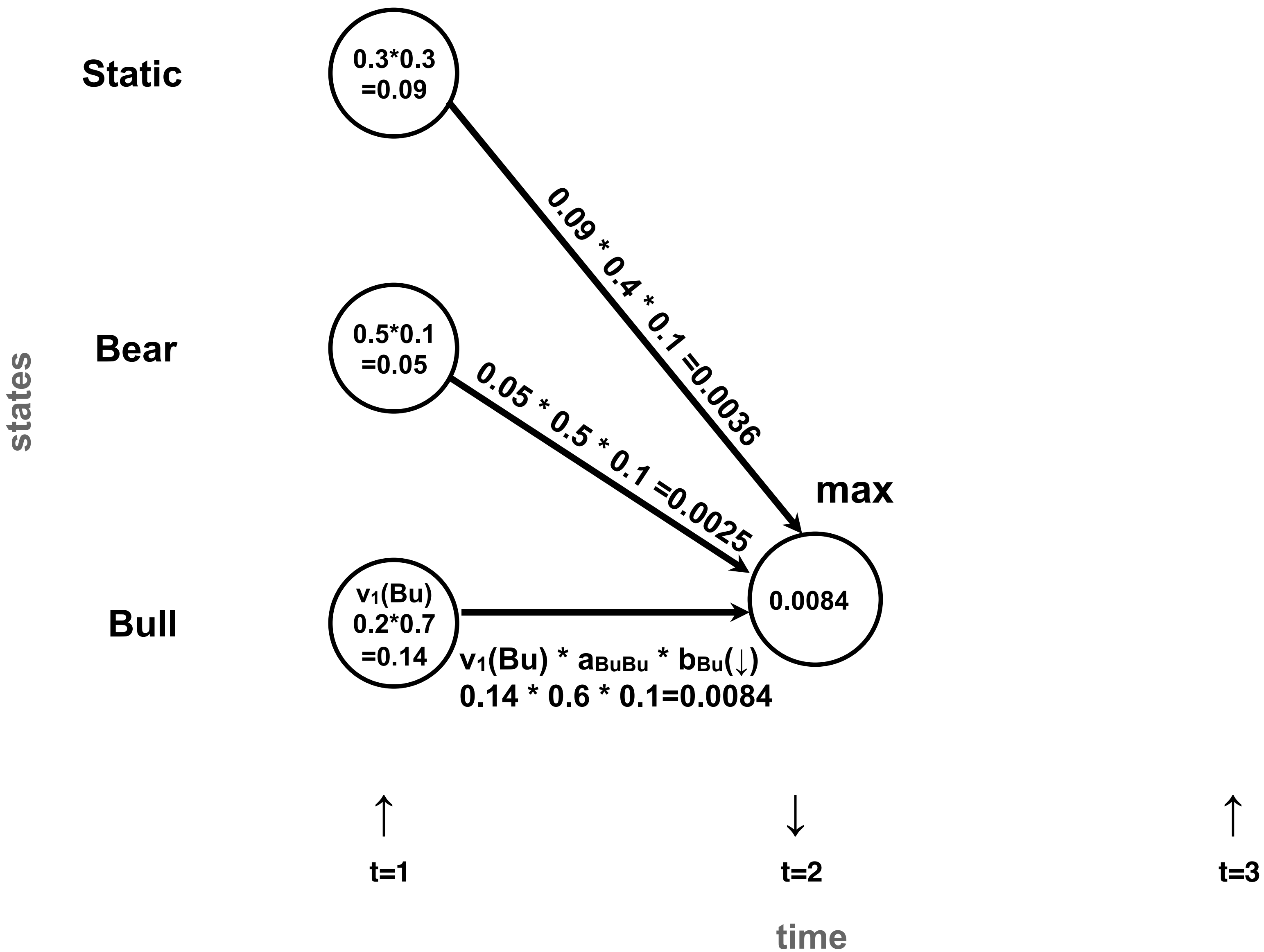
t=1 t=2 t=3

time

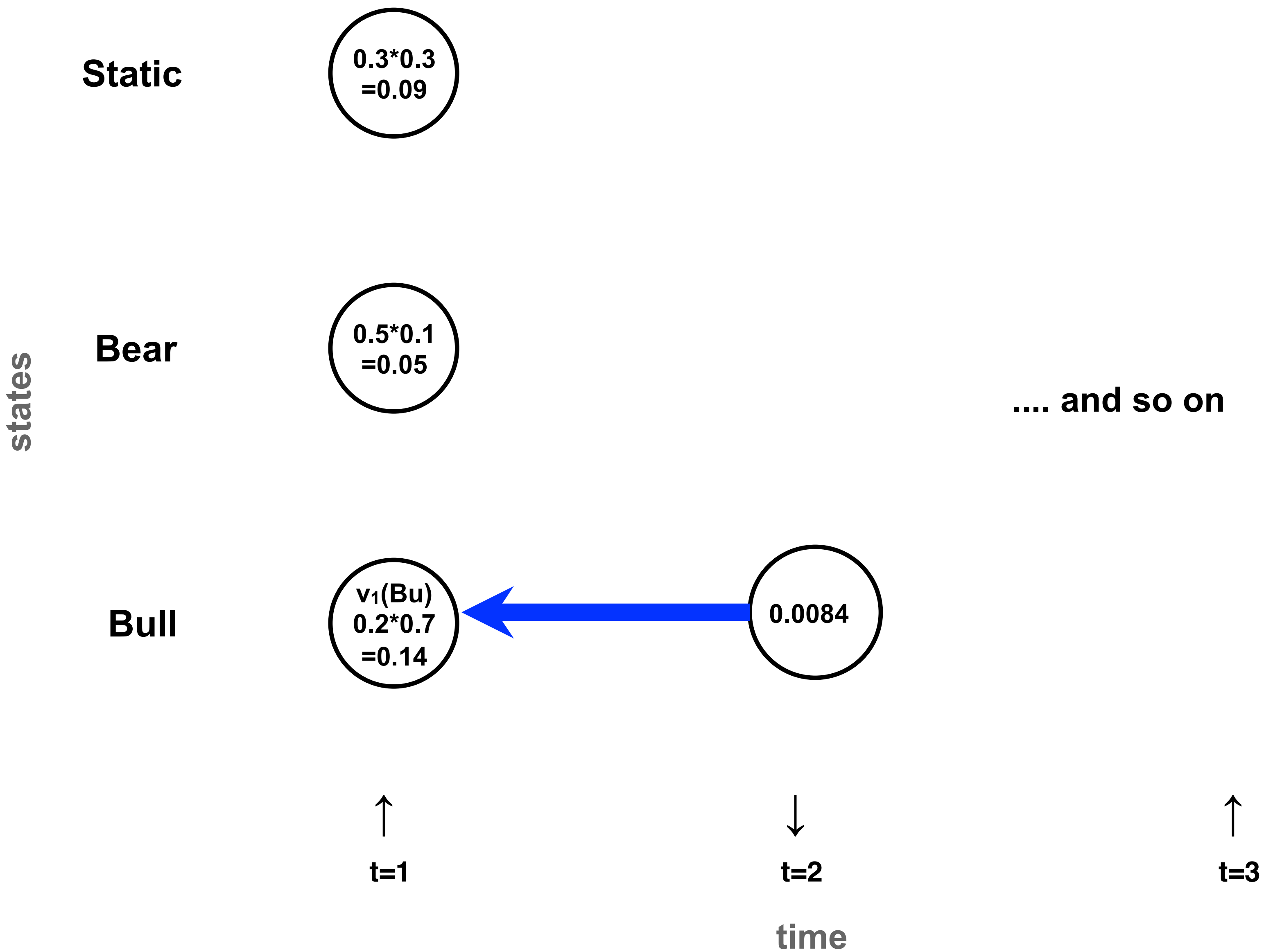
Viterbi Algorithm (Initialization)



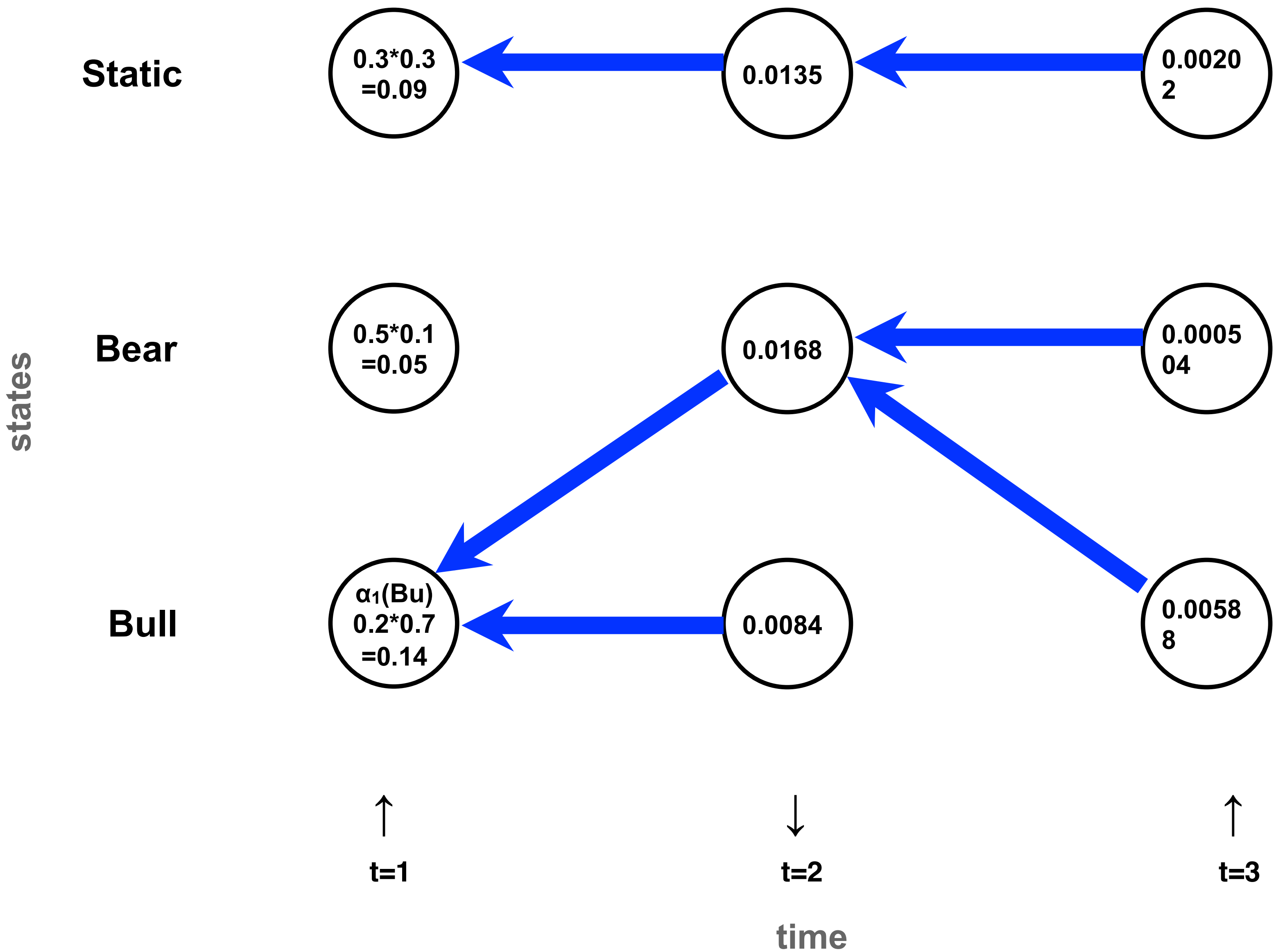
Viterbi Algorithm (Recursion)



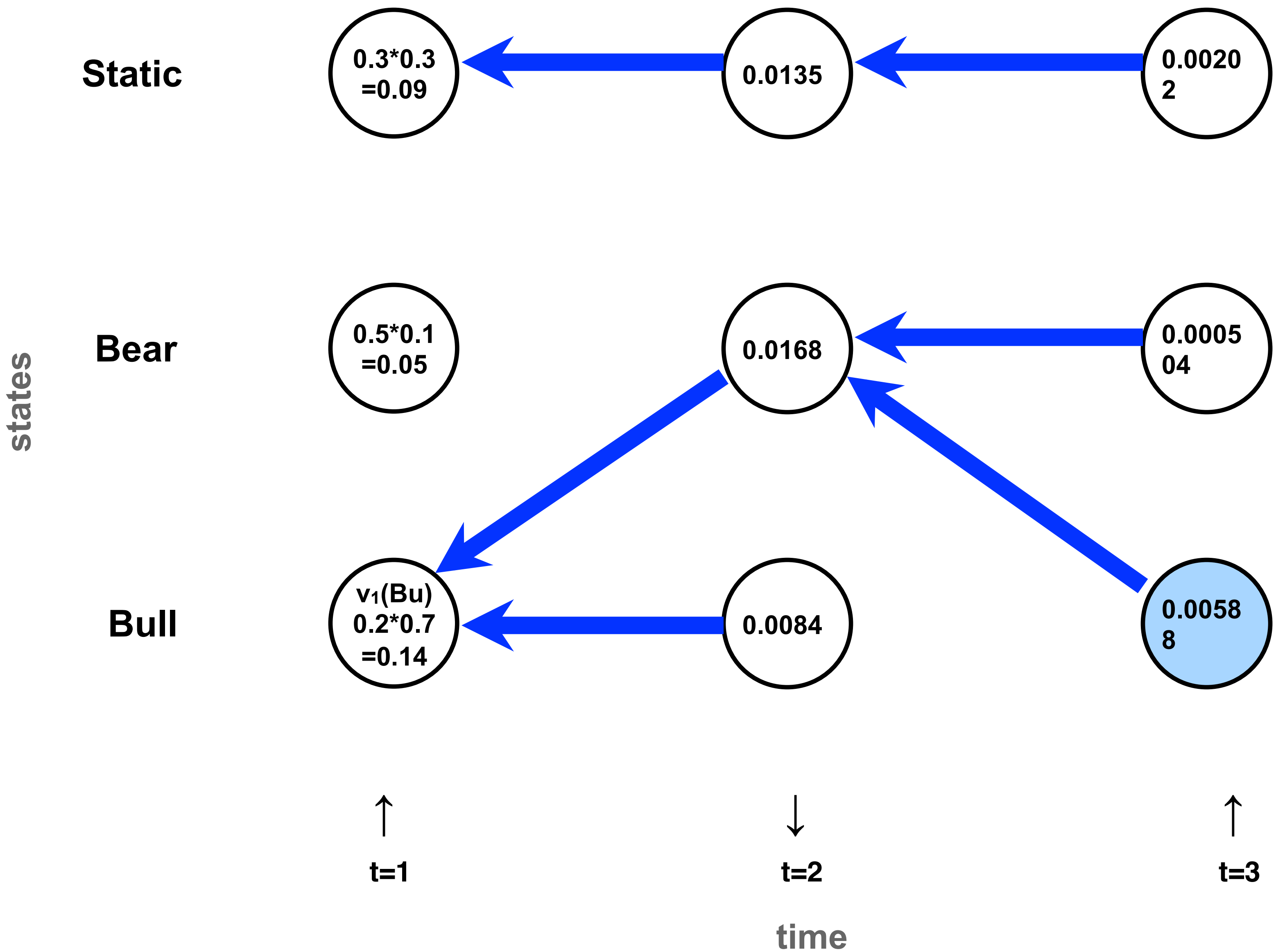
Viterbi Algorithm (Recursion)



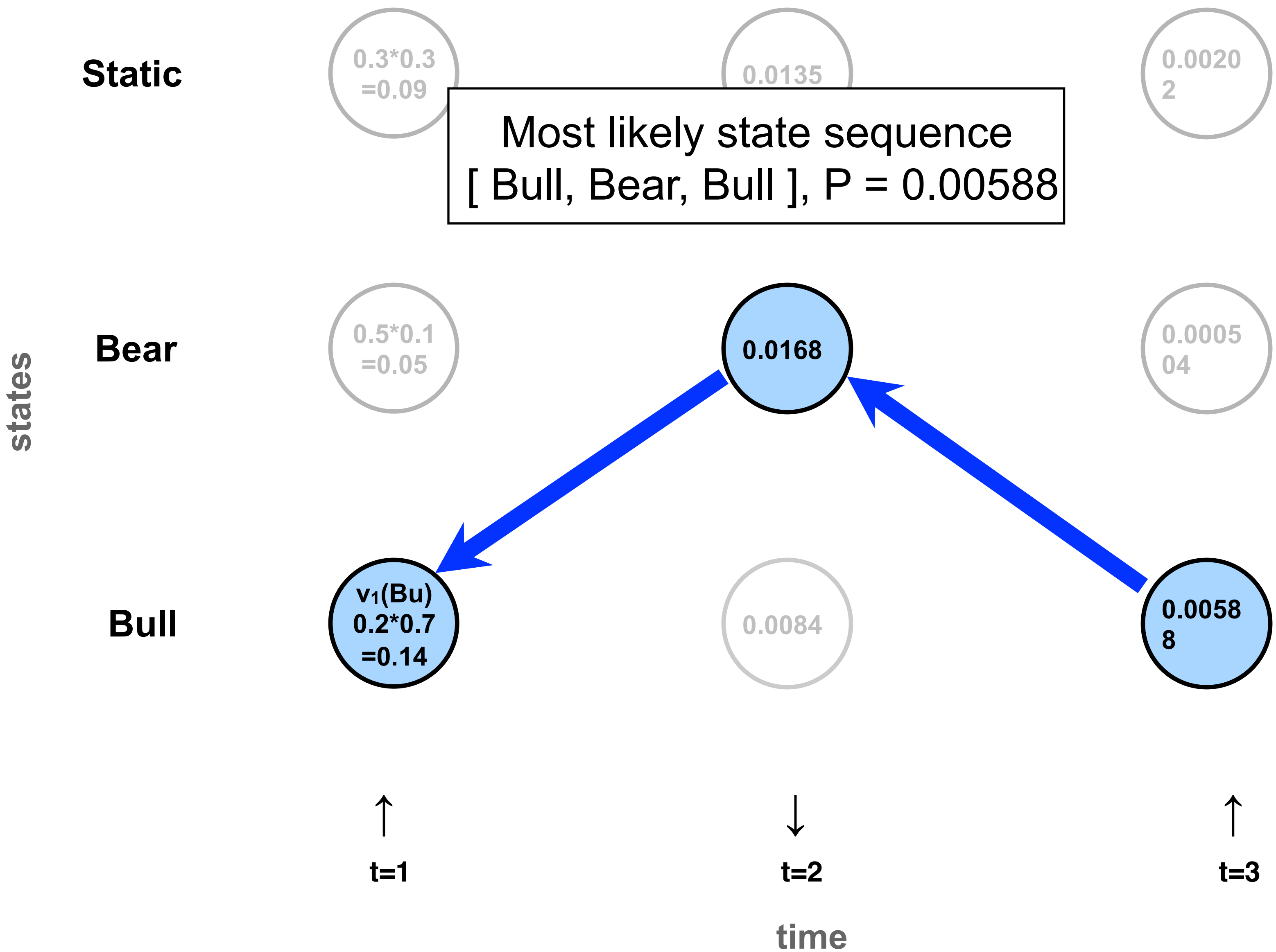
Viterbi Algorithm (Recursion)



Viterbi Algorithm (Termination)



Viterbi Algorithm (Termination)



Why are HMMs useful?

- Models of data that is ordered *sequentially*
  - Recall sequence of market up/down/static observations
- Other more useful sequences
  - Words in a sentence
  - Base pairs in a gene
  - Letters in a word
- Have been used for almost everything
  - Automatic speech recognition
  - Stock market forecasting (you thought I was joking?!)
  - Aligning words in a bilingual parallel text
  - **Tagging words with parts of speech**

Md. Rafiul Hassan and Baikunth Nath. *Stock Market Forecasting Using Hidden Markov Models: A New Approach*. Proceedings of the International Conference on Intelligent Systems Design and Applications.

Part of Speech Tagging

Part of Speech (POS) Tagging

- Parts of speech are well recognized linguistic entities
- *The Art of Grammar* circa 100 B.C.
  - Written to allow post-Classical Greek speakers to understand Odyssey and other classical poets
  - 8 *classes* of words
    [Noun, Verb, Pronoun, Article, Adverb, Conjunction, Participle, Preposition]
  - Remarkably enduring list
- Occur in almost every language
- Defined primarily in terms of syntactic and morphological criteria (affixes)

Part of Speech (POS) Tagging

- Two broad categories of POS tags

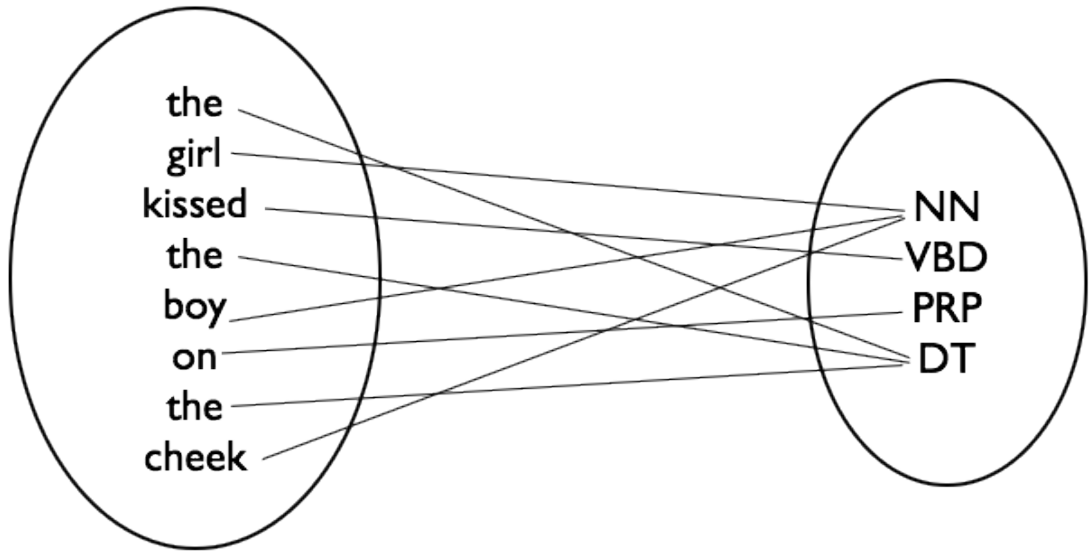
- Closed Class:
  - Relatively fixed membership
  - Conjunctions, Prepositions, Auxiliaries, Determiners, Pronouns …
  - *Function* words: short and used primarily for structuring

- Open Class:
  - Nouns, Verbs, Adjectives, Adverbs
  - Frequent neologisms (borrowed/coined)

Part of Speech (POS) Tagging

- Several English tagsets have been developed
- Vary in number of tags
  - Brown Tagset (87)
  - Penn Treebank (45) [More common]
- Language specific
  - Simple morphology = more ambiguity = smaller tagset
- Size depends on language and purpose

Part of Speech (POS) Tagging



POS Tagging: The process of assigning "one" POS or other lexical class marker to each word in a corpus

Why do POS tagging?

- Corpus-based Linguistic Analysis & Lexicography
- Information Retrieval & Question Answering
- Automatic Speech Synthesis
- Word Sense Disambiguation
- Shallow Syntactic Parsing
- Machine Translation

Why is POS tagging hard?

- Not really a lexical problem
- Sequence labeling problem
- Treating it as lexical problem runs us smack into the wall of ambiguity

I thought that you ...   ***(that: conjunction)***
That day was nice   ***(that: determiner)***
You can go that far   ***(that: adverb)***

HMMs & 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_M)
- Can an HMM find the best tagging for a given sentence ?
  - Yes ! Viterbi Decoding (best = most likely)
- Once we have an HMM model, tagging lots of data is embarrassingly parallel: a tagger in each mapper
- 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

- Recall the two types of training?

  - Supervised training: Use a large corpus of already tagged words as training data; count stuff; estimate model parameters

  - Unsupervised training: Use a corpus of 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 that the training data was actually generated by our model

Supervised Training

- Transition Probabilities
  - Any $P(t_i \mid t_{i-1}) = C(t_{i-1}t_i)/\Sigma_{t'}C(t_{i-1}t')$ from the training data
  - For P(NN|VB), count how many times a noun follows a verb and divide by the the number of times anything else follows a verb
- Emission Probabilities
  - Any $P(w_i \mid t_i) = C(w_i,t_i)/\Sigma_{w'}C(w', t_i)$ from the training data
  - For P(bank|NN), count how many times the word bank was seen tagged as a noun and divide by the number of times anything was seen tagged as a noun
- Priors
  - The prior probability of any state (tag)
  - For $\prod_{noun}$, count the number of times a noun occurs and divide by the total number of words in the corpus

Supervised Training in MapReduce

- Recall that we computed relative frequencies of words in MapReduce using the Stripes design
- Estimating HMM parameters via supervised training is identical

$$f(B|A) = \frac{c(A,B)}{\sum_{B'} c(A,B')}$$

(Eqn 3.1, p. 51)

Transition probabilities

Emission probabilities

$$p(t_i|t_{i-1}) = \frac{c(t_{i-1},t_i)}{\sum_{t'} c(t_{i-1},t')}$$

$$p(w_i|t_i) = \frac{c(w_i,t_i)}{\sum_{w'} c(w',t_i)}$$

$$\pi_i = \frac{c(t_i)}{N}$$

Priors is like counting words

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)

Expectation Maximization

- A fundamental tool for unsupervised machine learning techniques
- Forms basis of state-of-the-art systems in MT, Parsing, WSD, Speech Recognition and more
- Seminal paper (with a very instructive title) Maximum Likelihood from Incomplete Data via the EM algorithm, JRSS, Dempster et al., 1977

Motivating Example

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

Adapted 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."
- $P(data|model) = P(a,b,c,d|\mu) = K(1/2)^a(\mu)^b(2\mu)^c(1/2-3\mu)^d$
  [independent and identically distributed]
- L = log-likelihood = $\log P(a,b,c,d|\mu)$
  $= a \log(1/2) + b \log \mu + c \log 2\mu + d \log(1/2-3\mu)$
- How to maximize L w.r.t μ ? [Think Calculus]
- $\delta L/\delta\mu = 0$; $(b/\mu) + (2c/2\mu) - (3d/(1/2 - 3\mu)) = 0$
- $\mu = (b+c)/6(b+c+d)$ [Note missing 'a']
- We got our answer without EM. Boring !

Motivating Example

- $P(A) = 1/2$, $P(B) = \mu$, $P(C) = 2\mu$, $P(D) = 1/2 - 3\mu$
- Number of ‘A’s and ‘B’s = h, c ‘C’s and d ‘D’s
- Part of the observable information is hidden
- Can we compute the MLE for μ now?
- 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 (μ0)
- t = 1; Repeat
  - $b_t = \mu_{(t-1)}h/(1/2 + \mu_{(t-1)})$
    [E-step: Compute expected value of b given μ]
  - $\mu_t = (b_t + c)/6(b_t + c + d)$
    [M-step: Compute MLE of μ given b]
  - t = t + 1
- Until some convergence criterion is met

The EM Algorithm

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

Applying EM to HMMs

- Just the intuition; No gory details
- 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 $\prod$
- t = 1; Repeat
  - E-step: Compute expected values of ξ, γ using A_t, B_t, $\prod_t$
  - M-step: Compute MLE of A, B and $\prod$ using ξ_t, γ_t
  - $t = t + 1$
- Until some specified convergence criterion is met
- Optional: Read Section 6.2 in Lin & Dyer for gory details

EM in MapReduce

- Each iteration of EM is one MapReduce job
- A driver program spawns MR jobs, keeps track of the number of iterations and convergence criteria
- Model parameters static for the duration of each job are loaded by each mapper from HDFS
- Mappers map over independent instances from training data to do computations from E-step
- Reducers sum together stuff from mappers to solve equations from M-step
- Combiners are important to sum together the training instances in memory and reduce disk I/O

Source: Wikipedia (Japanese rock garden)