Lecture # 14
Session 2003

# Part I: Designing HMM-based ASR systems

**Rita Singh**

School of Computer Science
Carnegie Mellon University

# Table of contents

# Statistical Pattern Classification

- Given data *X*, find which of a number of classes $C_1$, $C_2$,…$C_N$ it belongs to, based on known distributions of data from $C_1$, $C_2$, etc.
- Bayesian Classification:

  $\text{Class} = C_i : i = \text{argmax}_j \, P(C_j)P(X|C_j)$

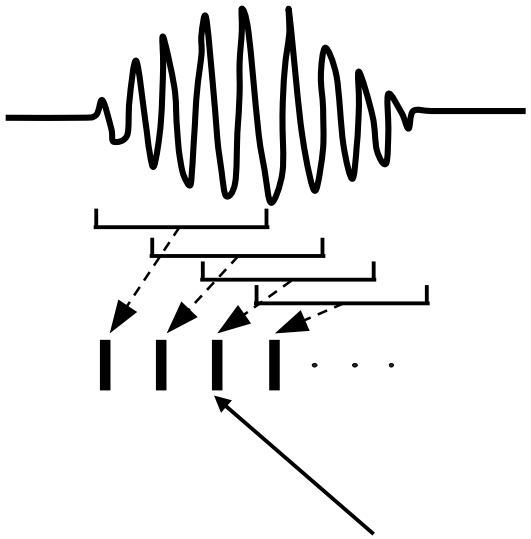
  *a priori* probability of $C_j$

  Probability of $X$ as given by the probability distribution of $C_j$

- The *a priori* probability accounts for the relative proportions of the classes
  - If you never saw any data, you would guess the class based on these probabilities alone
- $P(X|C_j)$ accounts for evidence obtained from observed data $X$

# Statistical Classification of Isolated Words

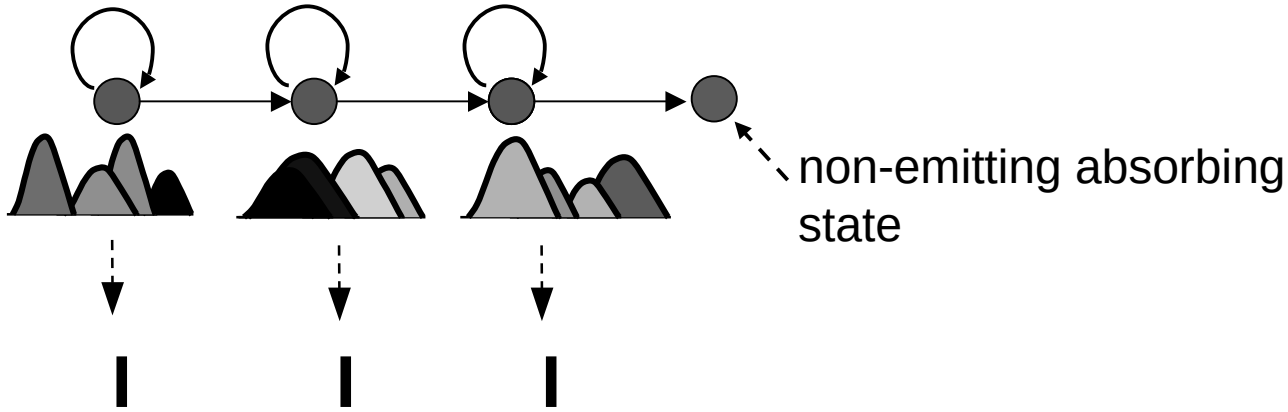
- Classes are words
- Data are instances of isolated spoken words
  - Sequence of feature vectors derived from speech signal, typically 1 vector from a 25ms frame of speech, with frames shifted by 10ms.



- Bayesian classification:
  Recognized_Word = $\text{argmax}_{word}$ P(*word*)P(*X*| *word*)
- P(*word*) is *a priori* probability of *word*
  - Obtained from our expectation of the relative frequency of occurrence of the word
- P(*X*|*word*) is the probability of *X* computed on the probability distribution function of *word*

# Computing P($X$|*word*)

- To compute P($X$|*word*), there must be a statistical distribution for $X$ corresponding to *word*
  - Each word must be represented by some statistical model.
- We represent each word by an HMM
  - An HMM is really a graphical form of probability density function for time-varying data

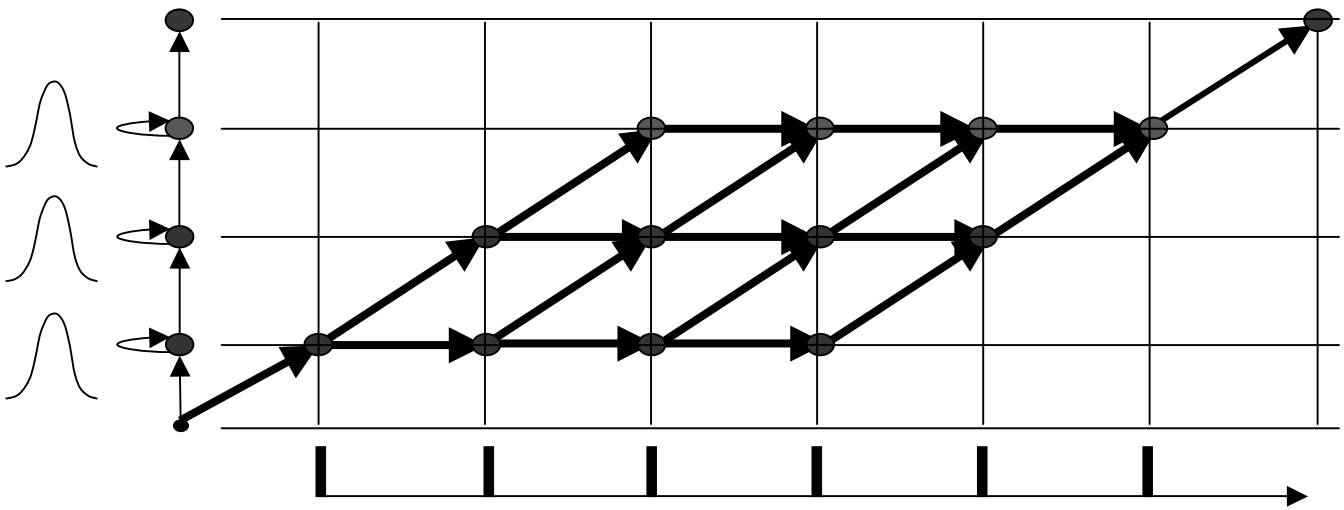



- Each state has a probability distribution function
- Transitions between states are governed by transition probabilities
- At each time instant the model is in some state, and it emits one observation vector from the distribution associated with that state

# Computing P(*X*|*word*)

- The actual state sequence that generated *X* is never known.
  - P(*X*|*word*) must therefore consider *all* possible state sequences.

$$P(X \mid word) = \sum_{s \in \{\text{all state sequences}\}} P(X, s \mid word)$$



The probabilities of all possible state sequences must be added to obtain the total probability

# Computing P(*X*|*word*)

- The actual state sequence that generated *X* is never known.
  - P(*X*|*word*) must therefore consider *all* possible state sequences.

$$P(X \mid word) = \sum_{s \in \{\text{all state sequences}\}} P(X, s \mid word)$$

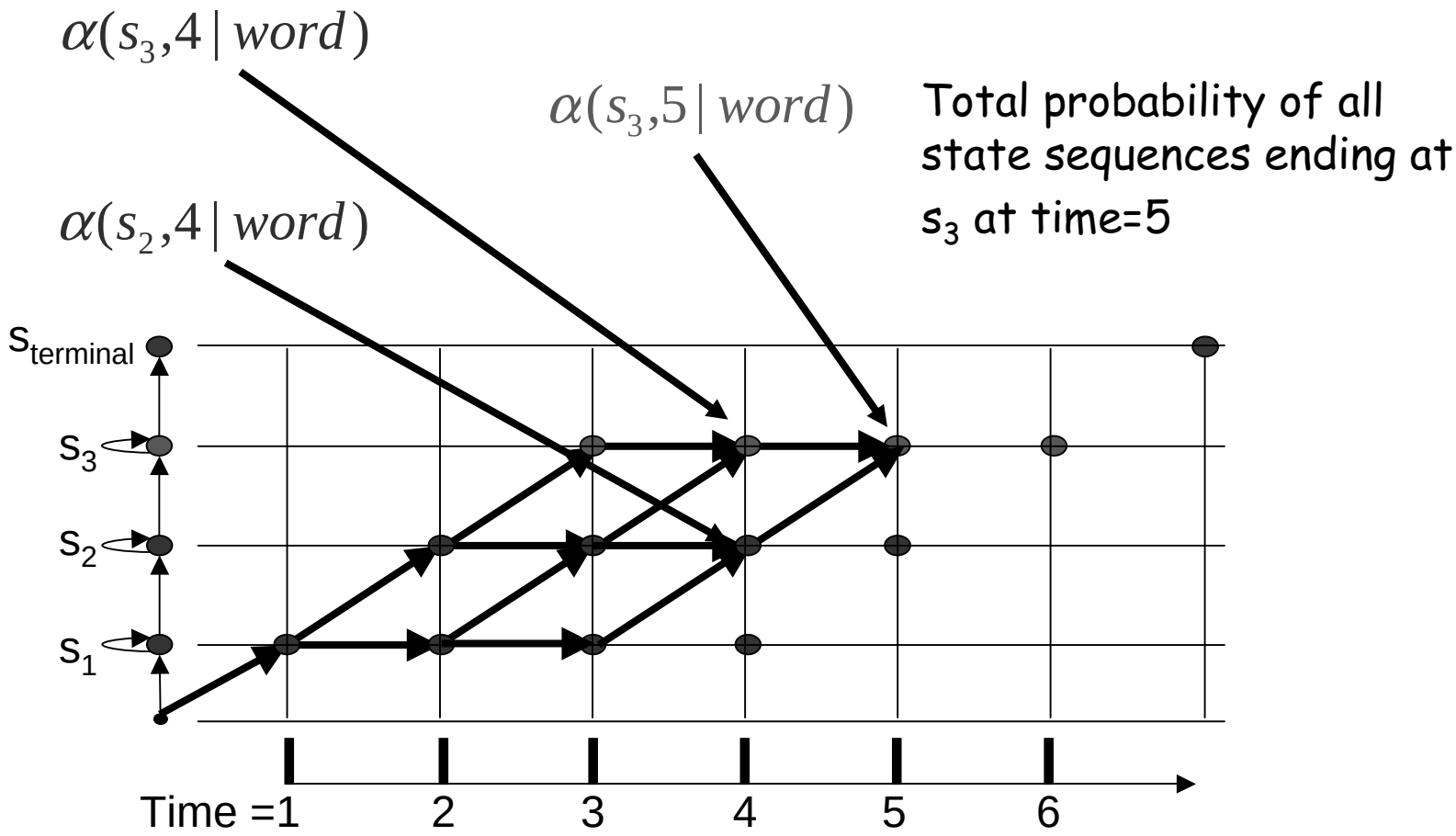
- The actual number of state sequences can be very large
  - Cannot explicitly sum over all state sequences
- P(*X*|*word*) can however be efficiently calculated using the forward recursion

$$\alpha(s,t \mid word) = \sum_{s'} \alpha(s',t-1 \mid word) P(s \mid s') P(X_t \mid s)$$

Total probability of all state sequences ending in state *s* at time *t*

# Computing P($X$|*word*)

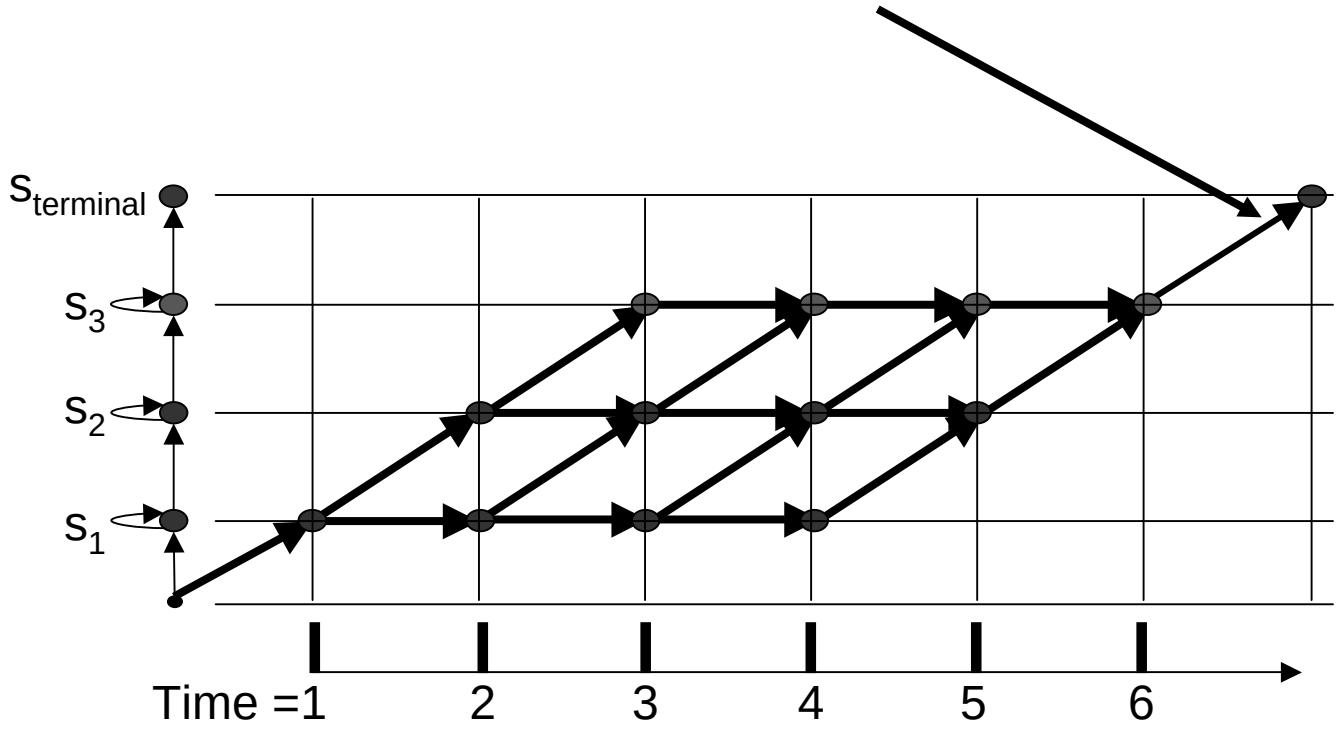
- The forward recursion



# Computing P(*X*|*word*)

- The total probability of *X*, including the contributions of all state sequences, is the forward probability at the final non-emitting node

$$P(X \mid word) = \alpha(s_{terminal}, T \mid word)$$

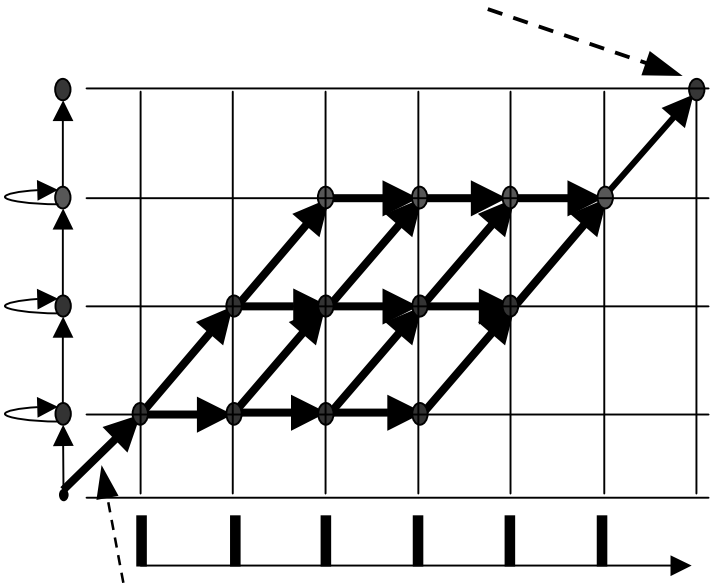


# Decoding isolated words

# Classifying between two words: *Odd* and *Even*

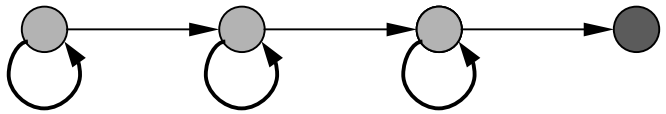
HMM for *Odd*

P(*Odd*)P(*X*|*Odd*)

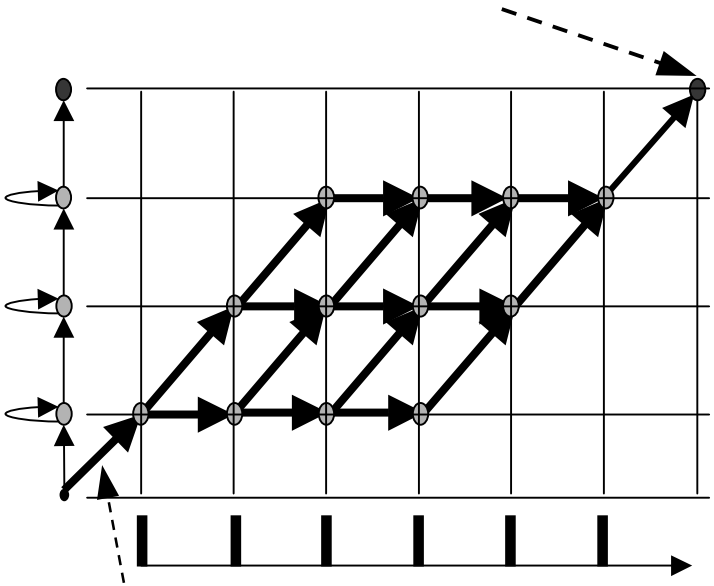


P(*Odd*)

HMM for *Even*

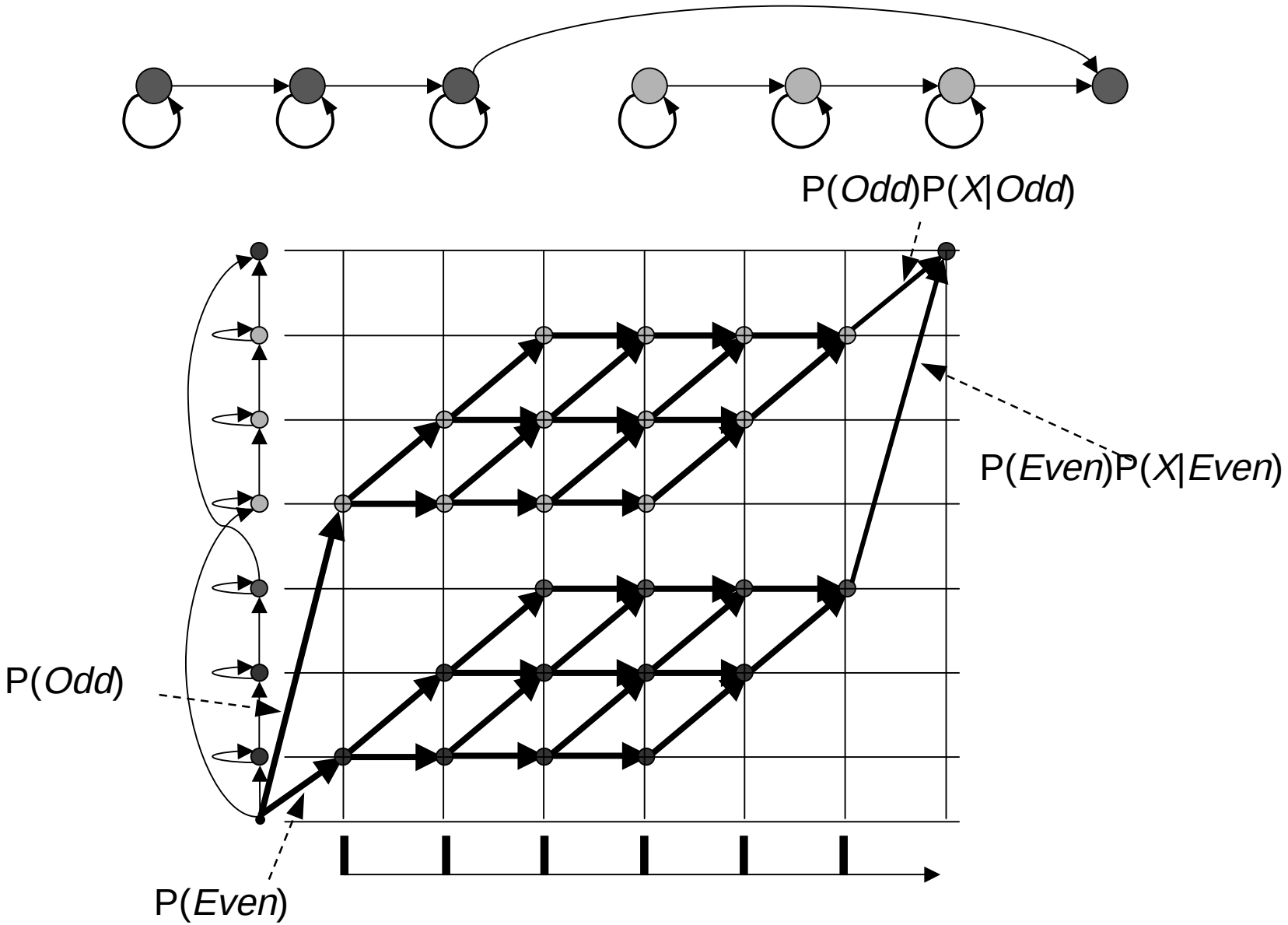


P(*Even*)P(*X*|*Even*)



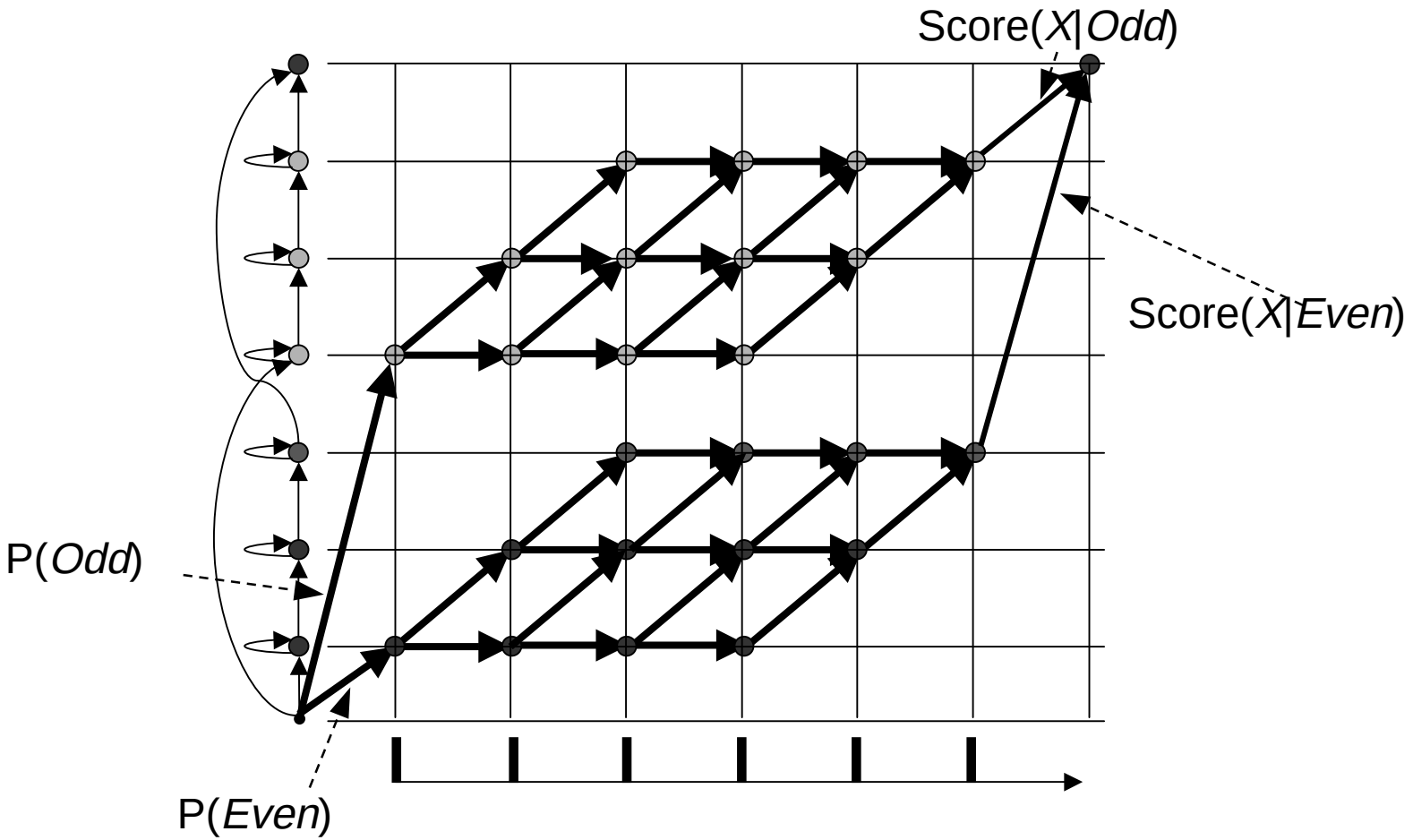
P(*Even*)

# Classifying between *Odd* and *Even*



# Decoding to classify between *Odd* and *Even*

- Approximate total probability of all paths with probability of best path
  - Computations can be done in the log domain. Only additions and comparisons are required
    - Cheaper than full-forward where multiplications and additions are required



# Computing Best Path Scores for Viterbi Decoding

- The approximate score for a word is

$$P(X \mid word) \approx \max_{s \in \{\text{all state sequences}\}} \{P(X, s \mid word)\}$$

- Written explicity

$$P(X_1..X_t \mid word) \approx \max_{s_1, s_2, ..., s_T} \{\pi(s_1)P(X_1 \mid s_1)P(s_2 \mid s_1)P(X_2 \mid s_2)....P(s_T \mid s_{T-1})P(X_T \mid s_T)\}$$

state sequence

- The word score can be be recursively computed using the Viterbi algorithm

$$P_s(t) = \max_{s'} \{P_{s'}(t-1)P(s \mid s')P(X_t \mid s)\}$$

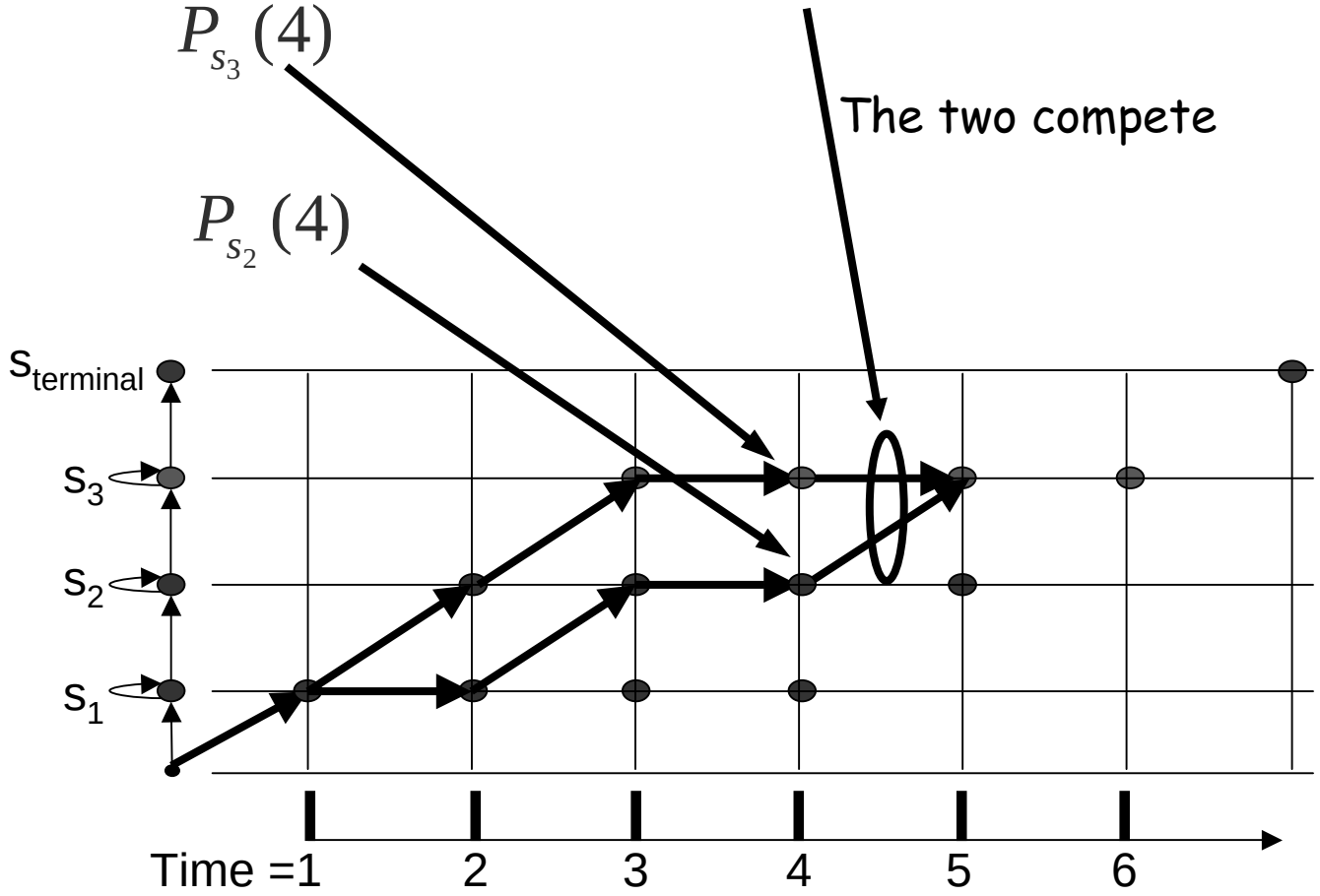
Best path score of all state sequences ending in state *s* at time *t*

Best path score of all state sequences ending in state *s'* at time *t-1*

# Computing Best Path Scores for Viterbi Decoding

- The forward recursion

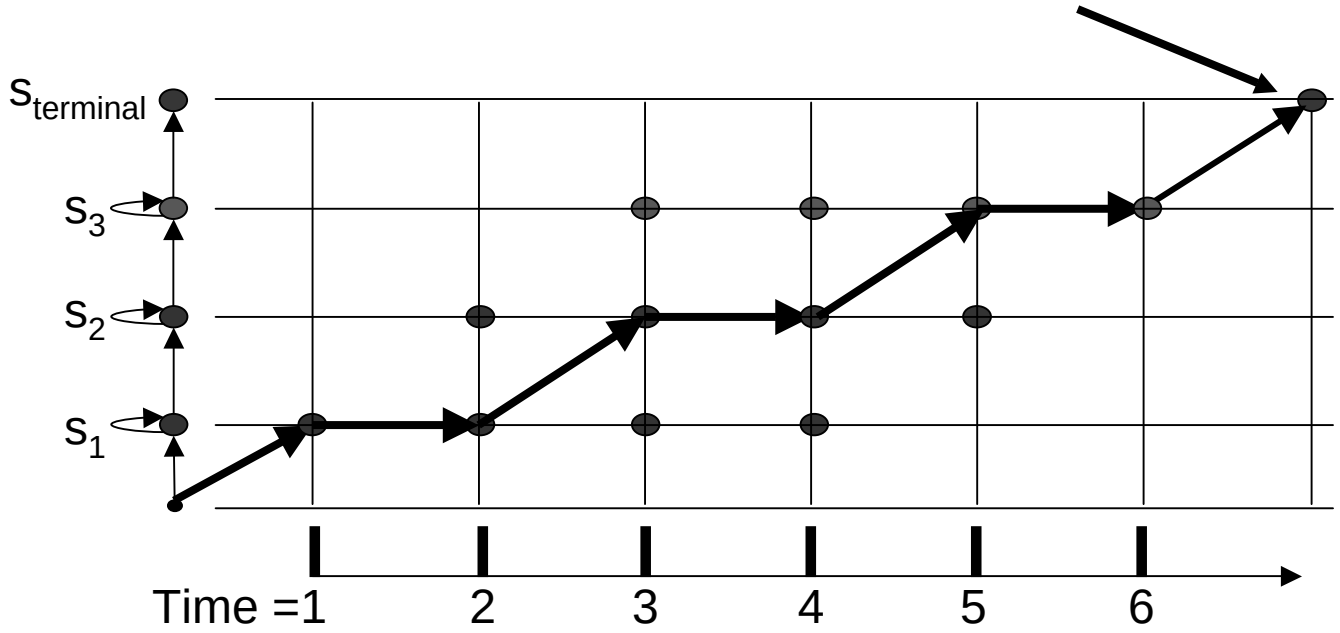
$$P_{s_3}(5) = \max\{ P_{s3}(4)P(s_3 \mid s_3)P(X_5 \mid s_3), P_{s_2}(4)P(s_3 \mid s_2)P(X_5 \mid s_3)\}$$



# Computing Best Path Scores for Viterbi Decoding

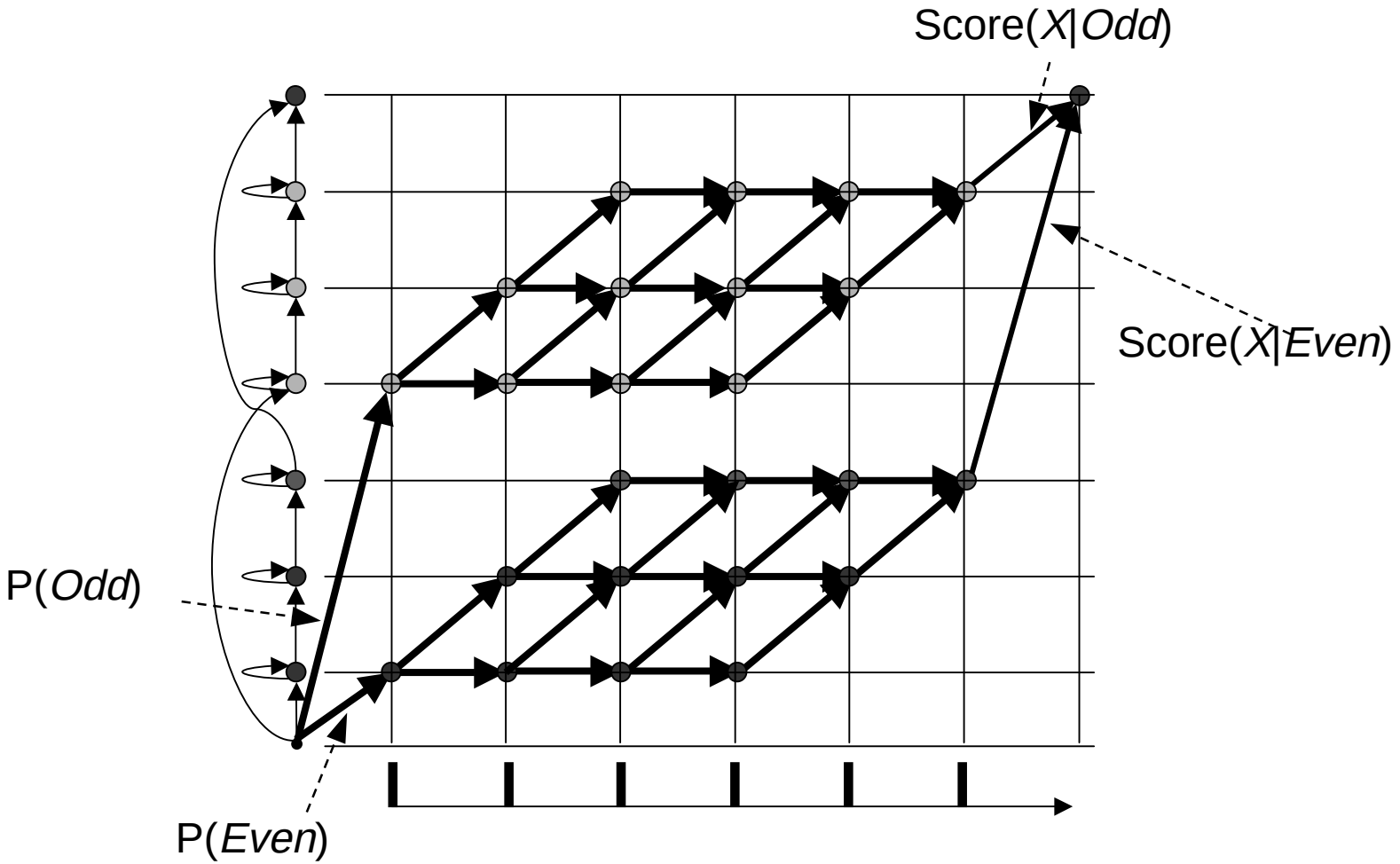
- The forward recursion is termed Viterbi *decoding*
  - The terminology is derived from decoding of error correction codes
- The score for the word is the score of the path that wins through to the terminal state
  - We use the term *score* to distinguish it from the total probability of the word

$$Score(X \mid word) = P_{s_{terminal}}(T)$$



# Decoding to classify between *Odd* and *Even*

- Compare scores (best state sequence probabilities) of all competing words



# Decoding word sequences

# Statistical Classification of Word Sequences

- Classes are word sequences
- Data are spoken recordings of word sequences

- Bayesian classification:

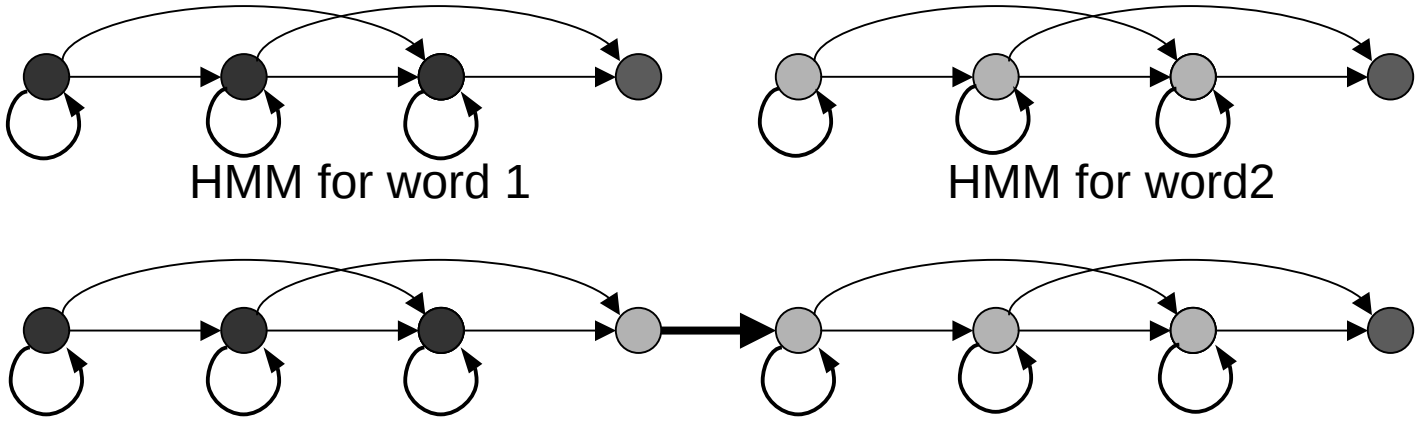
$$word_1, word_2, ..., word_N =$$
$$\arg\max_{wd_1, wd_2, ..., wd_N} \{P(X \mid wd_1, wd_2, ..., wd_N) P(wd_1, wd_2, ..., wd_N)\}$$

- P($wd_1$,$wd_2$,$wd_3$..) is *a priori* probability of word sequence $wd_1$,$wd_2$,$wd_3$..
  - Obtained from a model of the language

- P($X$| $wd_1$,$wd_2$,$wd_3$..) is the probability of $X$ computed on the probability distribution function of the word sequence $wd_1$,$wd_2$,$wd_3$..
  - HMMs now represent probability distributions of word sequences

# Constructing HMMs for word sequences

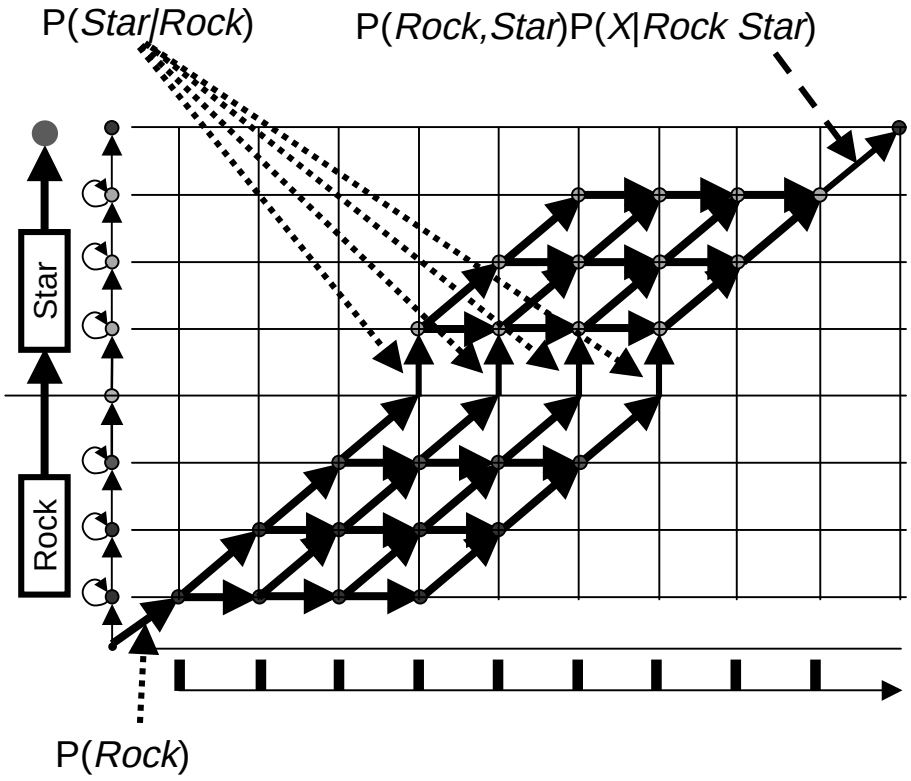
- Conacatenate HMMs for each of the words in the sequence

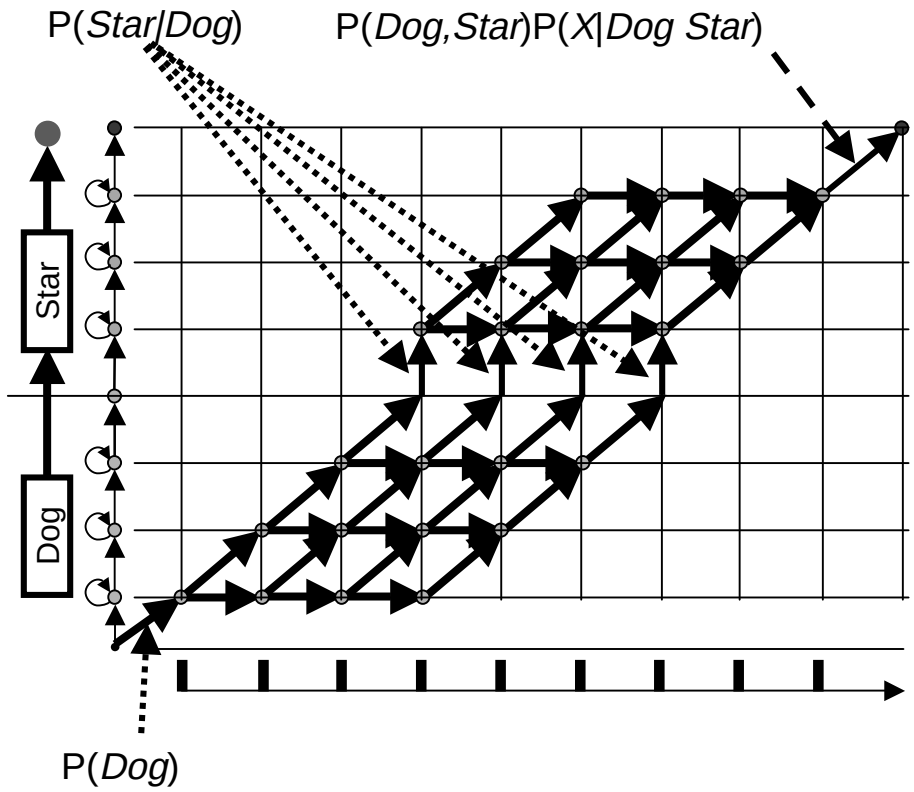



- In fact, word HMMs themselves are frequently constructed by concatenating HMMs for phonemes
  - Phonemes are far fewer in number than words, and occur more frequently in training data
  - Words that were never seen in the training data can be constructed from phoneme HMMs

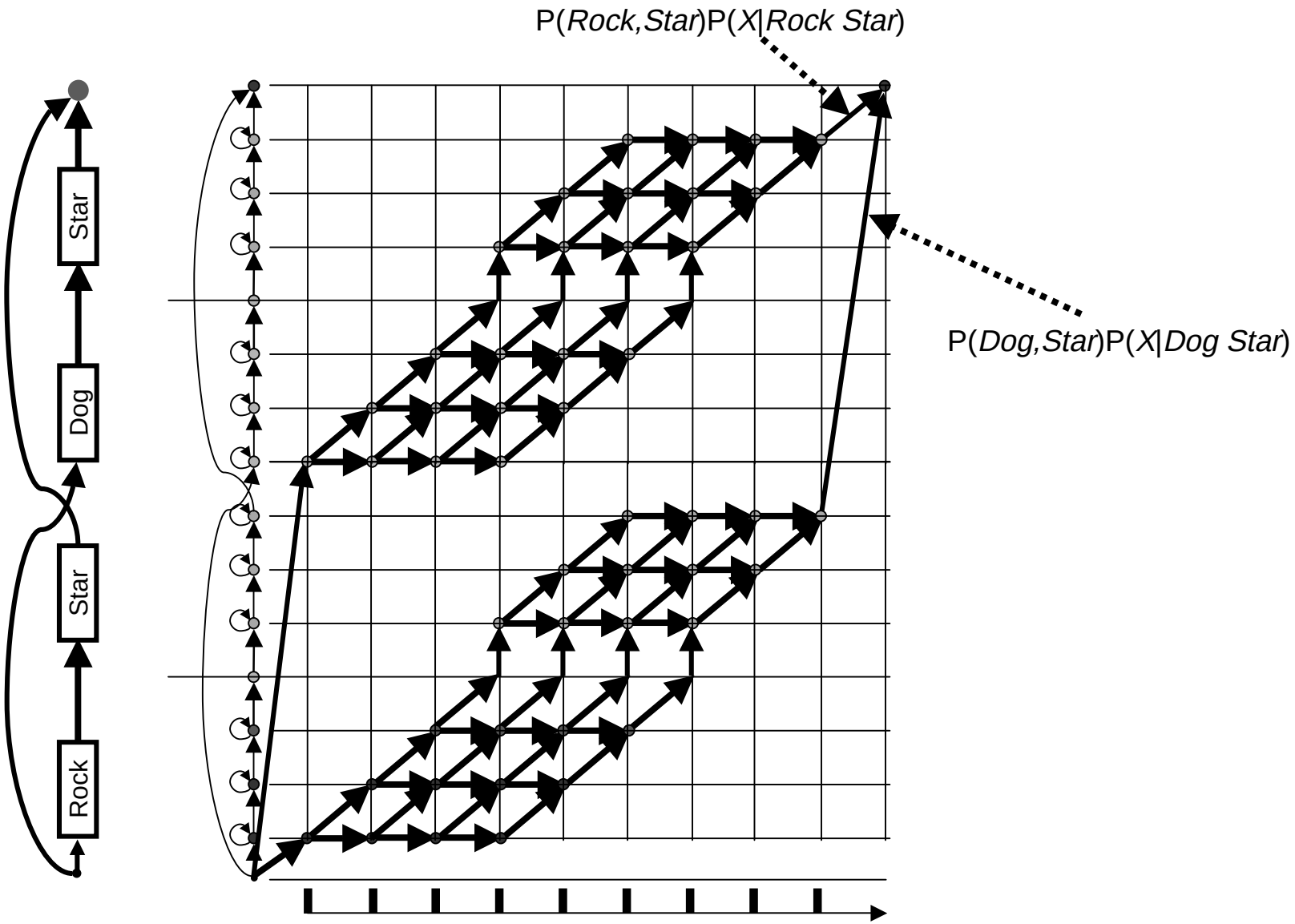
# Bayesian Classification between word sequences

- Classifying an utterance as either “Rock Star” or “Dog Star”
  - Must compare P(Rock,Star)P(X|Rock Star) with P(Dog,Star)P(X|Dog Star)



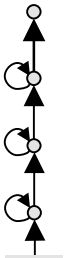
# Bayesian Classification between word sequences



# Extending paths through the trellis: Breadth First vs. Depth First Search

# Designing optimal graph structures for the language HMM

# Language HMMs for Natural language: building graphs to incorporate trigram representations



Each word is an HMM

- Explanation with example: three words vocab “the”, “rock”, “star”
  - The graph initially begins with bigrams of START, since nothing precedes START
  - Each word in the second level represents a specific set of two terminal words in a partial word sequence