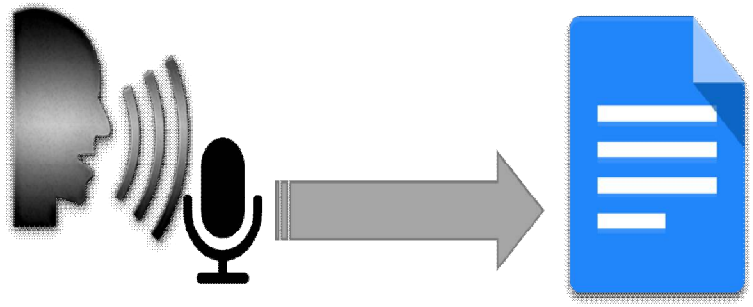
First Tbilisi International Summer School in Logic, Language, Artificial Intelligence

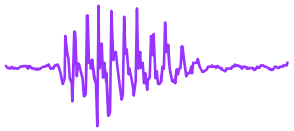
# Language Modeling Techniques for Continuous Speech Recognition

Bagher BabaAli
University of Tehran

# Introduction to speech recognition

- Converting Speech to Text





Speech Signal

Speech Recognition

Words

*"How are you?"*

# Basic concepts of speech recognition

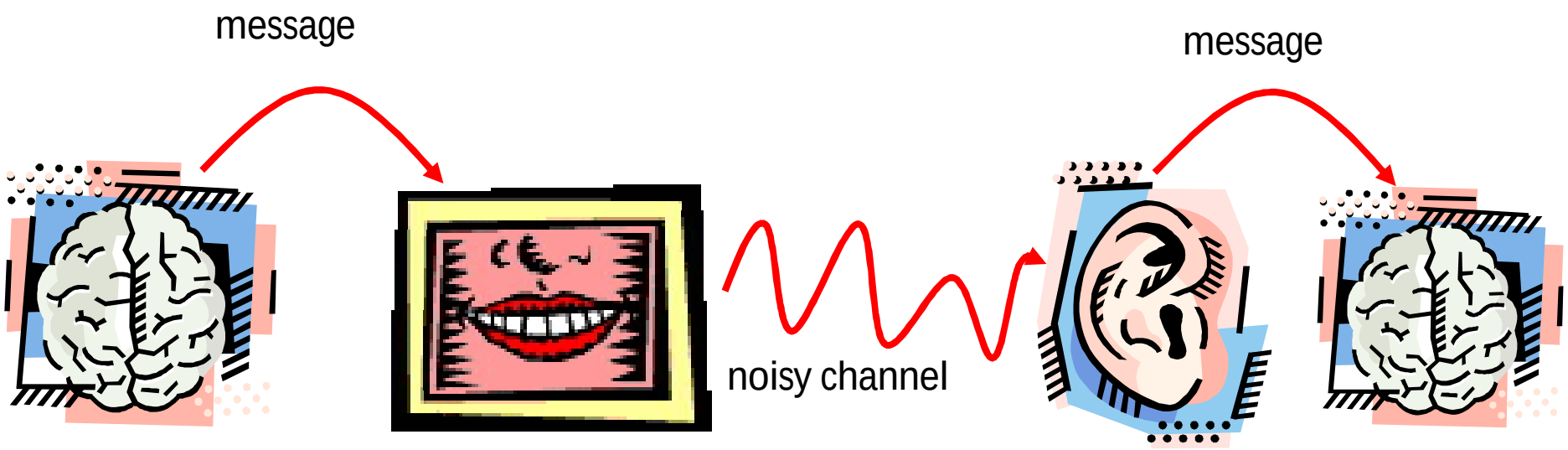
- Structure of speech
- Recognition process
- Main components of ASR
- Other used concepts
- Evaluation Criteria

# Structure of speech

- Speech is a complex phenomenon.
- A dynamic process without clearly distinguished parts
- All modern descriptions of it are probabilistic
- Speech is a continuous audio stream with dynamically changed states.

- Phone => Subword => Word => Utterance

# The Noisy Channel Model




Message + Channel =Signal

Decoding model: find Message* = argmax P(Message|Signal)
But how do we represent each of these things?

# The Noisy Channel Model: Assumptions

- What is the most likely sentence out of all sentences in the language L, given some acoustic input O?
- Treat acoustic input O as sequence of individual acoustic observations
  - $O = o_1, o_2, o_3, \ldots, o_t$
- Define a sentence W as a sequence of words:
  - $W = w_1, w_2, w_3, \ldots, w_n$

# The Noisy Channel Model:

- Probabilistic implication: Pick the highest probable sequence:

$$\hat{W} = \arg\max_{W \in L} P(W \mid O)$$

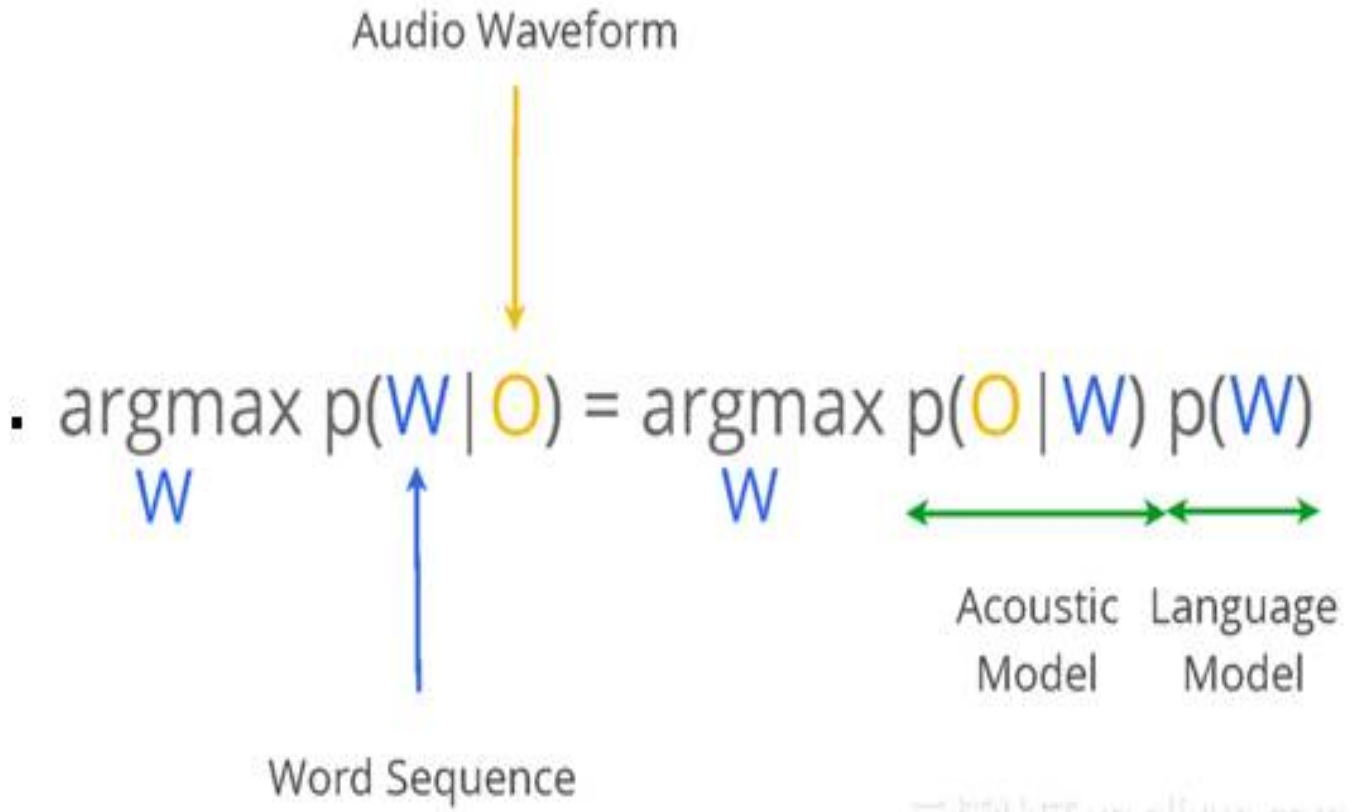
- We can use Bayes rule to rewrite this:

$$\hat{W} = \arg\max_{W \in L} \frac{P(O \mid W)P(W)}{P(O)}$$

- Since denominator is the same for each candidate sentence W, we can ignore it for the argmax:

$$\hat{W} = \arg\max_{W \in L} P(O \mid W)P(W)$$

# Math. formula representation of ASR



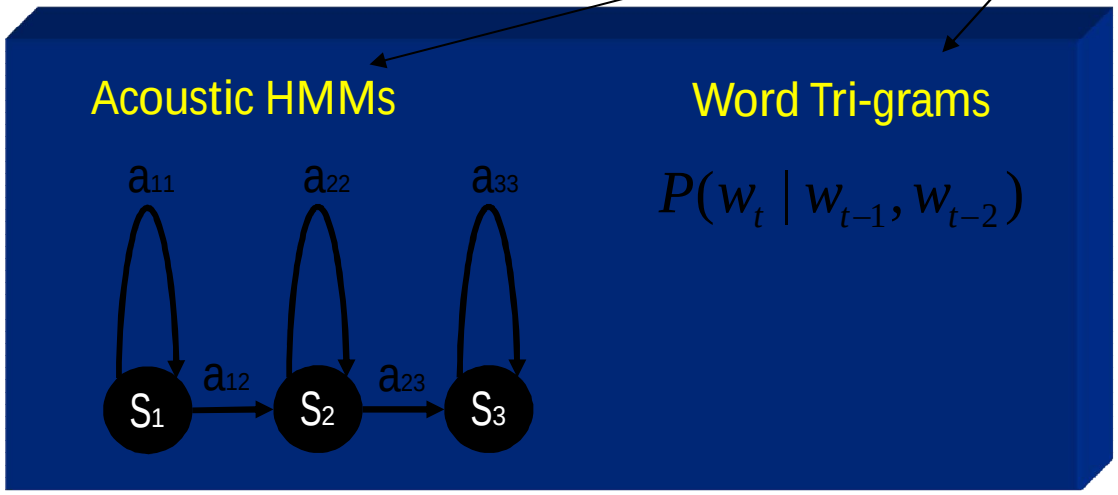
# ASR , Statistical Approach

- Based on work on Hidden Markov Models done by Leonard Baum at IDA, Princeton in the late 1960s
- Purely statistical approach pursued by Fred Jelinek and Jim Baker, IBM T.J.Watson Research

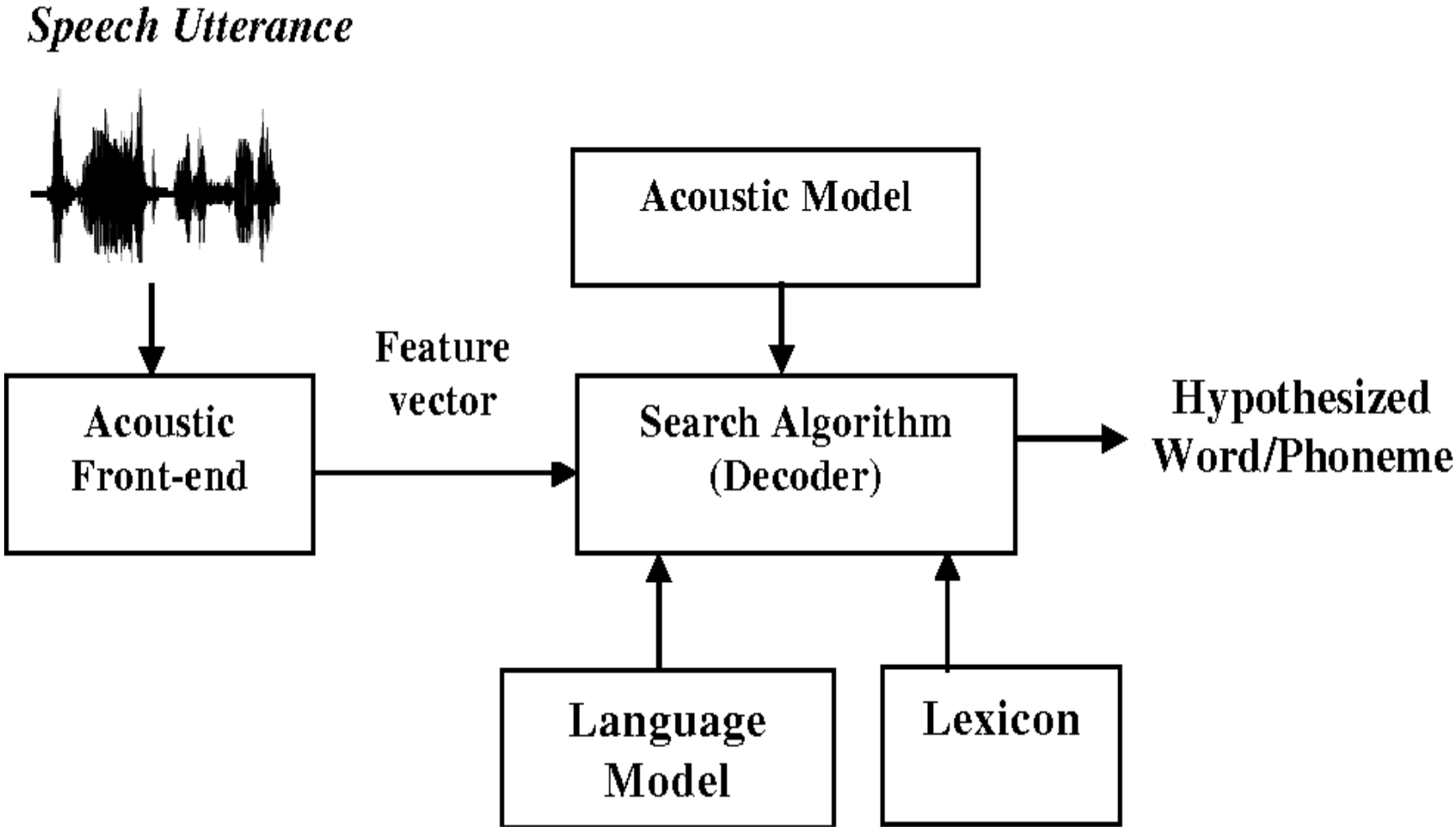
*Fred Jelinek*

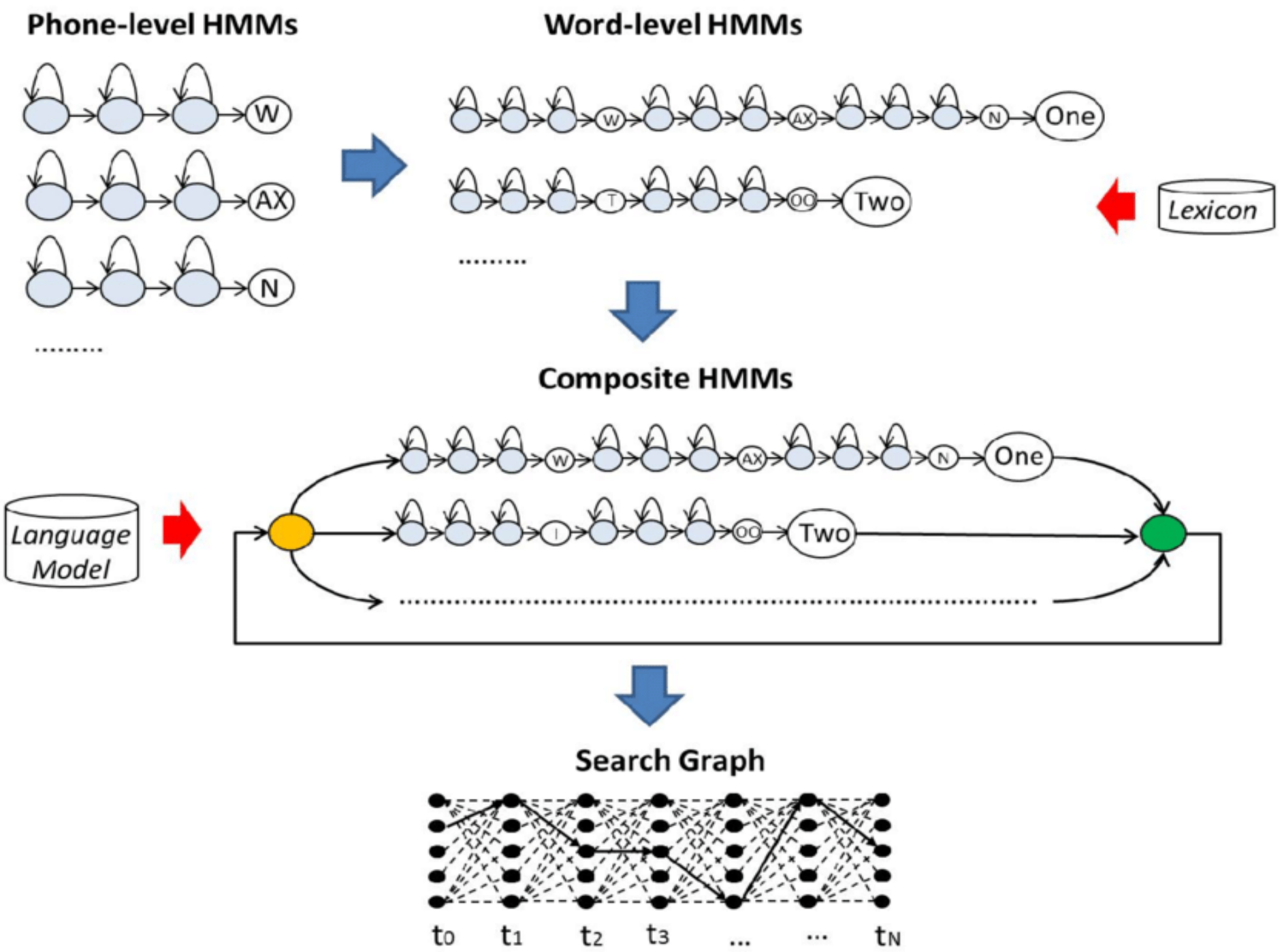
*Jim Baker*

$$\hat{W} = \arg\max_{W} P(O \mid W) P(W)$$

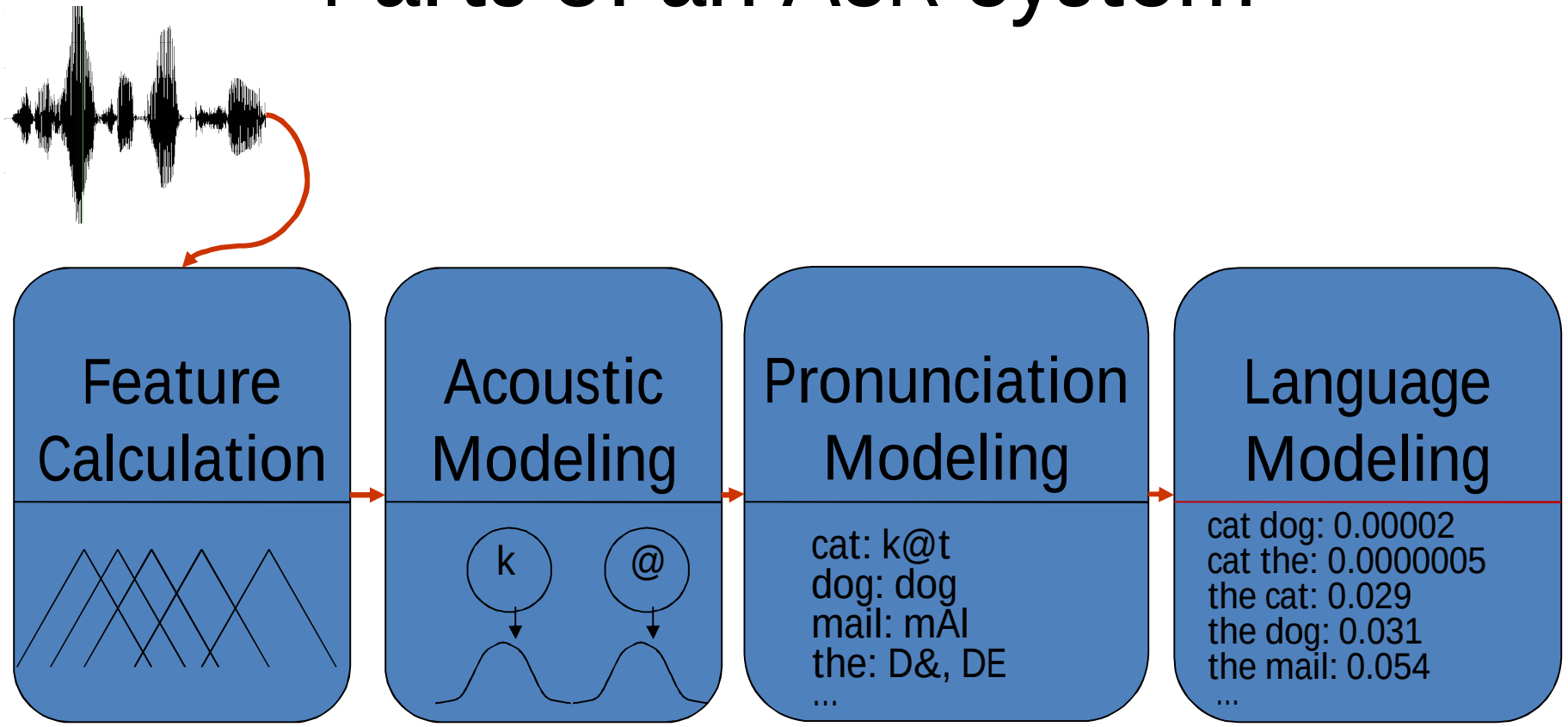


# Recognition process




Phone-level HMMs
W
AX
N
.....
Word-level HMMs
W
AX
N
One
T
OO
Two
.....
Lexicon
Composite HMMs
Language Model
W
AX
N
One
I
OO
Two
Search Graph
t0
t1
t2
t3
...
...
tN

# Parts of an ASR System




Produces acoustics ($x_t$)

Maps acoustics to phones

Maps phones to words

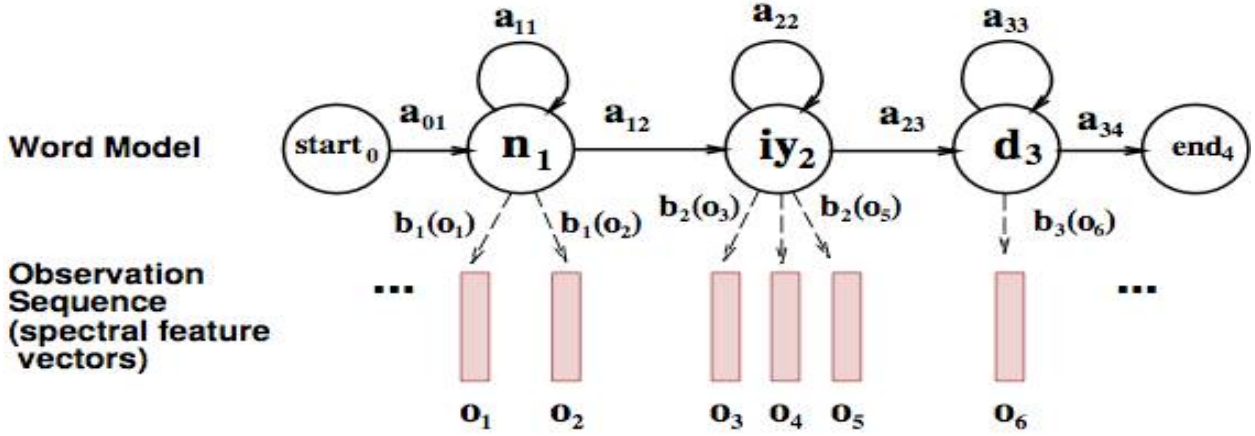
Strings words together

# Components of an ASR System

- Corpora for training and testing of components
- Feature extraction component
- Pronunciation Model (Lexicon)
- Acoustic Model
- Language Model
- Algorithms to search hypothesis space efficiently (Decoder)

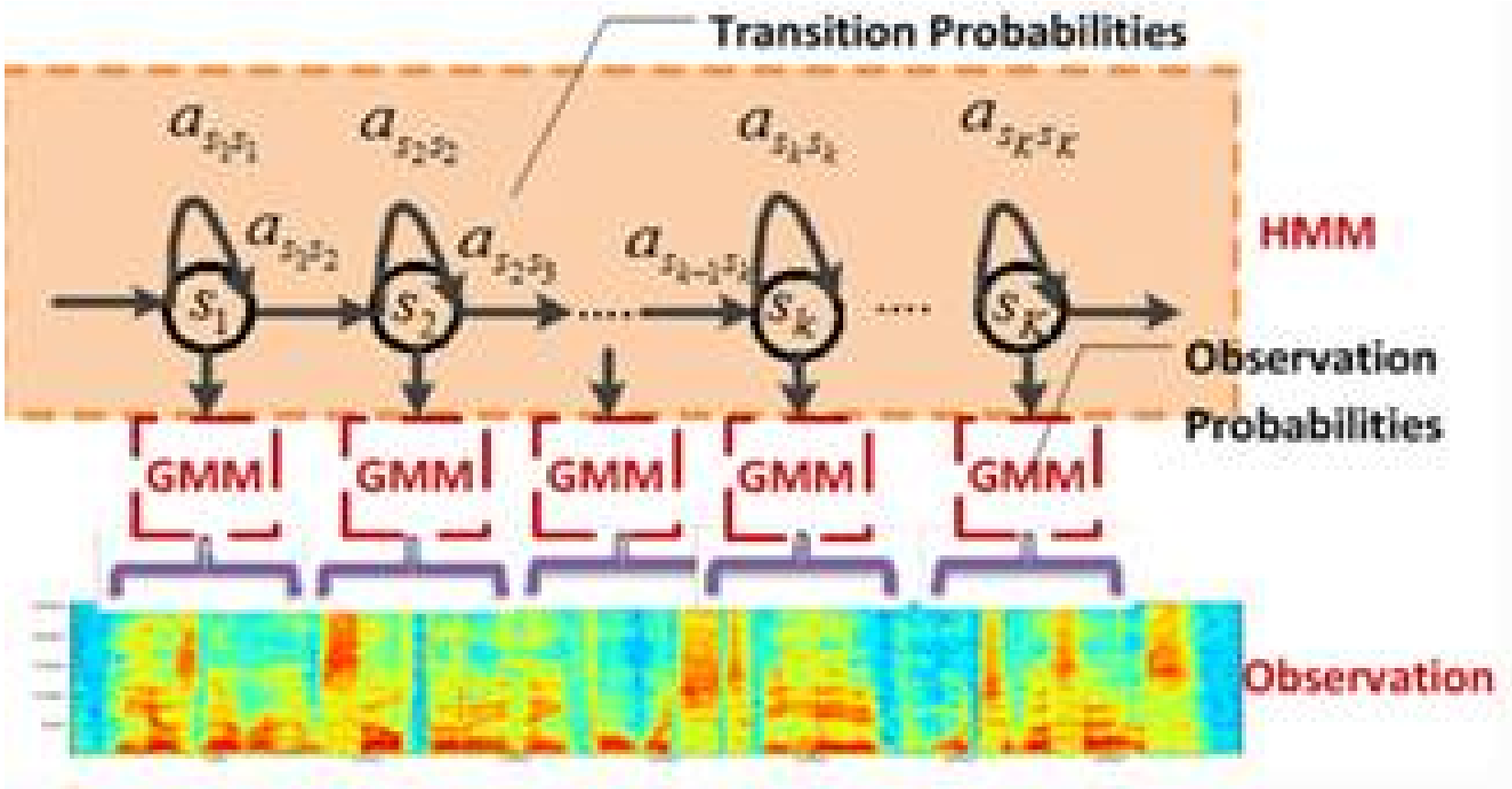
# Acoustic model

- Connection between the acoustic information and phonetics
- Three approaches:
  - HMM-GMM
  - HMM-DNN
  - Pure DNN (End-to-end approach)



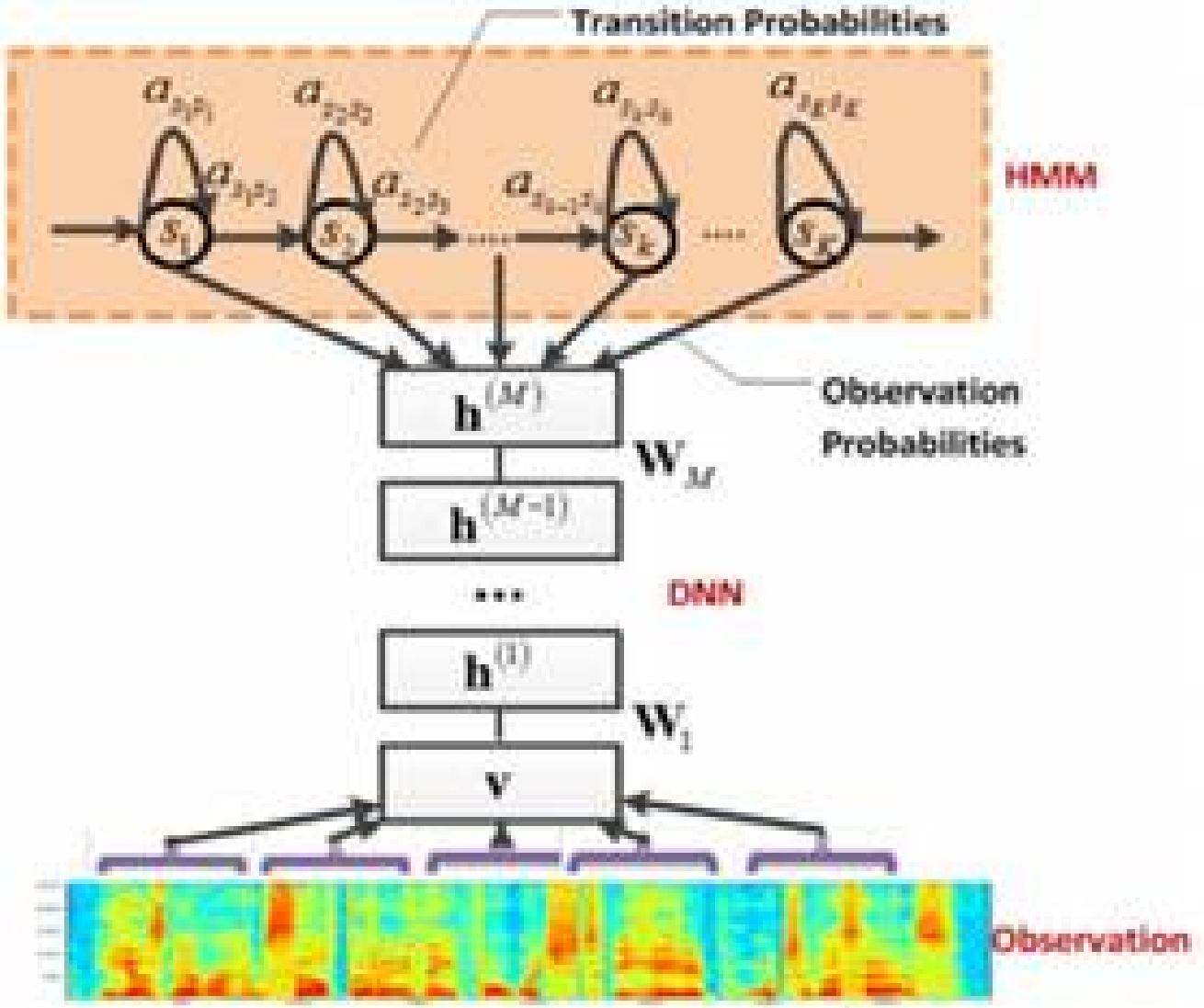
# Acoustic model

- HMM-GMM



# Acoustic model

- HMM-DNN



# End-to-End ASR System

- End-to-End (DNN),
- HMM is eliminated
- recognized word sequence directly from acoustic characteristic inputs
- Two common approaches:
  - Connectionist Temporal Classification (CTC)
  - Attention-based Encoder-Decoder

# Phonetic dictionary

- Mapping from words to phones
- It might look like this:
  - hello H EH L OW
  - world W ER L D
- Alternative pronunciations.
  - the TH IH
  - the(2) TH AH

# Language model

- A **language model** is used to restrict word search.
- The most common language model  is n-gram language model
- Text database (Corpus)

recognize speech
or
wreck a nice beach

If P(recognize speech) >P(wreck a nice beach)

Output =
“recognize speech”

# Language model Impact (Lee,1988)

- Resource Management domain
- Speaker-independent, continuous-speech
- 997 word vocabulary
- Word-pair perplexity ~ 60, Bigram ~ 20

| | No LM | Word-Pair | Bigram |
|---|---|---|---|
| % Word Error Rate | 29.4 | 6.3 | 4.2 |

# Language model Applications

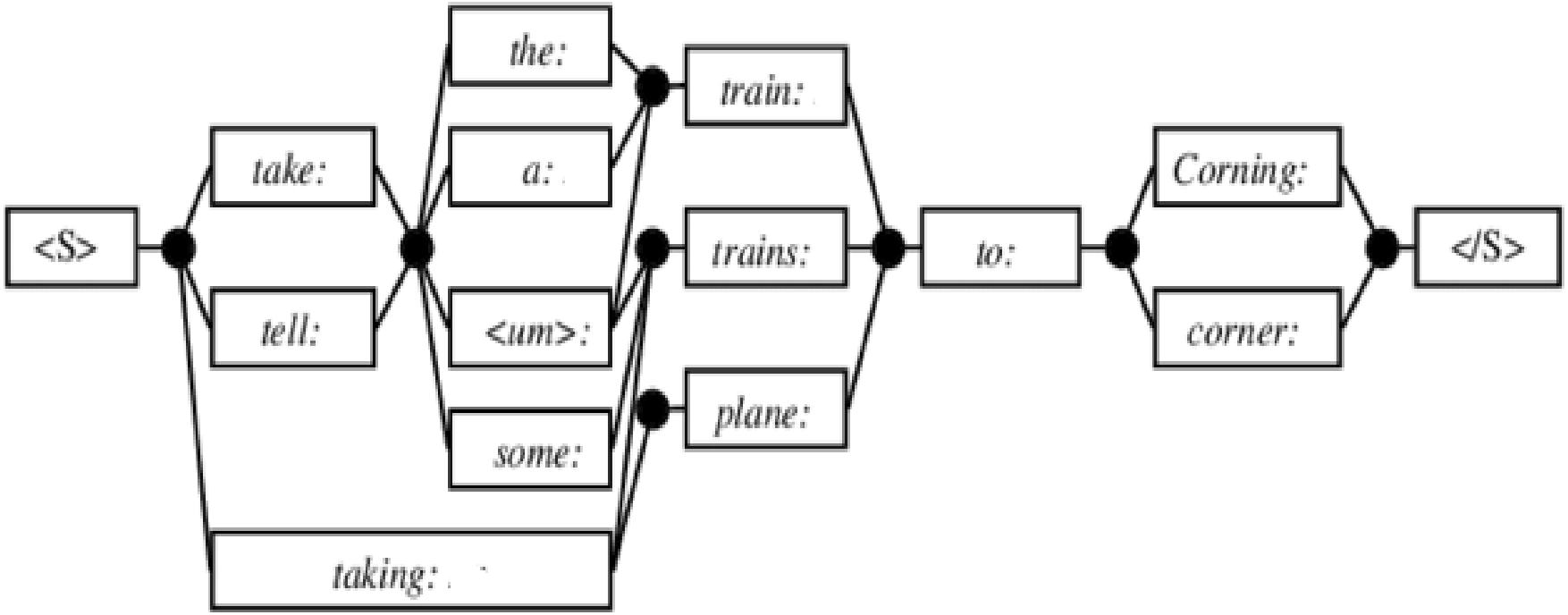
- Speech Recognition
- Machine Translation
- Optical Character Recognition (OCR) and Handwriting Recognition
- Language, Genre, Dialect and Authorship Identification
- Information Retrieval/ Search Engine
- Spell & Grammar Checker
- Predictive Keyboard

# Other used concepts

- Lattice
- N-best lists
- Speech database
- Text databases
- Criteria for evaluation

# Lattice & N-best lists

| |
|---|
| *take the train to Corning* |
| *take a train to Corning* |
| *take <um> trains to Corning* |
| *take some trains to Corning* |
| *take the plane to Corning* |
| *take the train to Cornell* |
| *taking trains to Corning* |
| *tell the train to Corning* |



# Speech & Text databases

- Speech databases :
  - Amount of data, Type (Telephone or Microphone), number of speakers, quality
  - Used to train acoustic model
  - TIMIT, RM, WSJ, Switchboard, LibriSpeech
- Text databases :
  - Amount of data, Type (Monolingual or Multilingual)
  - Used to train language model
  - Brown Corpus, British National Corpus, Google Books N-gram Corpus, American National Corpus

# Criteria for evaluation

- Word Error Rate (WER):
  - WER = (I + D + S) / N
    - S : number of substitutions
    - D : number of deletions
    - I : number of insertion
    - N : number of words in the reference
- Accuracy:
  - ACC = 1- (I + D + S) / N = 1 - WER

- Go *from* Boston *to* Washington on December 29 vs.
  Go *to* Boston *from* Washington on December 29
- 2subst/8words * 100 = 25% WER or 75% Word Accuracy

# Current Word Error Rates

| Task | **Vocabulary** | **Error (%)** |
|---|---|---|
| Digits | **11** | **0.5** |
| WSJ read speech | **5K** | **3** |
| WSJ read speech | **20K** | **3** |
| Broadcast news | **64,000+** | **10** |
| Conversational Telephone | **64,000+** | **20** |

# language model

- Count-based LMs
  - Statistical based LMs, e.g. N-gram
  - Grammar based LMs, e.g. CFG
- Continuous-space language models
  - Neural based LMs

# Grammar based LMs

- Grammar is not flexible
- Complicated to build
- Hard to modify to accommodate new data:
  - Add capability to make a reservation
  - Add capability to ask for help
  - Add ability to understand greetings
  - …
- Parsing input with large grammar is slow for real-time applications
- So…for large applications , n-gram models

# Statistical language model

- Goal: compute the probability of a sentence or sequence of words:

$$P(w_1 w_2 \ldots w_n)$$

- How to compute this joint probability:
  - Chain Rule of probability :

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

# N-gram language model

- Markov Assumption

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1})$$

- In other words, we approximate each component in the product as follows :

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-k} \ldots w_{i-1})$$

# N-gram language model

- Simplest case : unigram model

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

- Using ML estimation, probabilities are based on word counts:

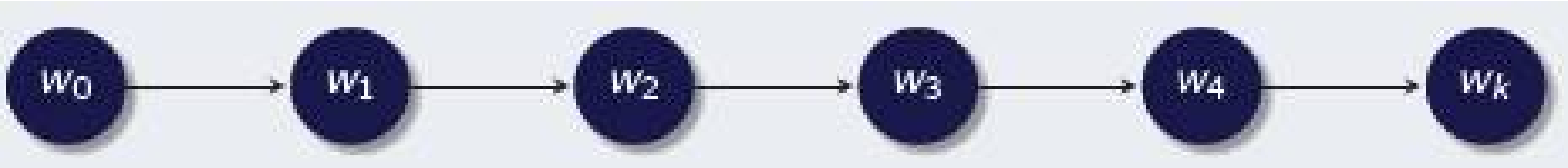
$$p(w_i) = \frac{\text{count}(w_i)}{\text{count}(w)}$$

# N-gram language model

- Bigram model
- Condition on the previous word:

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

$$p(w_i \mid w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$



# N-gram language model

- Bigram model example

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$$P(\text{I} \mid \text{<s>}) = \frac{2}{3} = .67 \qquad P(\text{Sam} \mid \text{<s>}) = \frac{1}{3} = .33 \qquad P(\text{am} \mid \text{I}) = \frac{2}{3} = .67$$

$$P(\text{</s>} \mid \text{Sam}) = \frac{1}{2} = 0.5 \qquad P(\text{Sam} \mid \text{am}) = \frac{1}{2} = .5 \qquad P(\text{do} \mid \text{I}) = \frac{1}{3} = .33$$

# N-gram language model

- One more bigram model example
- Berkeley Restaurant Project sentences

  - can you tell me about any good cantonese restaurants close by
  - mid priced thai food is what i'm looking for
  - tell me about chez panisse
  - can you give me a listing of the kinds of food that are available
  - i'm looking for a good place to eat breakfast
  - when is caffe venezia open during the day

# N-gram language model

- Raw bigram counts:

| | i | want | to | eat | chinese | food | lunch | spend |
|---|---|---|---|---|---|---|---|---|
| i | 5 | 827 | 0 | 9 | 0 | 0 | 0 | 2 |
| want | 2 | 0 | 608 | 1 | 6 | 6 | 5 | 1 |
| to | 2 | 0 | 4 | 686 | 2 | 0 | 6 | 211 |
| eat | 0 | 0 | 2 | 0 | 16 | 2 | 42 | 0 |
| chinese | 1 | 0 | 0 | 0 | 0 | 82 | 1 | 0 |
| food | 15 | 0 | 15 | 0 | 1 | 4 | 0 | 0 |
| lunch | 2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| spend | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

# N-gram language model

- Raw bigram probabilities:

| | i | want | to | eat | chinese | food | lunch | spend |
|---|---|---|---|---|---|---|---|---|
| i | 0.002 | 0.33 | 0 | 0.0036 | 0 | 0 | 0 | 0.00079 |
| want | 0.0022 | 0 | 0.66 | 0.0011 | 0.0065 | 0.0065 | 0.0054 | 0.0011 |
| to | 0.00083 | 0 | 0.0017 | 0.28 | 0.00083 | 0 | 0.0025 | 0.087 |
| eat | 0 | 0 | 0.0027 | 0 | 0.021 | 0.0027 | 0.056 | 0 |
| chinese | 0.0063 | 0 | 0 | 0 | 0 | 0.52 | 0.0063 | 0 |
| food | 0.014 | 0 | 0.014 | 0 | 0.00092 | 0.0037 | 0 | 0 |
| lunch | 0.0059 | 0 | 0 | 0 | 0 | 0.0029 | 0 | 0 |
| spend | 0.0036 | 0 | 0.0036 | 0 | 0 | 0 | 0 | 0 |

# N-gram language model

- Bigram estimates of sentence probabilities:

$$P(\text{<s> I want english food </s>}) =$$
$$P(\text{I}|\text{<s>})$$
$$\times\ P(\text{want}|\text{I})$$
$$\times\ P(\text{english}|\text{want})$$
$$\times\ P(\text{food}|\text{english})$$
$$\times\ P(\text{</s>}|\text{food})$$
$$=\ .000031$$

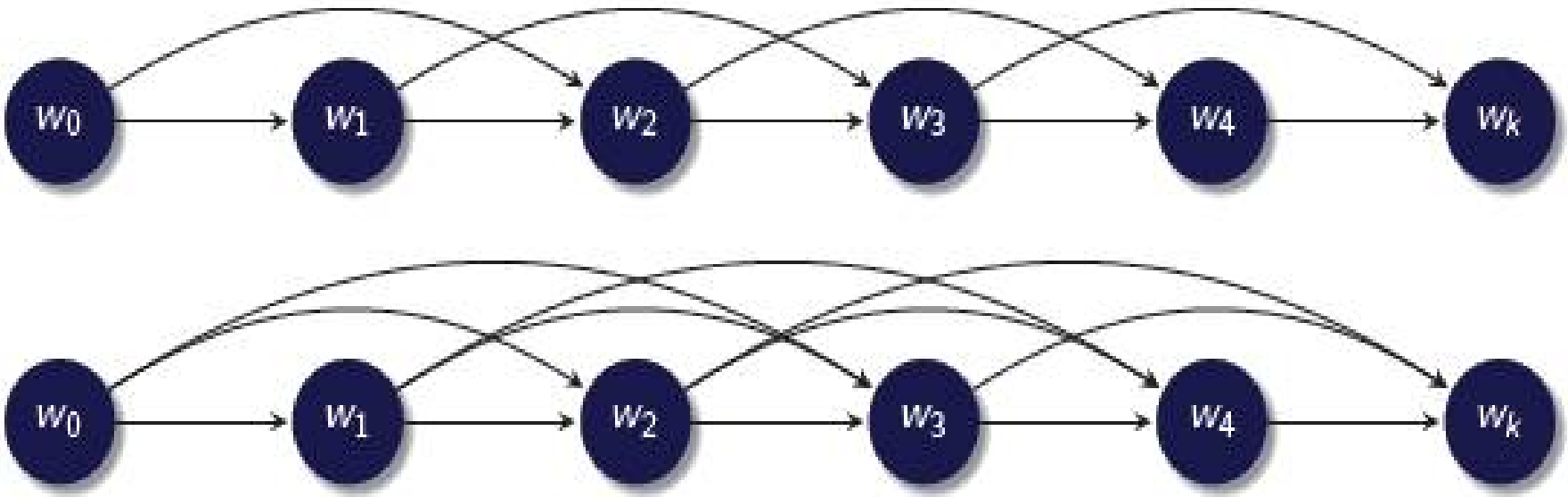
# N-gram language model

- What kinds of knowledge?
  - P(english|want) = .0011
  - P(chinese|want) = .0065
  - P(to|want) = .66
  - P(eat | to) = .28
  - P(food | to) = 0
  - P(want | spend) = 0
  - P (i | <s>) = .25

# N-gram language model

- Trigram and other n-gram LMs use a longer contiguous history:

$$p(w_i | w_{i-2}, w_{i-1}) = \frac{\text{count}(w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-2}, w_{i-1})}$$



# Building a language model

- Data collection
- Text cleanup
- Train/test separation
- Initial model estimation

$$P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1}) = \frac{\operatorname{count}(w_{i-(n-1)}, \ldots, w_{i-1}, w_i)}{\operatorname{count}(w_{i-(n-1)}, \ldots, w_{i-1})}$$

- Language model interpolation with generic model
- Language model pruning

# Language Model Evaluation

- What is a good LM?
- How to estimate how good is a LM?
- Two approaches:
  - Extrinsic Evaluation, e.g. WER
  - Intrinsic Evaluation, e.g. Perplexity
- Data split
  - Training
  - Development
  - Test

# Perplexity

- Higher Probability = Lower Perplexity = Better Prediction

$$\begin{aligned} \mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \end{aligned}$$

- Chain rule:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

- For Bigram:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}}$$

# Perplexity as branching factor

- Perplexity: as the weighted average branching factor of a language.
- Example: Let's suppose a sentence consisting of random digits
- What is the perplexity of this sentence according to a model that assign P=1/10 to each digit?

$$\begin{aligned} \mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10 \end{aligned}$$

# Perplexity

- Models trained on 38 million words from the Wall Street Journal (WSJ) using a 19,979 word vocabulary.
- Evaluate on a disjoint set of 1.5 million WSJ words.

| N-gram Order | Unigram | Bigram | Trigram |
|---|---|---|---|
| Perplexity | 962 | 170 | 109 |

# Ngram Properties

- Using of N-gram is popular because :
  - Easy to implement, estimate
  - Fast run-time computation (table lookup)
  - Can be compiled into finite-state networks and used efficient in speech recognition search (composition with pronunciation and phone models)

# Ngram Properties

- As the value of N is increased, the accuracy of the model increases.
- Considers limited size of context.
- Only works well if the test corpus looks like the training corpus (Overfitting).
- Training matrix is sparse, even for very large corpora.

- How to **Estimate** the likelihood of ***unseen n-grams***

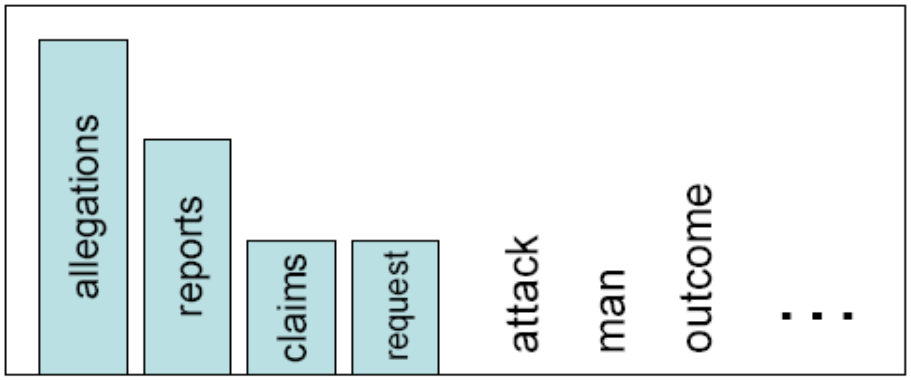
# Advanced Techniques

- N-gram models:
  - Finite approximation of infinite context history
- Issues: Zeroes and other sparseness
- Strategies: Smoothing
  - Laplace, add-K, Good-Turing, Kneser-Ney, etc
  - Use partial n-grams: interpolation, backoff
- Refinements
  - Class, cache, topic, trigger LMs

# Smoothing

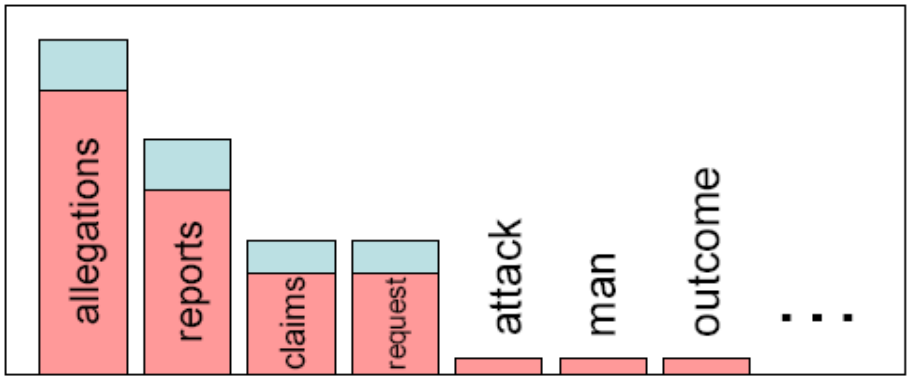
- Steal from the rich, give to the poor (probability mass)
  - We often want to make predictions from sparse statistics:

P(w | denied the)
3 allegations
2 reports
1 claims
1 request
7 total




  - Smoothing flattens spiky distributions so they generalize better

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total




  - Very important all over NLP, but easy to do badly!

Slide from Dan Klein

# Laplace smoothing

- Also called add-one smoothing
- Just add one to all the counts!
- Pretend we saw each word one more time than we did

- MLE estimate: $$P(w_i) = \frac{c_i}{N}$$

- Laplace estimate: $$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

- Reconstructed counts: $$c_i^* = (c_i + 1)\frac{N}{N + V}$$

# Laplace smoothed bigram counts

| | i | want | to | eat | chinese | food | lunch | spend |
|---|---|---|---|---|---|---|---|---|
| i | 6 | 828 | 1 | 10 | 1 | 1 | 1 | 3 |
| want | 3 | 1 | 609 | 2 | 7 | 7 | 6 | 2 |
| to | 3 | 1 | 5 | 687 | 3 | 1 | 7 | 212 |
| eat | 1 | 1 | 3 | 1 | 17 | 3 | 43 | 1 |
| chinese | 2 | 1 | 1 | 1 | 1 | 83 | 2 | 1 |
| food | 16 | 1 | 16 | 1 | 2 | 5 | 1 | 1 |
| lunch | 3 | 1 | 1 | 1 | 1 | 2 | 1 | 1 |
| spend | 2 | 1 | 2 | 1 | 1 | 1 | 1 | 1 |

# Laplace-smoothed bigrams

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

| | i | want | to | eat | chinese | food | lunch | spend |
|---|---|---|---|---|---|---|---|---|
| i | 0.0015 | 0.21 | 0.00025 | 0.0025 | 0.00025 | 0.00025 | 0.00025 | 0.00075 |
| want | 0.0013 | 0.00042 | 0.26 | 0.00084 | 0.0029 | 0.0029 | 0.0025 | 0.00084 |
| to | 0.00078 | 0.00026 | 0.0013 | 0.18 | 0.00078 | 0.00026 | 0.0018 | 0.055 |
| eat | 0.00046 | 0.00046 | 0.0014 | 0.00046 | 0.0078 | 0.0014 | 0.02 | 0.00046 |
| chinese | 0.0012 | 0.00062 | 0.00062 | 0.00062 | 0.00062 | 0.052 | 0.0012 | 0.00062 |
| food | 0.0063 | 0.00039 | 0.0063 | 0.00039 | 0.00079 | 0.002 | 0.00039 | 0.00039 |
| lunch | 0.0017 | 0.00056 | 0.00056 | 0.00056 | 0.00056 | 0.0011 | 0.00056 | 0.00056 |
| spend | 0.0012 | 0.00058 | 0.0012 | 0.00058 | 0.00058 | 0.00058 | 0.00058 | 0.00058 |

# Reconstructed counts

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times C(w_{n-1})}{C(w_{n-1})+V}$$

| | i | want | to | eat | chinese | food | lunch | spend |
|---|---|---|---|---|---|---|---|---|
| i | 3.8 | 527 | 0.64 | 6.4 | 0.64 | 0.64 | 0.64 | 1.9 |
| want | 1.2 | 0.39 | 238 | 0.78 | 2.7 | 2.7 | 2.3 | 0.78 |
| to | 1.9 | 0.63 | 3.1 | 430 | 1.9 | 0.63 | 4.4 | 133 |
| eat | 0.34 | 0.34 | 1 | 0.34 | 5.8 | 1 | 15 | 0.34 |
| chinese | 0.2 | 0.098 | 0.098 | 0.098 | 0.098 | 8.2 | 0.2 | 0.098 |
| food | 6.9 | 0.43 | 6.9 | 0.43 | 0.86 | 2.2 | 0.43 | 0.43 |
| lunch | 0.57 | 0.19 | 0.19 | 0.19 | 0.19 | 0.38 | 0.19 | 0.19 |
| spend | 0.32 | 0.16 | 0.32 | 0.16 | 0.16 | 0.16 | 0.16 | 0.16 |

# Drawback of Laplace Technique

- Causes big change to counts
  - Count of (want to) went from 608 to 238!
  - Discount (c*/c) for “Chinese food” =.10!!! A 10x reduction
- In general, Laplace is a blunt instrument
- Add-k is more fine-grained method
- Laplace smoothing not often used for N-grams, as we have much better methods.

# Better Smoothing Techniques

- Intuition: use the count of Words we've seen once to help estimate the count of Words we've never seen

- Intuition in many smoothing algorithms:
  - Good-Turing
  - Kneser-Ney (Most commonly used)
  - Witten-Bell

# Backoff and Interpolation

- Don’t try to account for unseen n-grams, just backoff to a simpler model until you’ve seen it.

- Start with estimating the trigram: P(z | x, y)
  - but C(x,y,z) is zero!
- Backoff and use info from the bigram: P(z | y)
  - but C(y,z) is zero!
- Backoff to the unigram: P(z)

- How to combine the trigram/bigram/unigram info?

# Backoff versus Interpolation

- **Backoff**: use trigram if you have it, otherwise bigram, otherwise unigram

- **Interpolation**: always mix all three
  - Simple interpolation

$$\hat{P}(w_n|w_{n-1}w_{n-2}) = \lambda_1 P(w_n|w_{n-1}w_{n-2}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n)$$

$$\sum_i \lambda_i = 1$$

# Katz Backoff

- Use the trigram probabilty if the trigram was observed:
  - P(dog | the, black) if C(“the black dog”) > 0
- “Backoff” to the bigram if it was unobserved:
  - P(dog | black) if C(“black dog”) > 0
- “Backoff” again to unigram if necessary:
  - P(dog)

# LM Interpolation

- Given two LMs, P1 and P2, get a better probability estimate by "Mixing" individual estimates:

$$P(w|h) = \lambda P_1(w|h) + (1-\lambda)P_2(w|h)$$

- λ is estimated one held-out set to minimize perplexity.

- Can be generalized to more than 2 mixture components

- Combined probability will be high if it is high in at least one of the component LMs

# LM Adaptation

- Challenge:  Need LM for new domain
  - Have little in-domain data
- Intuition: Much of language is pretty general
  - Can build from ‘general’ LM + in-domain data
- Approach: LM adaptation
  - Train on large domain independent corpus
  - Adapt with small in-domain data set
- What large corpus?
  - Web counts! e.g. Google n-grams

# LM Adaptation by Interpolation

- Interpolation (Mixing) can be used to combine a large, but poorly matched LM with a small LM that is matched to the target domain.

- Typical example:
- P1 is web LM
- P2 is a speech task domain

# Cache LMs

- Because of topical coherence the same words tend to reoccur in a stream of words
- Intuition:  Words occurred recently, more likely to reoccur again.
- Approach: Boost probabilities of recently seen words
- Implementation : interpolated LM where P2 is a unigram LM of recently seen words

# Parsing-based LMs

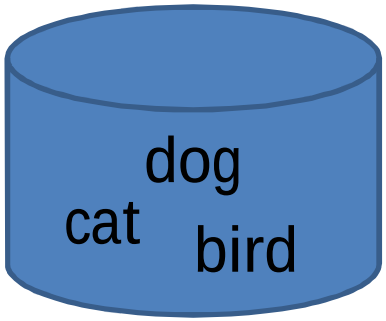
- Intuition: Current language models make no use of the syntactic properties, Example:
  - "Microsoft stock, which had been falling recently, | went up today."
  - "recently" is a bad predictor of "went up"
- Approach: Use syntactic rather than textual proximity to determine the history for predicting a word
- Implementation: Needs parsing technology to uncover syntactic structure

# Discriminative LMs

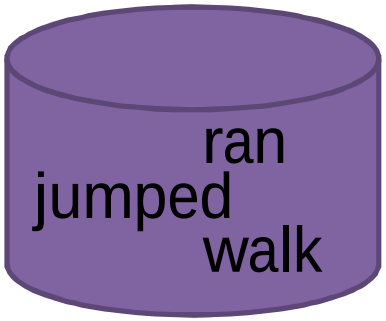
- Idea : Choose n-gram weights to improve a task, not to fit the  training set
- Standard LM estimation relies on maximizing the probability of generating the training data
- Better:  if you have a classification task (e.g., speech recognition) maximize the probability of the correct decision, ***taking other knowledge sources*** (e.g., acoustic model) ***into account***

# Class based Language Model

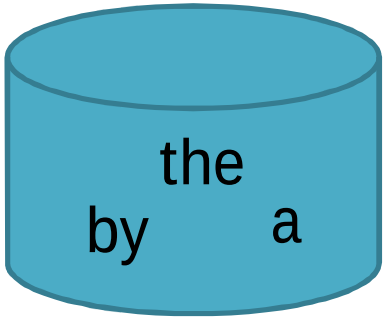
class 1: ***A***nimal




class 2: ***V***erb




class 3: ***F***unction word



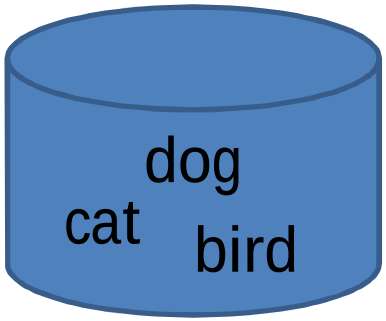

$W$ = "$w_1\ w_2\ w_3$"  $C(w_i)$: class of word $w_i$

~~$P(W) = P(w_1|START)\ P(w_2|w_1)\ P(w_3|w_2)$~~

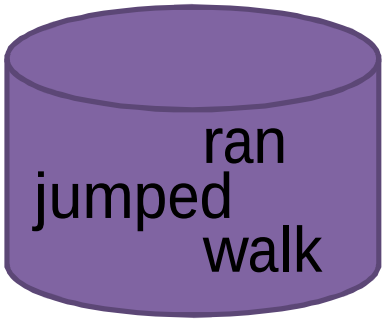
$$P(W) = P(C(w_1)|START)\ P(C(w_2)|C(w_1))\ P(C(w_3)|C(w_2))$$
$$\times\ P(w_1|C(w_1))\ P(w_2|C(w_2))\ P(w_3|C(w_3))$$

# Class based Language Model

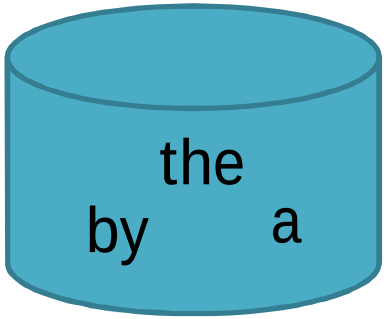
class 1: ***A***nimal




class 2: ***V***erb




class 3: ***F***unction word




W = “the dog ran”

***F*** ***A*** ***V***

P(W) = P(***F***|START) P(***A***|***F***) P(***V***|***A***)

X P(the|***F***) P(dog|***A***) P(ran|***V***)

P( class i | class j ) and P(word w| class i ) are estimated from training data.

# Class based Language Model

P( class i | class j ) and P(word w| class i ) are estimated from training data.

***Training data***

| the | dog | ran |
|---|---|---|
| *F* | *A* | *V* |

| the | cat | jumped |
|---|---|---|
| *F* | *A* | V |

W = “the cat ran”

*F* *A* *V*

P(ran|cat) is zero given the training data

However, P( ***V***erb | ***A***nimal ) is not zero

# Neural Network-based LMs

- Y. Bengio et al. (2001, 2003)
- Encode words as real-valued vectors
- Train multi-layer perceptrons (neural nets) to predict the next word from the word-codes of previous N-1 words
- Train word encoding and predictor simultaneously
- NNs learns to represent words that behave similarly with similar codes
- Good generalization to unseen N-grams

# Summary

- Language models assign a probability that a sentence is a legal string in a language.
- They are useful as a component of many NLP systems, such as ASR, OCR, and MT.
- Simple N-gram models are easy to train on unsupervised corpora and can provide useful estimates of sentence likelihood.
- MLE gives inaccurate parameters for models trained on sparse data.
- Smoothing techniques adjust parameter estimates to account for unseen words.

# Thanks

# Any Question?