Design and Implementation of Speech Recognition Systems

Spring 2011

Class 12: Continuous Speech
28 Feb 2011
Spell Checking

• I retruned and saw unnder thhe sun thet the erace is nott to the svift nor the batle to the sdragon neither yet bread to the weise nor yet riches to men of andurstendin nor yet feyvor to nen of skill but tym and chance happene to them all

• How to correct spelling?
  – For each word
    • Compare word to all words in dictionary
    • Select closest word
I retruned and saw unnder thhe sun thet therace is notto the svift northe batleto the strong neither yet bread tothe weise nor yet riches to men ofandurstendin nor yet feyvor tomen of skill but tyme and chance happeneto them all

• How to correct spelling?
  – Some words have “merged”
Spell Checking

• Irereunedandsawunnderthhesunthesun the race is not to the swift nor the battle to the strong neither yet brea d to the wise nor the richest of men of and the sturdy inn or yefey vortomen of skill but my and chance hap pen to them all

• How to correct spelling now?
A Simpler Problem

• I returned and saw under the sun that the race is not to the swift nor the battle to the strong neither to the rich nor to the wise nor to the men of understanding nor to the men of skill but to time and chance happen to them all

• Automatic introduce spaces
The Basic Spellchecker

- Compare the string to each of the words in the dictionary
The Basic Spellchecker

- Compare the string to each of the words in the dictionary

- The corresponding trellis
  - Cross products of the dictionary strings and input string
The Basic Spellchecker

- Compare the string to each of the words in the dictionary

- The corresponding trellis
  - Cross products of the dictionary strings and input string

- An equivalent trellis
  - Note the template model
The Trellis as a Product

- The Trellis is a “cross product” of the data string.
- And a model.

* U R O P *

28 Feb 2011
Continuous text: Looping around

- To model continuous text, include a loopback

Green arrows to terminating node, red arrows returning to initial node
Continuous text: Looping around

- Loopback from the end of each word
- Alignment finds word boundaries at the dummy node
Continuous text: Looping around

• To encourage (or discourage) word boundaries, assign appropriate penalties to loopback edges
  – The red edges
  – By default these are insertion edges
  – Helps decide between “Tothe” == “To The” and “Tothe” == “Tithe”
The trellis can be formed from the loopy lextree
• Loopback arcs always move forward to the next input symbol
  – Cannot represent deletions!
Continuous text with arbitrary spaces

• The methods shown so far permit checking and segmentation (into words) of text without spaces
  – E.g.
    Iretrunedandsawunnderthhesunthettheraceisnottoth esviftnorthebatletothestrong

• How about text with potentially erroneous spaces
  – E.g. I retruned and saw unnder thhe sun thet therace is notto the svift northe batleto the strong
Models with optional spaces

- Flat structure (each chain is a word)
- The spaces are optional
Models with optional spaces

- Lextree (each leaf is a word)
- The spaces are optional
Preview of Topics

• Topics so far: Isolated word recognition

• Today: continuous speech recognition, including:
  – Notion and construction of a sentence HMM
  – Review construction of search trellis from sentence HMM (or any graphical model)
  – Non-emitting states for simplifying sentence HMM construction
  – Modifying the search trellis for non-emitting states

• To cover later
  – The word-level back-pointer table data structure for efficient retrieval of the best word sequence from the search trellis
  – New pruning considerations: word beams, and absolute pruning
  – Measurement of recognition accuracy or errors
  – The generation of word lattices and N-best lists
    • The A* algorithm and the Viterbi N-best list algorithm
Isolated Word vs Continuous Speech

• A simple way to build a continuous speech recognizer:
  – Learn Templates for all possible sentences that may be spoken
  – E.g. record “delete the file” and “save all files” as separate templates
    • For a voice-based UI to an editor
  – Recognize entire sentences (no different from isolated word recognition)

• Problem: Extremely large number of sentences possible
  – Even a simple digit recognizer for phone numbers: A billion possible phone numbers!
  – Cannot record every possible phone number as template
Templates for “Sentences”

• Recording entire sentences as “templates” is a reasonable idea

• But quickly becomes infeasible as the number of sentences increases

• Inflexible: Cannot recognize sentences for which no template has been recorded
Other Issues with Continuous Speech

• Much greater variation in speaking rate
  – Having to speak with pauses forces one to speak more uniformly
  – Greater variation demands better acoustic models for accuracy

• More pronounced contextual effects
  – Pronunciation of words influenced by neighboring words
    • “Did you” -> “Dijjou”

• Spontaneous (unrehearsed) speech may include mispronunciations, false-starts, non-words (e.g. umm and ahh)
  – Need templates for all pronunciation and disfluency variants
Treat it as a series of isolated word recognition problems?

- Record only *word* templates
  - Segment recording into words, recognize individual words

- But how do we choose word boundaries?
  - Choosing different boundaries affects the results
    - E.g. “This car” or “This scar”? “The screen” or “This green”?

- Similar to reading text without spaces:

  ireturnedandsawunderthesunthattheraceisnottotheswiftnorthebattletothestrongneitheryetbreatd
  othewisenoryetrichestomenofunderstandingnoryetfavourtomenofskillbuttimeandchancehappe
  nethtothemall
Recording only Word Templates

- Brute force: Consider all possibilities
  - Segment recording in every possible way
  - Run isolated word recognition on each segment
  - Select the segmentation (and recognition) with the lowest total cost of match
    - I.e. cost of best match to first segment + cost of best match to second..

- Quickly gets very complex as the number of words increases
  - Combinatorially high number of segmentations
  - Compounded by fact that number of words is unknown
A Simple Solution

• Build/Record word templates

• Compose sentence templates from word templates

• Composition can account for all variants, disfluencies etc.
  – We will see how..
Building Sentence Templates

• Build sentence HMMs by concatenating the HMMs for the individual words
  – *e.g.* sentence “red green blue”
  
  ![Diagram: red-green-blue sentence HMM]

  – The sentence HMM looks no different from a word HMM
  – Can be evaluated just like a word HMM

• Caveat: Must have good models for the individual words
  – Ok for a limited vocabulary application
    • *E.g.* command and control application, such as robot control
Handling Silence

- People often pause between words in continuous speech
  - Often, but not always!
  - Not predictable when there will be a pause

- The composed sentence HMM fails to allow silences in the spoken input
  - If the input contained “[silence] red green [silence] blue [silence]”, it would match badly with the sentence HMM

- Need to be able to handle *optional* pauses between words
  - Like spaces between words
Optional silences can be handled by adding a *silence HMM* between every pair of words, but with a *bypass*:

- The “bypass” makes it optional: The person may or may not pause
  - If there is a pause, the best match path will go through the silence HMM
  - Otherwise, it will be bypassed

- The “silence” HMM must be separately trained
  - On examples of recordings with no speech in them (not strictly silence)
Composing HMMs for Word Sequences

• Given HMMs for word1 and word2
  – Which are both Bakis topology

![Diagram of HMMs for word1 and word2]

• How do we compose an HMM for the word sequence “word1 word2”
  – Problem: The final state in this model has only a self-transition
  – According the model, once the process arrives at the final state of word1 (for example) it never leaves
  – There is no way to move into the next word
Introducing the Non-emitting state

• So far, we have assumed that every HMM state models some output, with some output probability distribution
• Frequently, however, it is useful to include model states that do not generate any observation
  – To simplify connectivity

• Such states are called non-emitting states or sometimes null states
• NULL STATES CANNOT HAVE SELF TRANSITIONS
• Example: A word model with a final null state
HMMs with NULL Final State

- The final NULL state changes the trellis
  - The NULL state cannot be entered or exited within the word

- If there are exactly 5 vectors in word 5, the NULL state may only be visited after all 5 have been scored
The NULL final state

- The probability of transitioning into the NULL final state at any time $t$ is the probability that the observation sequence for the word will end at time $t$.
- Alternately, it represents the probability that the observation will exit the word at time $t$. 
The probability of leaving word 1 (i.e., the probability of going to the NULL state) is the same as the probability of entering word2.

- The transitions pointed to by the two ends of each of the colored arrows are the same.
Retaining a non-emitting state between words

• In some cases it may be useful to retain the non-emitting state as a connecting state
  – The probability of entering word 2 from the non-emitting state is 1.0
  – This is the only transition allowed from the non-emitting state
Retaining the Non-emitting State

HMM for word1

HMM for word2

HMM for word1

HMM for word2

HMM for the word sequence “word2 word1”
• Since non-emitting states are not associated with observations, they have no “time”
  – In the trellis this is indicated by showing them *between* time marks
  – Non-emitting states have no horizontal edges – they are always exited instantly
Viterbi with Non-emitting States

• Non-emitting states affect Viterbi decoding
  – The process of obtaining state segmentations

• This is critical for the actual recognition algorithm for word sequences
• At the first instant only the first state may be entered
• At $t=2$ the first two states have only one possible entry path
Viterbi through a Non-Emitting State

- At $t=3$ state 2 has two possible entries. The best one must be selected.
• At $t=3$ state 2 has two possible entries. The best one must be selected.
• After the third time instant we can arrive at the non-emitting state. Here there is only one way to get to the non-emitting state.
• Paths exiting the non-emitting state are now in word2
  – States in word1 are still active
  – These represent paths that have not crossed over to word2
• Paths exiting the non-emitting state are now in word2
  – States in word1 are still active
  – These represent paths that have not crossed over to word2
• The non-emitting state will now be arrived at after every observation instant.
• “Enterable” states in word2 may have incoming paths either from the “cross-over” at the non-emitting state or from within the word
  – Paths from non-emitting states may compete with paths from emitting states
• Regardless of whether the competing incoming paths are from emitting or non-emitting states, the best overall path is selected
The non-emitting state can be visited after every observation.
• At all times paths from non-emitting states may compete with paths from emitting states
Viterbi through a Non-Emitting State

- At all times paths from non-emitting states may compete with paths from emitting states
  - The best will be selected
  - This may be from either an emitting or non-emitting state
Viterbi with NULL states

• Competition between incoming paths from emitting and non-emitting states may occur at both emitting and non-emitting states

• The best path logic stays the same. The only difference is that the current observation probability is factored into emitting states

• Score for emitting state (as probabilities)

\[ P_u(s, t) = P(x_{u,t} \mid s) \max_{s'} \left( P_u(s', t-1)P(s \mid s') \mid s' \in \{\text{emitting}\}, P_u(s', t)P(s \mid s') \mid s' \in \{\text{nonemitting}\} \right) \]

• Score for non-emitting state

\[ P_u(s, t) = \max_{s'} \left( P_u(s', t-1)P(s \mid s') \mid s' \in \{\text{emitting}\}, P_u(s', t)P(s \mid s') \mid s' \in \{\text{nonemitting}\} \right) \]

• Using log probabilities

\[ \log(P_u(s, t)) = \log(P(x_{u,t} \mid s)) + \max_{s'} \left( \log(P_u(s', t-1)) + \log(P(s \mid s')) \mid s' \in \{\text{emitting}\}, \log(P_u(s', t)) + \log(P(s \mid s')) \mid s' \in \{\text{nonemitting}\} \right) \]

\[ \log(P_u(s, t)) = \max_{s'} \left( \log(P_u(s', t-1)) + \log(P(s \mid s')) \mid s' \in \{\text{emitting}\}, \log(P_u(s', t)) + \log(P(s \mid s')) \mid s' \in \{\text{nonemitting}\} \right) \]
We find the distance of the data from the “model” using the Trellis for the word
Pick the word for which this distance is lowest
Word = argmin sub word distance(data, model(word))
Using the DTW / HMM analogy
  Word = argmax sub word probability(data | model(word))
    Alternately, argmax sub word logprobability(data | model)
      Alternately still: argmin sub word -logprobability(data | model)
Speech Recognition as Bayesian Classification

• Different words may occur with different frequency
  – E.g. a person may say “SEE” much more frequently than “ZEE”

• This must be factored in
  – If we are not very sure they said “SEE” or “ZEE”, choose “SEE”
    • We are more likely to be right than if we chose ZEE

• The basic DTW equation does not factor this in
  – \( \text{Word} = \arg \max_{\text{word}} \text{probability(data | word)} \) does not account for prior bias

• Cast the problem instead as a Bayesian classification problem
  – \( \text{Word} = \arg \max_{\text{word}} p(\text{word}) \text{probability(data | word)} \)
  – “\( p(\text{word}) \)” is the \textit{a priori} probability of the word
  – Naturally accounts for prior bias
Given data $X$, find which of a number of classes $C_1, C_2, \ldots C_N$ it belongs to, based on known distributions of data from $C_1, C_2$, etc.

Bayesian Classification:

$$
\text{Class} = C_i : i = \arg\max_j \log(P(C_j)) + \log(P(X|C_j))
$$

- The \textit{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 $X$
Isolated Word Recognition as Bayesian Classification

- Classes are words
- Data are instances of spoken words
  - Sequence of feature vectors derived from speech signal

Bayesian classification:

Recognized Word = \arg\max_{word} \log(P(word)) + \log(P(X|\text{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 $P(X \mid \text{word})$

- $P(X \mid \text{word})$ is computed from the HMM for the word
  - HMMs are actually probability distributions
- Ideally $P(X \mid \text{word})$ is computed using the forward algorithm
- In reality computed as the best path through a Trellis
  - A priori probability $P(\text{word})$ is factored into the Trellis
Factoring in *a priori* probability into Trellis

The prior bias is factored in as the edge penalty at the entry to the trellis.
Time-Synchronous Trellis: Odd and Even

\[ \log(P(Odd)) \]

\[ \log(P(Even)) \]

BestPathLogProb(X, Odd)

BestPathLogProb(X, Even)

Merged final states
Time Synchronous Decode Odd and Even

• Compute the probability of best path
  – Computations can be done in the log domain. Only additions and comparisons are required

\[ \log(P(Odd)) \quad \log(P(Even)) \]

BestPathLogProb(X, Odd)
BestPathLogProb(X, Even)
Decoding to classify between *Odd* and *Even*

- Compare scores (best state sequence probabilities) of all competing words
- Select the word sequence corresponding to the path with the best score

\[
\log(P(\text{Odd})) \quad \log(P(\text{Even}))
\]
Decoding isolated words with word HMMs

• Construct a trellis (search graph) based on the HMM for each word
  – Alternately construct a single, common trellis

• Select the word corresponding to the best scoring path through the combined trellis
Why Scores and not Probabilities

• Trivial reasons
  – Computational efficiency: Use log probabilities and perform additions instead of multiplications
    • Use $\log$ transition probabilities and $\log$ node probabilities
    • Add log probability terms – do not multiply
  – Underflow: Log probability terms add – no underflow
    • Probabilities will multiply and underflow rather quickly

• Deeper reason
  – Using scores enables us to collapse parts of the trellis
  – This is not possible using forward probabilities
  – We will see why in the next few slides
Statistical classification of word sequences

- Given data $X$, find which of a number of classes $C_1, C_2, \ldots C_N$ it belongs to, based on known distributions of data from $C_1, C_2$, etc.

- Bayesian Classification:
  \[
  \text{Class} = C_i : i = \arg \max_j P(C_j)P(X|C_j)
  \]
  - Classes are word sequences
  - Data are spoken recordings of word sequences
  - Bayesian classification

\[
\text{arg max}_{wd_1, wd_2, \ldots, wd_N} \{ P(X \mid wd_1, wd_2, \ldots, wd_N) P(wd_1, wd_2, \ldots, wd_N) \}
\]

- $P(wd_1, wd_2, wd_3, \ldots)$ is a priori probability of word sequence $wd_1, wd_2, wd_3, \ldots$
  - Is the word sequence “close file” more common than “delete file”?

- $P(X \mid wd_1, wd_2, wd_3, \ldots)$ is the probability of $X$ computed on the HMM for the word sequence $wd_1, wd_2, wd_3$
  - Ideally must be computed using the forward algorithm
Decoding continuous speech

First step: construct an HMM for each possible word sequence

- $P(X | wd_1, wd_2, wd_3 ..)$ 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
  - Once again, this term must be computed by the forward algorithm
Bayesian Classification between word sequences

- Classifying an utterance as either “Rock Star” or “Dog Star”
  - Must compare $P(\text{Rock, Star})P(\text{X} | \text{Rock Star})$ with $P(\text{Dog, Star})P(\text{X} | \text{Dog Star})$
  - This is the complete forward score at the final trellis node
Bayesian Classification between word sequences

- The \textit{a priori} probability of the word sequences (P(Rock Star), P(Dog Star)) can be spread across the Trellis without changing final probabilities.
Decoding between word sequences

- In reality we find the score/cost of the best paths through the trellises.
- Not the full forward score.
- I.e. we perform DTW based classification, not Bayesian classification.

Log(P(Rock Star))  Score(Rock Star)  Log(P(Dog Star))  Score(Dog Star)

Log(P(Star | Rock))

Log(P(Star | Dog))

Log(P(Rock))

Log(P(Dog))
Time Synchronous Bayesian Classification between word sequences

\[ P(\text{Rock,Star})P(X|\text{Rock Star}) \]

\[ P(\text{Dog,Star})P(X|\text{Dog Star}) \]
Time synchronous decoding to classify between word sequences

Use best path score to determine
Decoding to classify between word sequences

The best path through *Dog Star* lies within the dotted portions of the trellis.

There are four transition points from *Dog* to *Star* in this trellis.

There are four different sets paths through the dotted trellis, each with its own best path.
Decoding to classify between word sequences

SET 1 and its best path

The best path through *Dog Star* lies within the dotted portions of the trellis.

There are four transition points from *Dog* to *Star* in this trellis.

There are four different sets paths through the dotted trellis, each with its own best path.

```
28 Feb 2011
```
Decoding to classify between word sequences

The best path through *Dog Star* lies within the dotted portions of the trellis.

There are four transition points from *Dog* to *Star* in this trellis.

There are four different sets paths through the dotted trellis, each with its own best path.
Decoding to classify between word sequences

SET 3 and its best path

The best path through *Dog Star* lies within the dotted portions of the trellis.

There are four transition points from *Dog* to *Star* in this trellis.

There are four different sets paths through the dotted trellis, each with its own best path.
Decoding to classify between word sequences

The best path through *Dog Star* lies within the dotted portions of the trellis.

There are four transition points from *Dog* to *Star* in this trellis.

There are four different sets paths through the dotted trellis, each with its own best path.
Decoding to classify between word sequences

The best path through *Dog Star* is the best of the four transition-specific best paths

\[
\text{max}(\text{dogstar}) = \max \left( \text{dogstar}_1, \text{dogstar}_2, \text{dogstar}_3, \text{dogstar}_4 \right)
\]
Decoding to classify between word sequences

Similarly, for *Rock Star* the best path through the trellis is the best of the four transition-specific best paths

\[
\text{max(rockstar)} = \text{max ( rockstar1, rockstar2, rockstar3, rockstar4 )}
\]
Decoding to classify between word sequences

Then we’d compare the best paths through *Dog Star* and *Rock Star*

\[
\begin{align*}
\max(\text{dogstar}) &= \max(\text{dogstar}_1, \text{dogstar}_2, \text{dogstar}_3, \text{dogstar}_4) \\
\max(\text{rockstar}) &= \max(\text{rockstar}_1, \text{rockstar}_2, \text{rockstar}_3, \text{rockstar}_4)
\end{align*}
\]

\[Viterbi = \max(\max(\text{dogstar}), \max(\text{rockstar}))\]
Decoding to classify between word sequences

argmax is commutative:

\[
\text{max} \left( \text{max}(\text{dogstar}), \text{max}(\text{rockstar}) \right) = \text{max} \left( \text{max}(\text{dogstar1}, \text{rockstar1}), \text{max}(\text{dogstar2}, \text{rockstar2}), \text{max}(\text{dogstar3}, \text{rockstar3}), \text{max}(\text{dogstar4}, \text{rockstar4}) \right)
\]
Max (dogstar1, rockstar1)

We can choose between Dog and Rock right here because the futures of these paths are identical.

For a given entry point the best path through STAR is the same for both trellises.

```
max(max(dogstar), max(rockstar))
= max (max(dogstar), max(rockstar))
= max (max(dogstar1, rockstar1), max(dogstar2, rockstar2), max(dogstar3, rockstar3), max(dogstar4, rockstar4))
```
Max (dogstar1, rockstar1)

We select the higher scoring of the two incoming edges here.

This portion of the trellis is now deleted.
Similar logic can be applied at other entry points to $Star$

$$\text{max}(\text{max}(\text{dogstar}), \text{max}(\text{rockstar})) = \text{max}(\text{max}(\text{dogstar1}, \text{rockstar1}), \text{max}(\text{dogstar2}, \text{rockstar2}), \text{max}(\text{dogstar3}, \text{rockstar3}), \text{max}(\text{dogstar4}, \text{rockstar4}))$$
Max (dogstar3, rockstar3)

Similar logic can be applied at other entry points to Star:

\[
\begin{align*}
  \max(\max(\text{dogstar}), \max(\text{rockstar})) &= \\
  &= \max(\max(\text{dogstar}_1, \text{rockstar}_1), \\
  &\quad \max(\text{dogstar}_2, \text{rockstar}_2), \\
  &\quad \max(\text{dogstar}_3, \text{rockstar}_3), \\
  &\quad \max(\text{dogstar}_4, \text{rockstar}_4))
\end{align*}
\]
Similar logic can be applied at other entry points to \( Star \):

\[
\max(\max(\text{dogstar}), \max(\text{rockstar})) = \\
\max(\max(\text{dogstar}_1, \text{rockstar}_1), \\
\max(\text{dogstar}_2, \text{rockstar}_2), \\
\max(\text{dogstar}_3, \text{rockstar}_3), \\
\max(\text{dogstar}_4, \text{rockstar}_4))
\]
Decoding to classify between word sequences

Similar logic can be applied at other entry points to Star.

This copy of the trellis for STAR is completely removed.
Decoding to classify between word sequences

- The two instances of *Star* can be collapsed into one to form a smaller trellis
Language-HMMs for fixed length word sequences

We will represent the vertical axis of the trellis in this simplified manner.
The Real “Classes”

- The actual recognition is DOG STAR vs. ROCK STAR
  - i.e. the two items that form our “classes” are entire phrases
- The reduced graph to the right is merely an engineering reduction obtained by utilizing commonalities in the two phrases (STAR)
  - Only possible because we use the best path score and not the entire forward probability
- This distinction affects the design of the recognition system
The word graph represents all allowed word sequences in our example
- The set of all allowed word sequences represents the allowed “language”

At a more detailed level, the figure represents an HMM composed of the HMMs for all words in the word graph
- This is the “Language HMM” – the HMM for the entire allowed language

The language HMM represents the vertical axis of the trellis
- It is the trellis, and NOT the language HMM, that is searched for the best path
Language-HMMs for fixed length word sequences

- Recognizing one of four lines from “charge of the light brigade”
  Cannon to right of them
  Cannon to left of them
  Cannon in front of them
  Cannon behind them

Each word is an HMM

- Each word is an HMM.
Where does the graph come from

• The graph must be specified to the recognizer
  – What we are actually doing is to specify the complete set of “allowed” sentences in graph form

• May be specified as an FSG or a Context-Free Grammar
  – CFGs and FSG do not have probabilities associated with them
  – We could factor in prior biases through probabilistic FSG/CFGs
  – In probabilistic variants of FSGs and CFGs we associate probabilities with options
    • E.g. in the last graph
Simplification of the language HMM through lower context language models

- Recognizing one of four lines from “charge of the light brigade”
- If we do not associate probabilities with FSG rules/transitions
Language HMMs for fixed-length word sequences: based on a grammar for Dr. Seuss

No probabilities specified – a person may utter any of these phrases at any time
Language HMMs for fixed-length word sequences: command and control grammar

Each word is an HMM

open
edit
delete
close
file
all
marked
files

No probabilities specified – a person may utter any of these phrases at any time

28 Feb 2011
Language HMMs for arbitrarily long word sequences

• Previous examples chose between a finite set of known word sequences

• Word sequences can be of arbitrary length
  – E.g. set of all word sequences that consist of an arbitrary number of repetitions of the word `bang`
    
    bang
    bang bang
    bang bang bang
    bang bang bang bang
    bang bang bang bang

    ......
  – Forming explicit word-sequence graphs of the type we’ve seen so far is not possible
    • The number of possible sequences (with non-zero \textit{a-priori} probability) is potentially infinite
    • Even if the longest sequence length is restricted, the graph will still be large
Arbitrary word sequences can be modeled with loops under some assumptions. E.g.:

- A “bang” can be followed by another “bang” with probability \( P(\text{“bang”}) \).
  - \( P(\text{“bang”}) = X \);
  - \( P(\text{Termination}) = 1-X \);
- Bangs can occur only in pairs with probability \( X \)
- A more complex graph allows more complicated patterns
- You can extend this logic to other vocabularies where the speaker says other words in addition to “bang”
  - e.g. “bang bang you’re dead”
Constrained set of word sequences with constrained vocabulary are realistic
  – Typically in command-and-control situations
    • Example: operating TV remote

– Simple dialog systems
  • When the set of permitted responses to a query is restricted

• Unconstrained word sequences: Natural Language
  – State-of-art large vocabulary decoders
  – Later in the program..
QUESTIONS?

• Next up:
  • Specifying grammars
  • Pruning
  • Simple continuous unrestricted speech
  • Backpointer table

• Any questions on topics so far?