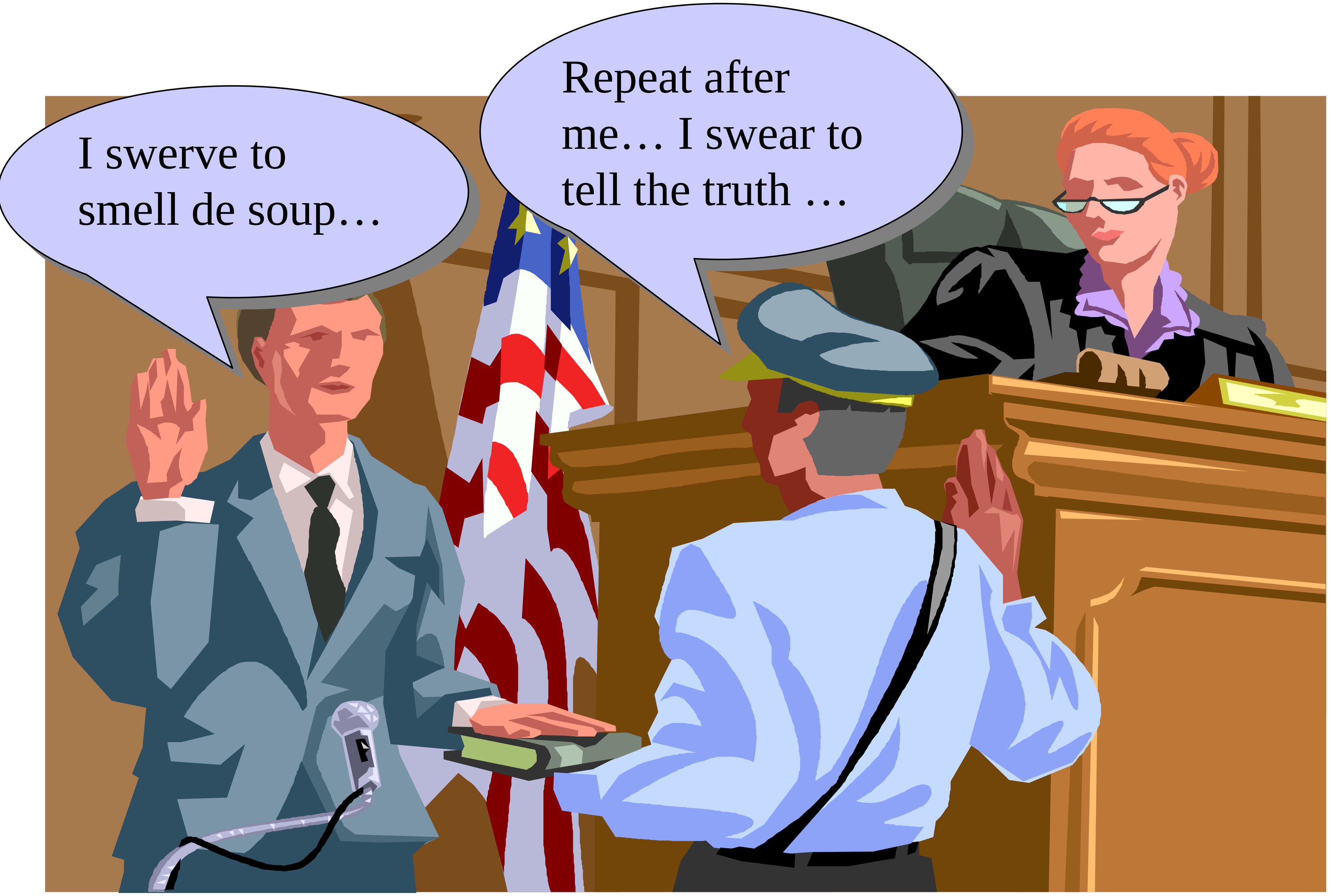
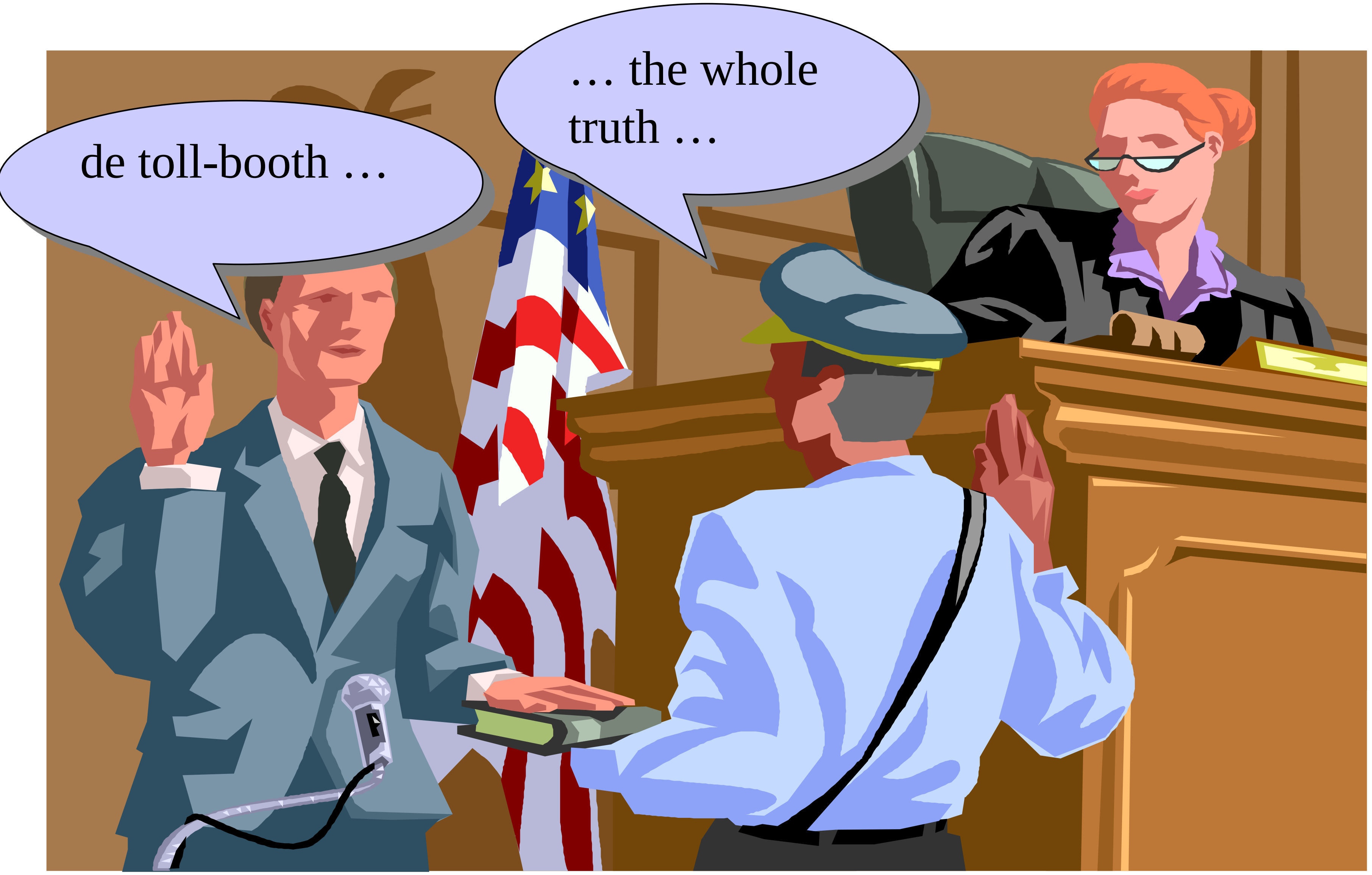
- This is our room for a four hour period .
- This is hour room four a for our . period

A Bad Language Model… From “Herman” by Jim Unger


I swerve to smell de soup…
Repeat after me… I swear to tell the truth …

A Bad Language Model… From “Herman” by Jim Unger



A Bad Language Model… From "Herman" by Jim Unger

A Bad Language Model… From “Herman” by Jim Unger

Now tell us in your own words exactly what happened.

What’s a Language Model?

- A language model is a probability distribution over word sequences
- p(“and nothing but the truth”) $\approx$ 0.001
- p(“an nuts sing on de roof”) $\approx$ 0

Where Are Language Models Used?

- Speech recognition
- Handwriting recognition
- Spelling correction
- Optical character recognition
- Machine translation
- Natural language generation
- Information retrieval
- Any problem that involves sequences ?

The statistical approach to speech recognition

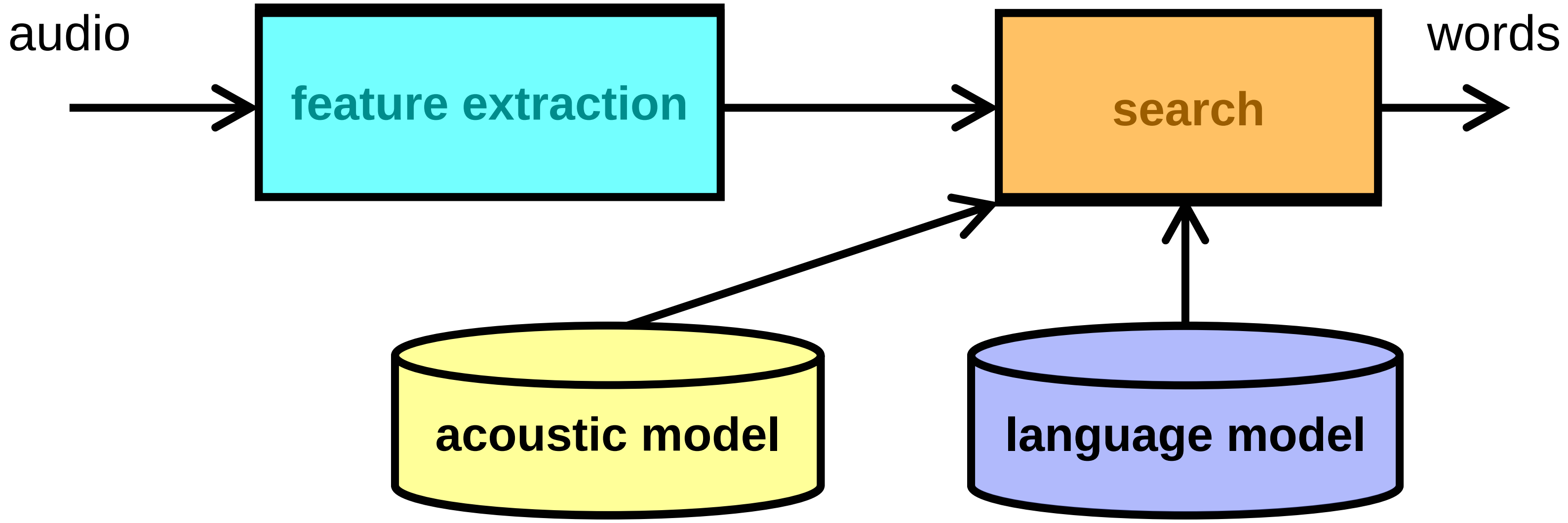
$$W^* = \underset{W}{\arg\max}\, P(W \mid X, \Theta)$$

$$= \underset{W}{\arg\max}\, \frac{P(X \mid W, \Theta) P(W \mid \Theta)}{P(X)} \qquad \text{Bayes' rule}$$

$$= \underset{W}{\arg\max}\, P(X \mid W, \Theta) P(W \mid \Theta) \qquad \text{P(X) doesn't depend on W}$$

- W is a sequence of words, W* is the best sequence.
- X is a sequence of acoustic features.
- Θ is a set of model parameters.

Automatic speech recognition – Architecture



$$W^* = \arg\max_{W} P(X \mid W, \Theta)\, P(W \mid \Theta)$$

Aside: LM Weight

- class(x) = $\text{argmax}_w \, p(w)^{\alpha} \, p(x|w)$
- … or is it the acoustic model weight? ☺
- α is often between 10 & 20
- one theory: modeling error
  - if we could estimate p(w) and p(x|w) perfectly…
  - e.g. at a given arc a_t, acoustic model assumes frames are independent

$$p(x_t,x_{t+1}|a_t=a_{t+1})=p(x_t|a_t)p(x_{t+1}|a_t)$$

LM Weight, cont'd

- another theory:
  - higher variance in estimates of acoustic model probs
  - generally $|\log p(x|w)| \gg |\log p(w)|$
  - $\log p(x|w)$ is computed by summing many more terms
  - e.g. continuous digits, $|\log p(x|w)| \cong 1000$, $|\log p(w)| \cong 20$
- Scale LM log probs in order for them not to be swamped by the AM probs
- In practice, it just works well…

Language Modeling and Domain

- Isolated digits: implicit language model

$$p(\text{"}one\text{"}) = \frac{1}{11}, p(\text{"}two\text{"}) = \frac{1}{11}, \ldots, p(\text{"}zero\text{"}) = \frac{1}{11}, p(\text{"}oh\text{"}) = \frac{1}{11}$$

- All other word sequences have probability zero
- Language models describe what word sequences the domain allows
- The better you can model acceptable/likely word sequences, or the fewer acceptable/likely word sequences in a domain, the better a bad acoustic model will look
- e.g. isolated digit recognition, yes/no recognition

Real-World Examples

- Isolated digits test set (i.e. single digits)
- Language model 1:
  - each digit sequence of length 1 equiprobable
  - probability zero for all other digit sequences
- Language model 2:
  - each digit sequence (of any length) equiprobable
  - LM 1: 1.8% error rate, LM 2: 11.5% error rate
- Point: use all of the available domain knowledge

  e.g. name dialer, phone numbers, UPS tracking numbers

How to Construct an LM

- For really simple domains:
- Enumerate all allowable word sequences
  - i.e. all word sequences w with p(w)>0
  - e.g. yes/no, isolated digits
- Use common sense to set p(w)
  - e.g. uniform distribution: p(w) = 1/vocabulary size
  - in the uniform case, ASR reduces to ML classification

$$\arg\max_{w} p(w)\,p(x \mid w) = \arg\max_{w} p(x \mid w)$$

Example

- 7-digit phone numbers

  enumerate all possible sequences:

  OH OH OH OH OH OH OH

  OH OH OH OH OH OH ONE

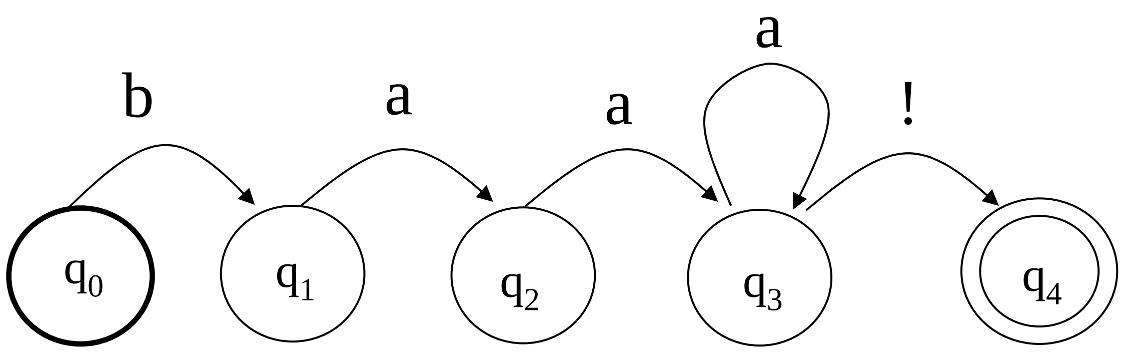
  OH OH OH OH OH OH TWO

etc.

- Is there a way we can compactly represent this list of strings?

Finite-State Automata

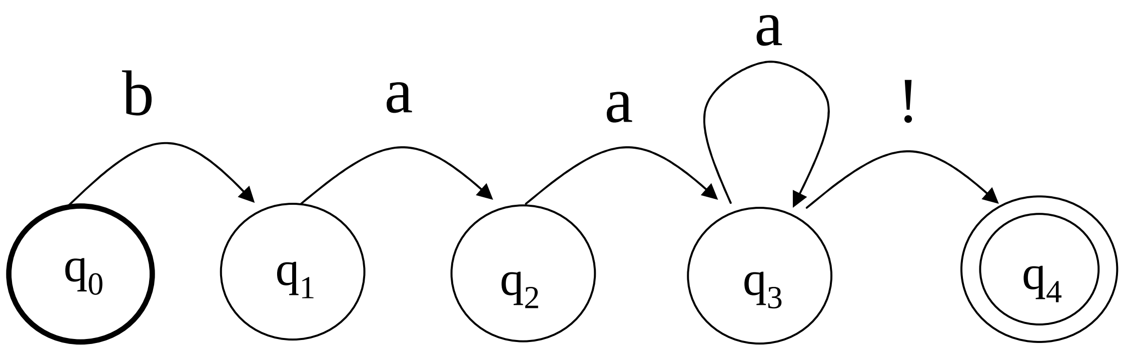
- Also called a *grammar* or *finite-state machine*
- Like a regular expression, a finite-state automaton matches or "recognizes" strings
- Any regular expression can be implemented as an FSA
- Any FSA can be described with a regular expression
- For example, the Sheep language /baa+!/ can be represented as the following FSA:



States and Transitions

A finite-state automaton consists of:

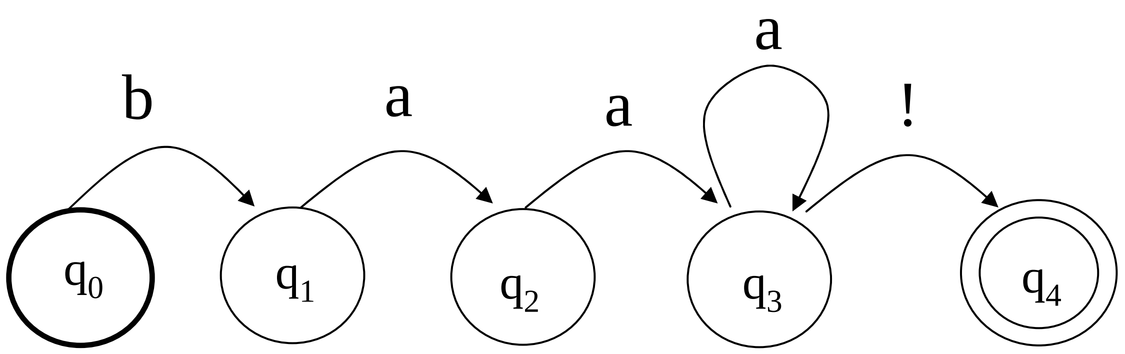
- A finite set of *states* which are represented by vertices (circular nodes) on a graph
- A finite set of *transitions*, which are represented by arcs (arrows) on a graph
- Special states:
  - The start state, which is outlined in bold
  - One or more final (accepting) states represented with a double circle



How the automaton recognizes strings

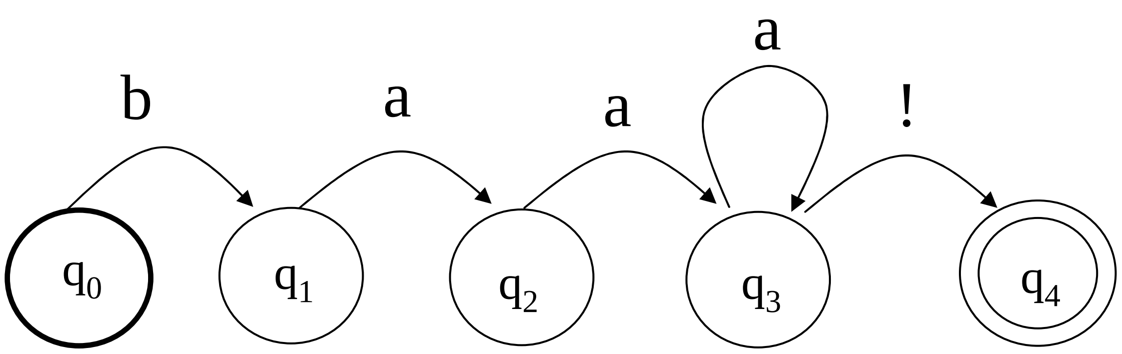
- Start in the start state q_0
- Iterate the following process:
  1. Check the next letter of the input
  2. If it matches the symbol on an arc leaving the current state, then cross that arc into the state it points to
  3. If we’re in an accepting state and we’ve run out of input, report success

Example: accepting the string baaa!



- Starting in state q_0, we read each input symbol and transition into the specified state
- The machine accepts the string because we run out of input in the accepting state

Example: rejecting the string baba!



- Start in state q_0 and read the first input symbol “b”
- Transition to state q_1, read the 2nd symbol “a”
- Transition to state q_2, read the 3rd symbol “b”
- Since there is no “b” transition out of q_2, we reject the input string

Sample Problem

- Man with a wolf, a goat, and a cabbage is on the left side of a river
- He has a small rowboat, just big enough for himself plus one other thing
- Cannot leave the goat and wolf together (wolf will eat goat)
- Cannot leave goat and cabbage together (goat will eat the cabbage)
- Can he get everything to the other side of the river?

Model

- Current state is a list of what things are on which side of the river:
- All on left MWGC-
- Man and goat on right WC-MG
- All on right (desired) -MWGC

State Transitions

- Indicate with arrows changes between states

$$\begin{tikzcd}
\text{MWGC-} \arrow[r, bend left, "g"] & \text{WC-MG} \arrow[l, bend left, "g"]
\end{tikzcd}$$

Letter indicates what happened:

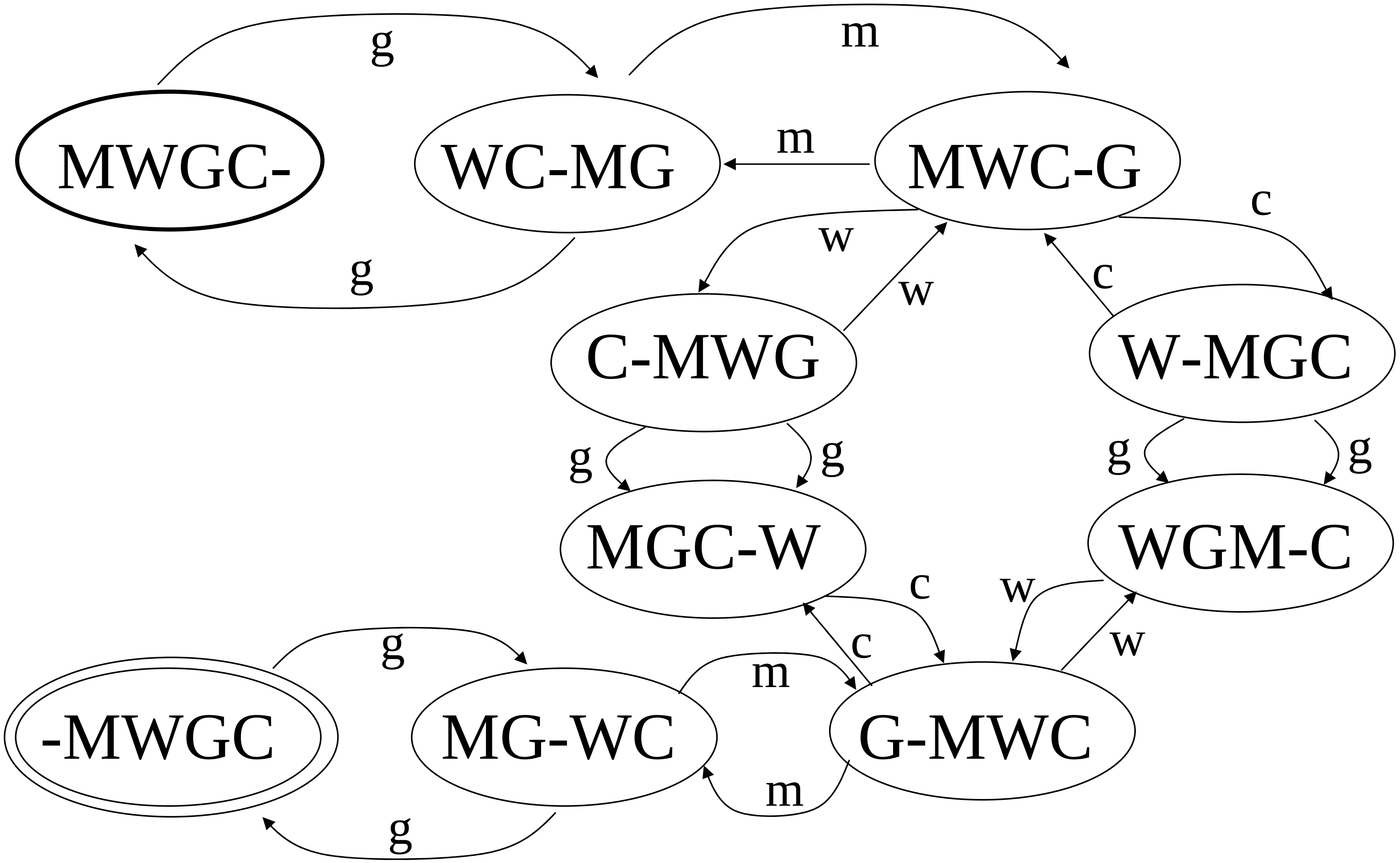
g: man took goat　　　c: man took cabbage

w: man took wolf　　　m: man went alone

Some States are Bad!

$$\begin{tikzcd}
\text{MWGC-} \arrow[r, bend left, "c"] & \text{WG-MC}
\end{tikzcd}$$

- Don’t draw those…


MWGC-
WC-MG
MWC-G
C-MWG
W-MGC
MGC-W
WGM-C
-MWGC
MG-WC
G-MWC
g
m
c
w

Finite-State Automata

- Can introduce probabilities on each path
- Probability of a path = product of probabilities on each arc along the path times the final probability of the state at the end of the path
- Probability of a word sequence is the sum of the probabilities of all paths labeled with that word sequence

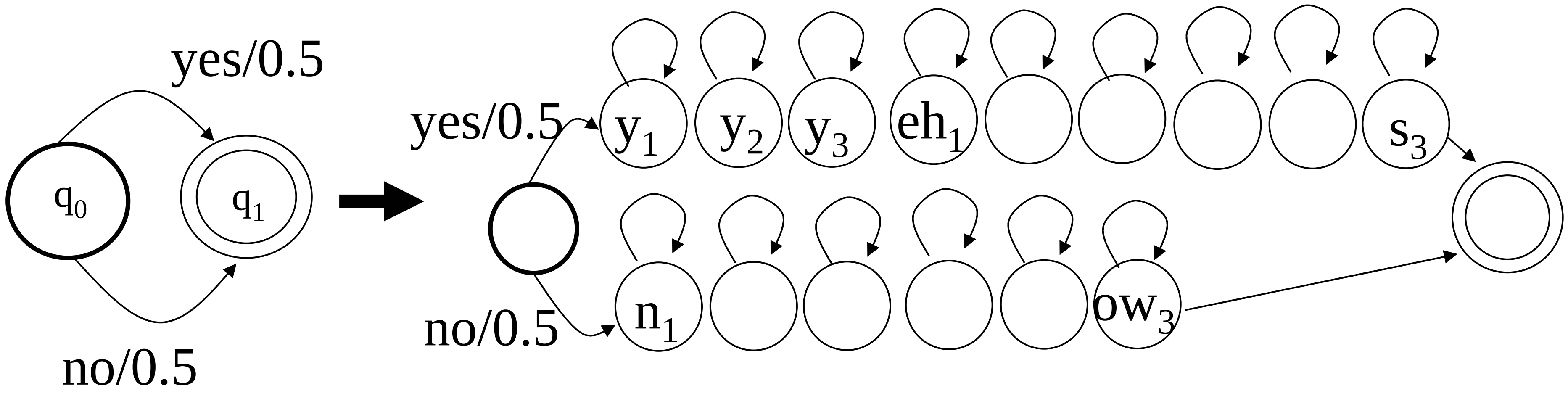
Setting Transition Probabilities in an FSA

Could use:

- common sense and intuition e.g. phone number grammar
- collect training data: in-domain word sequences
  - forward-backward algorithm
- LM training: just need text, not acoustics
  - on-line text is abundant
  - in-domain text may not be

Using a Grammar LM in ASR

- In decoding, take word FSA representing LM
- Replace each word with its HMM
- Keep LM transition probabilities
- voila!



Grammars…

- Awkward to type in FSM's
- e.g. "arc from state 3 to state 6 with label SEVEN"
- Backus-Naur Form (BNF)

  [noun phrase] → [determiner] [noun]
  [determiner] → A | THE
  [noun] → CAT | DOG

- Exercise: How to express 7-digit phone numbers in BNF?

Compiling a BNF Grammar into an FSM

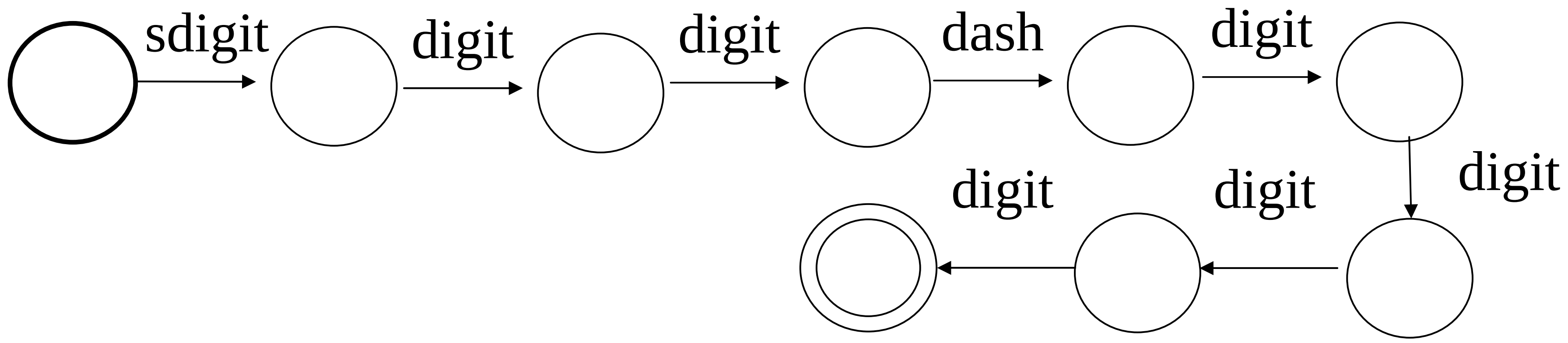
1. Express each individual rule as an FSM
2. Replace each symbol with its FSM

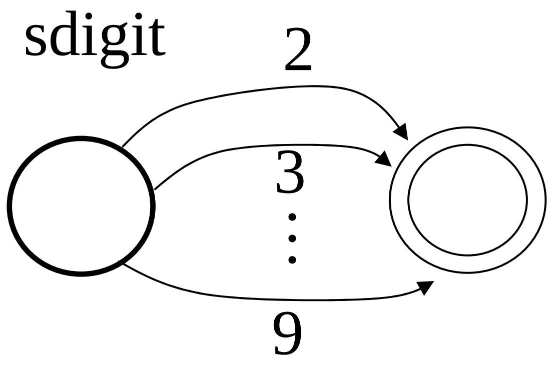
Can we handle recursion/self-reference?

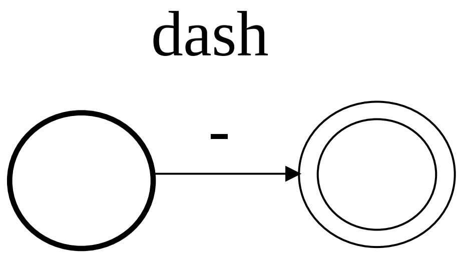
Not always possible unless we restrict the form of the rules

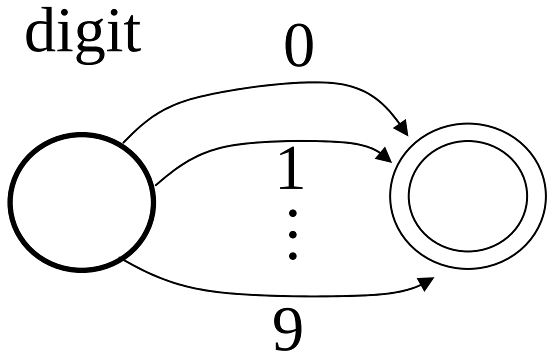
Compiling a BNF Grammar into an FSM, cont’d

7-digit phone number









Aside: The Chomsky Hierarchy

- An FSA encodes a set of word sequences
- A set of word sequences is called a *language*
- Chomsky hierarchy:
- Regular language: a language expressible by (finite) FSA
- Context-free languages: a language expressible in BNF
- {Regular languages} $\subset$ {Context-free languages}
- *e.g.* the language $a^n b^n$

  i.e.{ab, aabb, aaabbb,aaaabbbb, …} is context free but not regular

Aside: The Chomsky Hierarchy

- Is English regular? i.e. can it be expressed with an FSA?
  - probably not
- Is English context-free?
- Well, why don't we just write down a grammar for English?
  - too many rules (i.e. we're too stupid)
  - people don't follow the rules
  - machines cannot do it either

When Grammars Just Won't Do…

- Can't write grammars for complex domains
- what to do?
- goal: estimate p(w) over all word sequences w
- simple maximum likelihood?

$$p(\vec{w}) = \frac{count(\vec{w})}{\sum_{w} count(\vec{w})}$$

- can't get training data that covers a reasonable fraction of w

Vocabulary Selection

- Trade-off:
  - The more words, the more things you can confuse each word with
  - The fewer words, the more out-of-vocabulary (OOV) words you will likely encounter
  - You cannot get a word correct if it's OOV
- In practice…
  - Just choose the k most frequent words in training data
  - k is around 50,000 for unconstrained speech
  - k< 10,000 for constrained tasks

N-gram Models

- It’s hard to compute
  p(“and nothing but the truth”)
- Decomposition using conditional probabilities can help

  p(“and nothing but the truth”) = p(“and”) x
  p(“nothing”|“and”) x p(“but”|“and nothing”) x
  p(“the”|“and nothing but”) x
  p(“truth”|“and nothing but the”)

The N-gram Approximation

- Q: What’s a trigram? What’s an n-gram?
  A: Sequence of 3 words. Sequence of n words.

- Assume that each word depends only on the previous two words (or n-1 words for n-grams)

  p(“and nothing but the truth”) = p(“and”) x p(“nothing”|“and”) x p(“but”|“and nothing”) x p(“the”|“nothing but”) x p(“truth”|“but the”)

- Trigram assumption is clearly false

  p(w | of the) vs. p(w | lord of the)
- Should we just make n larger?

  can run into data sparseness problem
- N-grams have been the workhorse of language modeling for ASR over the last 30 years
- Still the primary technology for LVCSR
- Uses almost no linguistic knowledge
- *Every time I fire a linguist the performance of the recognizer improves*. Fred Jelinek (IBM, 1988)

Technical Details: Sentence Begins & Ends

$$p(w = w_1 ... w_n) = \prod_{i=1}^{n} p(w_i \mid w_{i-2} w_{i-1})$$

Pad beginning with special beginning-of-sentence token:
$w_{-1} = w_0 = \triangleright$

Want to model the fact that the sentence is ending, so pad end with special end-of-sentence token:
$wn_{+1} = \triangleleft$

$$p(w = w_1 ... w_n) = \prod_{i=1}^{n+1} p(w_i \mid w_{i-2} w_{i-1})$$

Bigram Model Example

training data:

JOHN READ MOBY DICK

MARY READ A DIFFERENT BOOK

SHE READ A BOOK BY CHER

testing data / what’s the probability of:

JOHN READ A BOOK

$$p(JOHN \mid >) = \frac{count(>\ JOHN)}{count(>)} = \frac{1}{3}$$

$$p(READ \mid JOHN) = \frac{count(JOHN \cdot READ)}{count(JOHN)} = 1$$

$$p(A \mid READ) = \frac{count(READ \cdot A)}{count(READ)} = \frac{2}{3}$$

$$p(BOOK \mid A) = \frac{count(A \cdot BOOK)}{count(A)} = \frac{1}{2}$$

$$p(< \mid BOOK) = \frac{1}{2}$$

$$p(w) = \frac{1}{3} \cdot 1 \cdot \frac{2}{3} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{2}{36}$$

Trigrams, cont’d

Q: How do we estimate the probabilities?

A: Get real text, and start counting…

Maximum likelihood estimate would say:

p(“the”|“nothing but”) =
C(“nothing but the”) / C(“nothing but”)

where C is the count of that sequence in the data

Q: Why might we want to not use the ML estimate exactly?

Data Sparseness

- Let’s say we estimate our language model from yesterday’s court proceedings
- Then, to estimate
  p(“to”|“I swear”) we use
  count (“I swear to”) / count (“I swear”)
- What about p(“to”|“I swerve”) ?

  If no traffic incidents in yesterday’s hearing,

      count(“I swerve to”) / count(“I swerve”)

= 0 if the denominator > 0, or else is undefined

Very bad if today’s case deals with a traffic incident!

Sparseness, cont'd

- Will we see all the trigrams we need to see?
- (Brown et al,1992) 350M word training set
  - in test set, what percentage of trigrams unseen? > 15%
  - i.e. in 8-word sentence, about 1 unseen trigram

$$p(w_i \mid w_{i-2} w_{i-1}) \propto count(w_{i-2} w_{i-1} w_i) = 0$$

- decoder will never choose word sequence with zero probability!
- guaranteed errors.

Life after Maximum Likelihood

- Maybe MLE isn’t such a great idea?
- (Church & Gale, 1992) Split 44M word data set into two halves
- For a bigram that occurs, say, 5 times in the first half, how many times does it occur in the 2nd half, on average?
- MLE predicts 5
- in reality, it was about 4.2
- huh?

Explanation

- Some bigrams with zero count in the first half occurred in the 2nd half
- The bigrams that did occur in the 1st half must occur slightly less frequently in the 2nd half, on average, since the total number of bigrams in each half is the same
- how can we model this phenomenon?

Maximum a Posteriori Estimation

- Let’s say I take a coin out of my pocket, flip it, and observe “heads”
- Let p(heads) = θ,   p(tails) = 1-θ
- MLE: $\theta_{mle} = \arg\max_{\theta} p(x \mid \theta) = 1$
- In reality, we believe p(heads) is around 0.5
- Instead of finding θ to maximize p(x|θ), find θ to maximize p(x|θ)p(θ)
- p(θ) is the prior probability of the parameter θ

MAP Estimation

Prior distribution: $p(\theta=0.5) = 0.99$
$p(\theta=0.000) = p(\theta=0.001) = \ldots = p(\theta=0.999) = p(\theta=1.000) = 0.00001$

Data: 1 Flip, 1 Head
$p(\theta=0.5|D) \propto p(D|\theta=0.5)p(\theta=0.5) = 0.5x.99 = 0.495$
$p(\theta=1.0|D) \propto p(D|\theta=1.0)p(\theta=1.0) = 1.0x0.00001 = 0.00001$
All other values of θ yield even smaller probabilities…
$\theta_{MAP} = 0.5$

Data: 17 Flips, 17 Heads
$p(\theta=0.5|D) \propto p(D|\theta=0.5)p(\theta=0.5) = 0.5^{17} \text{ x } 0.99 = 0.000008$
$p(\theta=1.0|D) \propto p(D|\theta=1.0)p(\theta=1.0) = 1.0\text{x } 0.00001 = 0.000010$
All other values of θ yield smaller probabilities…
$\theta_{MAP} = 1.0$

So, little data, prior has a big effect
Lots of data, prior has little effect, MAP estimate converges to ML estimate

Language Model Smoothing

- How can we adjust the ML estimates to account to account for the effects of the prior distribution when data is sparse?
- Generally, we don’t actually come up with explicit priors, but we use it as justification for *ad hoc* methods

Smoothing: Simple Attempts

- Add one: (V is vocabulary size)

$$p(z \mid xy) \approx \frac{C(xyz)+1}{C(xy)+V}$$

Advantage: Simple
Disadvantage: Works very badly

- What about delta smoothing:

$$p(z \mid xy) \approx \frac{C(xyz)+\delta}{C(xy)+V\delta}$$

A: Still bad…..

Smoothing: Good-Turing

- Basic idea: seeing something once is roughly the same as not seeing it at all
- Count the number of times you observe an event once; use this as an estimate for unseen events
- Distribute unseen events’ probability equally over all unseen events
- Adjust all other estimates downward, so that the set of probabilities sums to 1
- Several versions; simplest is to scale ML estimate by (1-prob(unseen))

Good-Turing Example

- Imagine you are fishing in a pond containing {carp, cod, tuna, trout, salmon, eel, flounder, and bass}
- Imagine you've caught: 10 carp, 3 cod, 2 tuna, 1 trout, 1 salmon, and 1 eel so far.
- Q: How likely is it that the next catch is a new species (flounder or bass)?
- A: prob(new) = prob(1's) = 3/18
- Q: How likely is it that the next catch is a bass?
- A: prob(new)x0.5 = 3/36
- Q: What's the probability the next catch is an eel?
- A: 1/18 * 15/18 = 0.046 (compared to 0.055 for MLE)

Back Off

- (Katz, 1987) Use MLE if we have enough counts, otherwise *back off* to a lower-order model

$$p_{Katz}(w_i \mid w_{i-1}) = p_{MLE}(w_i \mid w_{i-1}) \qquad \text{if count}(\text{w}_{\text{i-1}}\text{w}_\text{i}) \geq 5$$

$$= p_{GT}(w_i \mid w_{i-1}) \qquad \text{if } 1 \leq \text{count}(\text{w}_{\text{i-1}}\text{w}_\text{i}) \leq 4$$

$$= \alpha_{w_{i-1}} p_{Katz}(w_i) \qquad \text{if count}(\text{w}_{\text{i-1}}\text{w}_\text{i}) = 0$$

- choose $\alpha_{w_{i-1}}$ so that $\sum_{w_i} p_{Katz}(w_i \mid w_{i-1}) = 1$

Smoothing: Interpolation

Idea: Trigram data is very sparse, noisy,
Bigram is less so,
Unigram is very well estimated from a large corpus
Interpolate among these to get the best combination

$$p(z \mid xy) = \lambda \frac{C(xyz)}{C(xy)} + \mu \frac{C(yz)}{C(y)} + (1 - \lambda - \mu) \frac{C(z)}{C(\bullet)}$$

Find $0 < \lambda , \mu < 1$ by optimizing on "held-out" data
Can use deleted interpolation in an HMM framework

Example

- Die Possible outputs: 1,2,3,4,5,6

- Assume our training sequence is: x = 1,3,1,6,3,1,3,5
- Test sequence is: y = 5,1,3,4
- ML estimate from training:

$$\theta_m = (3/8, 0, 3/8, 0, 1/8, 1/8)$$

$$p_{\theta_m} (y) = 0$$

- Need to smooth θ_m

Example, cont'd

- Let $\theta_u = (1/6, 1/6, 1/6, 1/6, 1/6, 1/6)$

- We can construct a linear combination from θ_m and θ_u

$$\theta_s = \lambda \theta_m + (1 - \lambda) \theta_u \quad 0 <= \lambda <= 1$$

- What should the value of $1 - \lambda$ be?

- A reasonable choice is a/N, where a is a small number, and N is the training sample size

Example, cont’d

- e.g. if a=2, then $1-\lambda = 2/8 = 0.25$

$$\theta_s = 0.75\ (.375, 0, .375, 0, .125, .125)$$
$$+\ 0.25\ (.167, .167, .167, .167, .167, .167)$$

$$= \begin{array}{cccccc} (.323, & .042, & .323, & .042, & .135, & .135) \\ \downarrow & \uparrow & \downarrow & \uparrow & \downarrow & \downarrow \end{array}$$

Held-out Estimation

- Split training data into two parts:

  Part 1: $x_1^n = x_1 \; x_2 \; \dots \; x_n$

  Part 2: $x_{n+1}^N = x_{n+1} \; x_{n+2} \; \dots \; x_N$

- Estimate θ_m from part 1, combine with θ_u

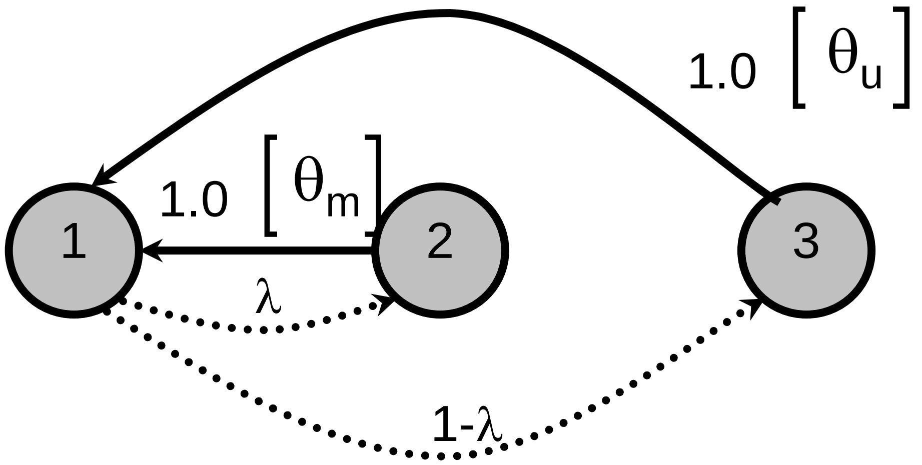
  $\theta_s = \lambda \, \theta_m + (1 - \lambda) \, \theta_u \qquad 0 <= \lambda <= 1$

- Pick λ so as to maximize the probability of Part 2 of the training data

- Q: What if we use the **same** dataset to estimate the MLE estimate θ_m and λ?

  Hint: what does MLE stand for?

- We can use the forward-backward algorithm to find the optimal λ.
- Smoothed model is equivalent to:



Example, cont'd

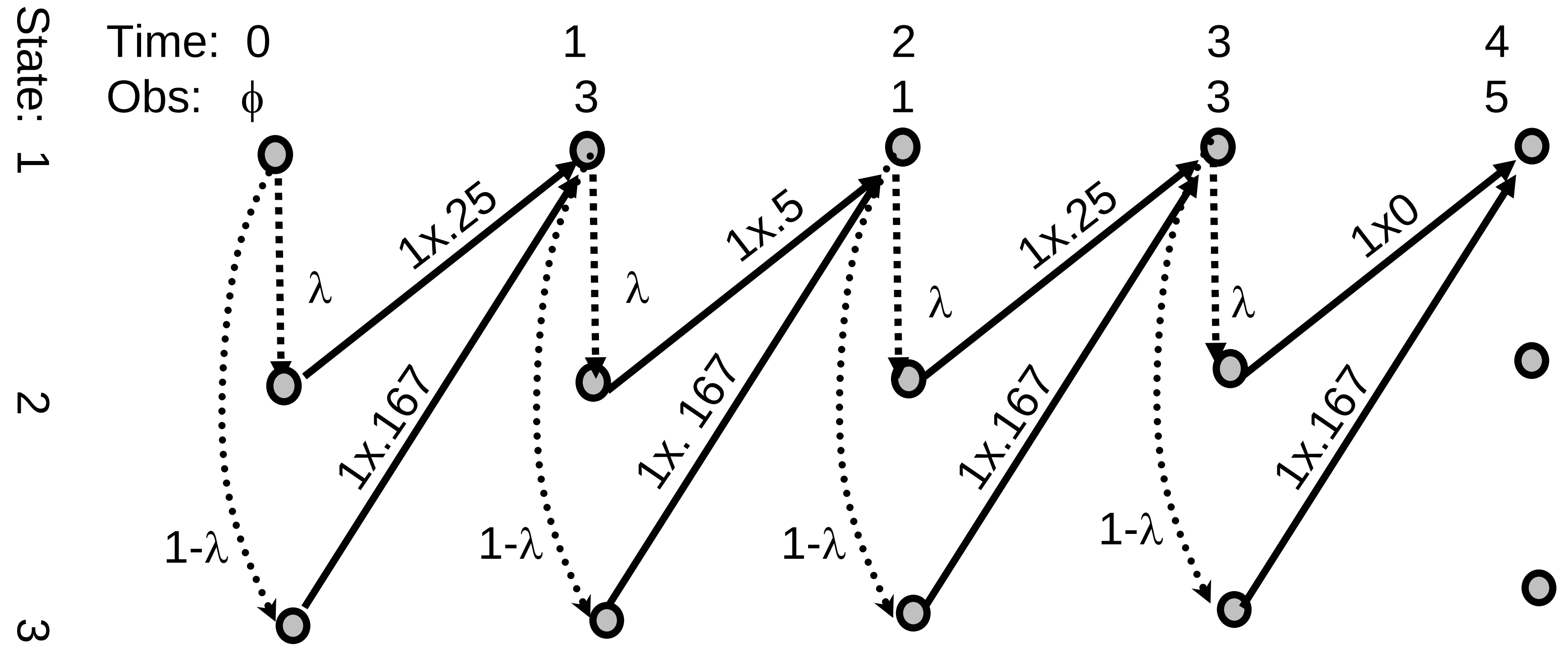
- Split training data into:

  Part 1: 1,3,1,6

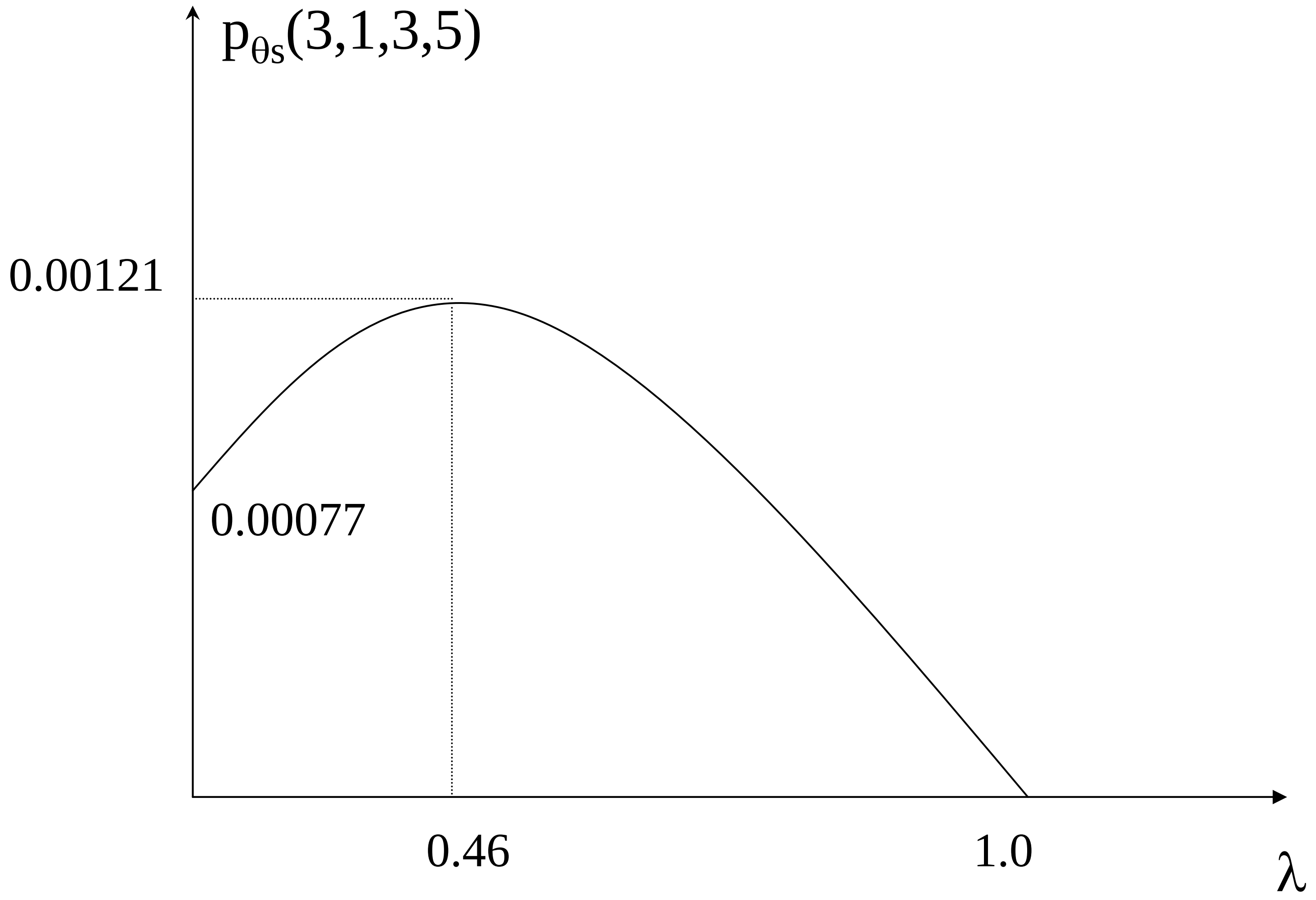
  Part 2: 3,1,3,5

In this case the ML estimate from part 1 is:

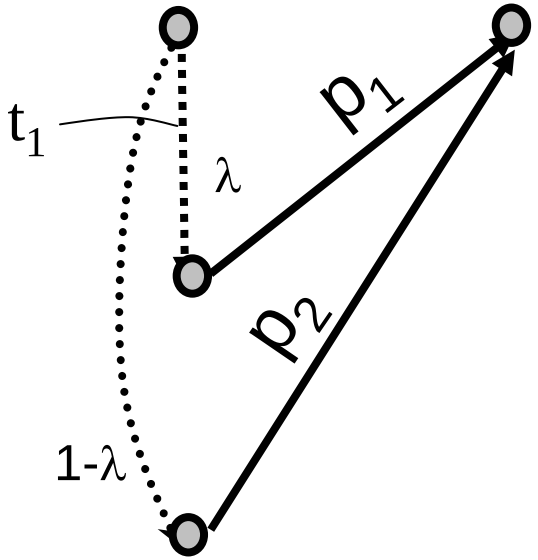
$$\theta_m = (2/4, 0, 1/4, 0, 0, 1/4)$$



$$
\begin{aligned}
p_{\theta s}(3,1,3,5) = \; & (\,.25\lambda + 0.167\,(1-\lambda)) \\
& \times (\,.5\,\lambda + 0.167\,(1-\lambda)) \\
& \times (.25\lambda + 0.167\,(1-\lambda)) \\
& \times (\,0\,\lambda + 0.167\,(1-\lambda))
\end{aligned}
$$


$p_{\theta s}(3,1,3,5)$
0.00121
0.00077
0.46
1.0
λ

- We can compute the a posteriori counts for each piece of the trellis separately



$$c(t_1 \mid x) = \frac{\lambda p_1}{\lambda p_1 + (1-\lambda)\, p_2}$$

- This is a simple form of the forward-backward algorithm

Returning to our example…

- Let's start with an initial guess $\lambda = 0.7$

	3	1	3	5	sum
$c(t_1\|x)$:	.778	.875	.778	0	2.431
$c(t_2\|x)$:	.222	.125	.222	1	1.569

New $\lambda = 2.431 / (2.431 + 1.569) = .608$

Iteration	λ	p(x)	
1	.7	.00101	
2	.608	.00114	
3	.555	.00118	
4	.523	.00120	
5	.503	.00121	
10	.467	.00121	
20	.461	.00121	
38	.460	.00121	converged

Notes

- It can be shown that log $p_{\theta s}(x_{n+1}^{N})$ is a convex I function of λ. Thus it has 1 global maximum and no other local maxima.

- Convexity result generalizes to linear combinations of more than two distributions.

- In held-out smoothing we use some of the data for estimating θ_m and some for estimating λ

- Can we use all of the data for each of the 2 purposes? Yes, if we use **deleted estimation**

Deleted Estimation

- Divide the data into L parts

$$x_1 \ldots x_{k_1} | \; x_{k_1+1} \ldots x_{k_2} | \; \ldots \; | \; x_{k_{L-1}+1} \ldots x_N$$

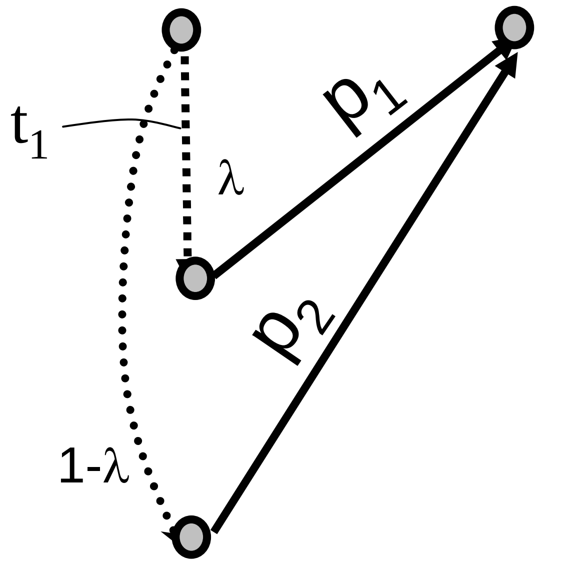
part 1 | part 2 | … | part L

- Let θ_{mL} = maximum likelihood values for the data with part L removed

- Smooth as before, using all the data for computing $p_\lambda(x)$ but:

for part 1, use $\lambda\theta_{m1} + (1-\lambda)\,\theta_u$

for part 2, use $\lambda\theta_{m2} + (1-\lambda)\,\theta_{u.}$ etc.



$$c(t_1 \mid x) = \frac{\lambda p_1}{\lambda p_1 + (1-\lambda)\,p_2}$$

- Once the optimal λ is found, we can compute θ_m from all of the data and use:

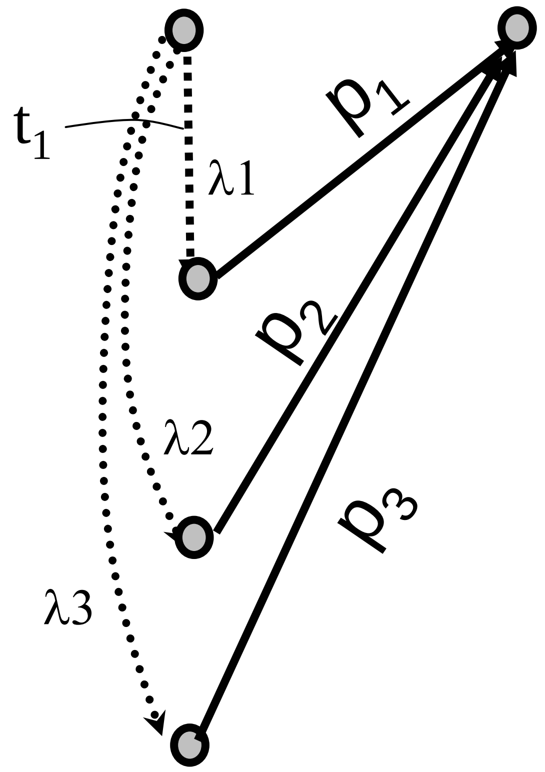
$$\theta_s = \lambda\,\theta_m + (1-\lambda)\,\theta_u$$

What about more estimators?

- e.g. we are interested in interpolating among 3-gram, bigram, and unigram models
- We can construct

  $\theta_s = \lambda_1\theta_1 + \lambda_2\theta_2 + \lambda_3\theta_3 + \ldots$

  where the λ's sum to 1



Smoothing: Kneser-Ney

- Combines back off and interpolation
- Motivation: consider bigram model
- Consider p(Francisco|eggplant)
- Assume that the bigram “eggplant Francisco” never occurred in our training data ... therefore we back off or interpolate with lower order (unigram) model
- Francisco is a common word, so both back off and interpolation methods will say it is likely
- But it only occurs in the context of “San” (in which case the bigram models it well)
- Key idea: Take the lower-order model to be the number of different contexts the word occurs in, rather than the unigram probability of the word

Smoothing: Kneser-Ney

- Subtract a constant D from all counts
- Interpolate against lower-order model which measures how many different contexts the word occurs in

$$p(z \mid xy) = \frac{C(xyz) - D}{C(xy)} + \lambda \frac{C(\cdot z)}{\sum C(\cdot z)}$$

- Modified K-N Smoothing: make D a function of the number of times the trigram xyz occurs

So, which technique to use?

- Empirically, interpolation is superior to back off
- State of the art is Modified Kneser-Ney smoothing (Chen & Goodman, 1999)

Does Smoothing Matter?

- No smoothing (MLE estimate):
  - Performance will be very poor
  - Zero probabilities will kill you
- Difference between bucketed linear interpolation (ok) and modified Kneser-Ney (best) is around 1% absolute in word error rate for a 3-gram model
- No downside to better smoothing (except in effort)
- Differences between best and suboptimal become larger as model order increases

Word Error Rate

- How do we measure the performance of an ASR system?
- Define WER = (substitutions + deletions+ insertions) / (number of words in reference script)
- Example:

deletion

ref: The dog is here now

hyp: The uh bog is now

insertion substitution

WER = 3/5 = 60%

- Compute WER efficiently using dynamic programming (DTW)
- Can WER be above 100% ?

Model Order

- Should we use big or small models?

  e.g. 3-gram or 5-gram?
- With smaller models, less sparse data issues → better probability estimates?
  - Empirically, bigger is better
  - With best smoothing, little or no performance degradation if model is too large
  - With lots of data (100M words +) significant gain from 5-gram
- Limiting resource: disk/memory
- Count cutoffs can be used to reduce the size of the LM
- Discard all n-grams with count less than threshold

Evaluating Language Models

- Best way: plug into ASR system, see how LM affects WER
  - Expensive to compute
- Is there something cheaper that predicts WER well?
  - “perplexity” (PP) of test data (only needs text)
  - Doesn’t always predict WER well, but has theoretical significance
  - Predicts best when 2 LM’s being compared are trained on same data

Perplexity

- Perplexity is average branching factor, i.e. how many alternatives the LM believes there are following each word
- Another interpretation: $\log_2$PP is the average number of bits per word needed to encode the test data using the model P( )

- Ask a speech recognizer to recognize digits: 0,1,2,3,4,5,6,7,8,9 simple task (?) perplexity = 10
- Ask a speech recognizer to recognize alphabet: a,b,c,d,e,…z

  more complex task … perplexity = 26
- alpha, bravo, charlie … yankee, zulu

  perplexity = 26

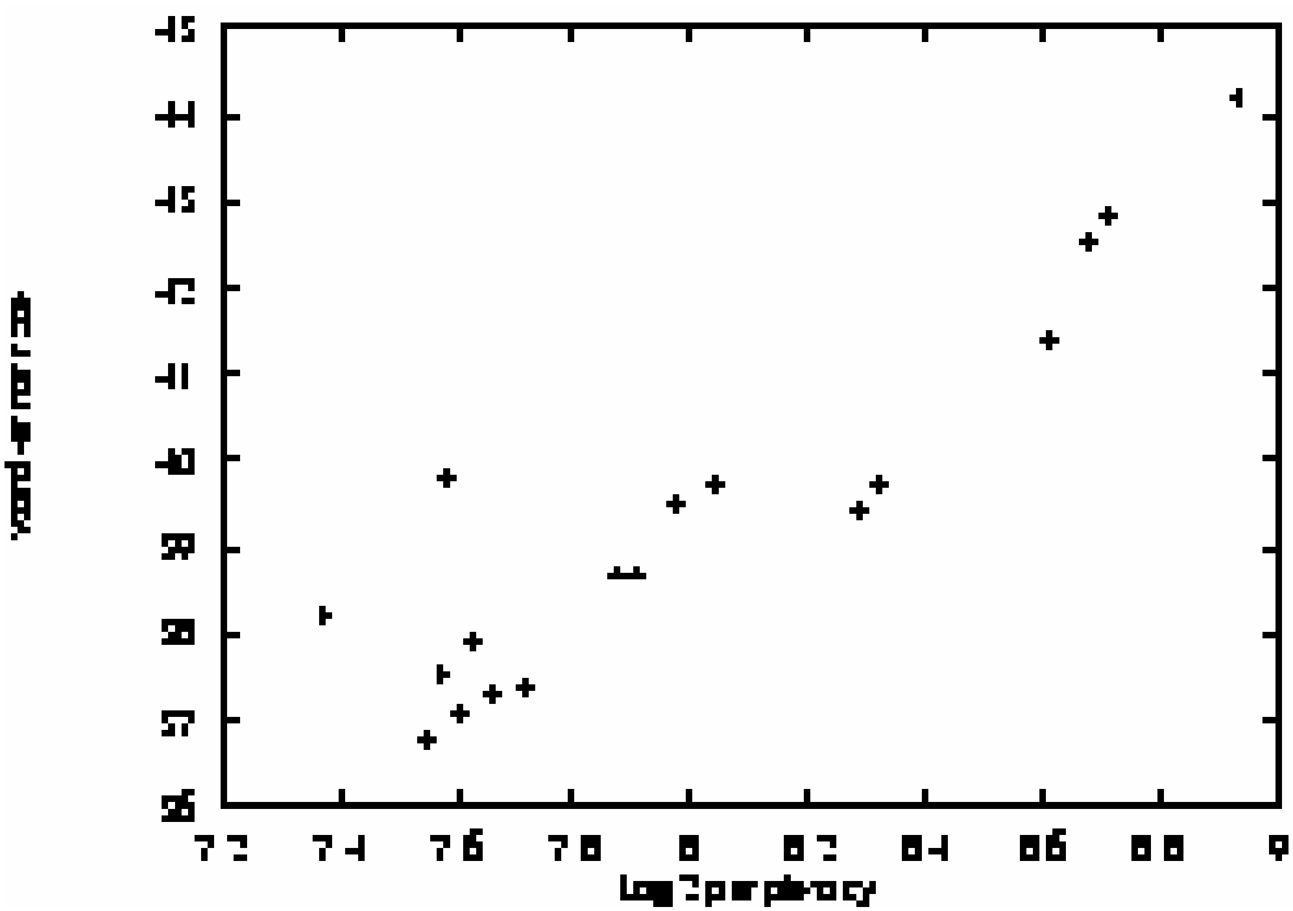
Perplexity measures LM difficulty, not acoustic difficulty

Computing Perplexity

1. Compute the geometric average probability assigned to each word in test data $w_1..w_n$ by model P( )

$$p_{avg} = [\prod_{i=1}^{n} P(w_i \mid w_1 ... w_{i-1})]^{\frac{1}{n}}$$

2. Invert it: PP = $1/p_{avg}$


45
44
43
42
41
40
39
38
37
36
7.2
7.4
7.6
7.8
8
8.2
8.4
8.6
8.8
9
Log2 perplexity
word error rate

Course Feedback

- Was this lecture mostly clear or unclear?
- What was the muddiest topic?
- Comments on difficulty of labs?
- Other feedback (pace, content, atmosphere)?