Agenda

- Language modeling overview
- N-gram (Markov) models and smoothing (regularization)
- Specific smoothing methods: Good-Turing; Katz; Jelinek Mercer; Absolute discounting; Witten Bell; Kneser-Ney
- Weighted finite-state automata encoding
- Whole sentence language modeling
  - Log linear models
  - Syntactic language models
  - Neural net models

Language models for ASR

- For a vocabulary Σ, we want to find the word string $\mathrm{w} \in \Sigma^*$ that is the best transcription for acoustic signal A

$$\widehat{\mathrm{w}} = \underset{\mathrm{w} \in \Sigma^*}{\operatorname{argmax}} \mathrm{P}(\mathrm{w} \mid A)$$

  Through the use of Bayes rule, this is the same as

$$\widehat{\mathrm{w}} = \underset{\mathrm{w} \in \Sigma^*}{\operatorname{argmax}} \mathrm{P}(A \mid \mathrm{w})\ \mathrm{P}(\mathrm{w})$$

- The $P(\mathrm{w})$ part (the language model) assigns probabilities to strings of words $\mathrm{w} = w_1 \ldots w_k$ for some k
- Can be thought of as a sort of prior over strings
  - Choose between acoustically confusable strings

Other LM applications

- Other noisy channel formulations
  - Machine translation (LM prior over possible translations)
  - Optical character recognition (prior over character sequences)
- Other uses for language models
  - LM-derived scores for information retrieval
  - Language models for word disambiguation in text entry (T9)
  - For coding of text, e.g., arithmetic coding (as in Dasher)
- In all these cases:
  - 'good' exemplars of language vs. 'bad' exemplars

Some key concepts in language modeling

- Open vs. closed vocabulary
  - Most natural language applications are closed vocabulary
- Simple multinomial models versus more complex models
  - e.g., continuous space neural networks
- Complexity of features used in models, scalability
  - Finite state models; latent variables; non-local features
- Intrinsic vs. extrinsic evaluation of language model quality
- Generative vs. discriminative models
  - Local vs. global normalization

Language models

- Joint probability is product of conditionals (chain rule)

$$\mathrm{P}(w_1 \ldots w_k) = \mathrm{P}(w_1) \prod_{i=2}^{k} \mathrm{P}(w_i \mid w_1 \ldots w_{i-1})$$

- Call previous words "history", i.e., $h_i = w_1 \ldots w_{i-1}$
  - May choose to keep more or less information in h_i
- Example: *John was very happy*

$$\begin{aligned}\mathrm{P}(\text{John was very happy}) = {} & \mathrm{P}(\text{John} \mid \text{<s>}) * \mathrm{P}(\text{was} \mid \text{John}) * \\ & \mathrm{P}(\text{very} \mid \text{John was}) * \\ & \mathrm{P}(\text{happy} \mid \text{John was very})\end{aligned}$$

Cross Entropy and perplexity (intrinsic quality)

Probability	$\mathrm{P}(w_1 \dots w_N) = \prod_{i=1}^{N} \mathrm{P}(w_i \mid h_i)$
Neg.Log Prob.	$-\log \mathrm{P}(w_1 \dots w_N) = -\sum_{i=1}^{N} \log \mathrm{P}(w_i \mid h_i)$
Cross Entropy	$H_{\mathrm{P}}(w_1 \dots w_N) = -\frac{1}{N} \sum_{i=1}^{N} \log \mathrm{P}(w_i \mid h_i)$
Perplexity	$\mathrm{PPX}_{\mathrm{P}}(w_1 \dots w_N) = \exp(H_{\mathrm{P}}(w_1 \dots w_N))$
	$= (\prod_{i=1}^{N} \mathrm{P}(w_i \mid h_i))^{-\frac{1}{N}}$

- With very large models, correlation between intrinsic and extrinsic improvements is often pretty good

Markov assumption

- Suppose string has 15 words. Must estimate $\mathrm{P}(w_{15} \mid w_1 \ldots w_{14})$
  - too many parameters (hard to estimate; store; access)
- Markov assumption: given the previous k words, the current word is conditionally independent of words further away
- n-gram model: Markov assumption of order $n-1$

Model	Order	$\mathrm{P}(w_1 \ldots w_k) =$
Unigram	0	$\mathrm{P}(w_1) \prod_{i=2}^{k} \mathrm{P}(w_i)$
Bigram	1	$\mathrm{P}(w_1) \prod_{i=2}^{k} \mathrm{P}(w_i \mid w_{i-1})$
Trigram	2	$\mathrm{P}(w_1) \prod_{i=2}^{k} \mathrm{P}(w_i \mid w_{i-2} w_{i-1})$
4-gram	3	$\mathrm{P}(w_1) \prod_{i=2}^{k} \mathrm{P}(w_i \mid w_{i-3} w_{i-2} w_{i-1})$

N-gram models

- De-facto standard language model approach
  - Relatively compact: only store observed n-grams
  - Relatively effective: difficult to improve on performance
  - Scales well: performance improves with more observations
  - Simple "stupid" training methods yield high performing models
- Recent advances in leveraging large amounts of data
  - More data yields robust estimates for larger Markov orders
  - Distributed methods using simplified smoothing
  - Lossy hash-based methods yield space savings

Smoothing (regularization)

- Let h_i be the conditioning history for w_i
- Probabilities usually estimated via maximum likelihood

$$\widehat{\mathrm{P}}(w_i \mid h_i) = \frac{\mathrm{C}(h_i w_i)}{\mathrm{C}(h_i)}$$

  where $C(h)$ is the frequency of h in a large corpus

- Unobserved n-grams get zero probability!
- To avoid zero probs, we smooth, i.e. reserve probability mass for unobserved n-grams
- Many techniques; most share same WFSA structure

Smoothing (cont.)

- Let $\widetilde{\mathrm{P}}(w \mid h)$ be some estimate that reserves probability mass for unobserved events – unlike $\widehat{\mathrm{P}}(w \mid h)$
  - many techniques for this (will cover in next few slides)
- For an n-gram history $h = wh'$, where $h \in \Sigma^k$ for some $k \geq 1$ call h' the backoff history
- Then most n-gram smoothing is encoded as

$$\mathrm{P}(w \mid h) = \begin{cases} \widetilde{\mathrm{P}}(w \mid h) & \text{if } c(hw) > 0 \\ \alpha_h \mathrm{P}(w \mid h') & \text{otherwise} \end{cases}$$

where α_h ensures normalization: $\sum_{w \in V} P(w \mid h) = 1$

Well-known smoothing methods

- Katz backoff
  - Based on Good-Turing estimation (leave one out rationale)
- Jelinek-Mercer (deleted interpolation)
  - Mixture of models, can be estimated with EM
- Absolute discounting
  - Steals counts from n-grams to allocate to unobserved
- Witten-Bell
  - Adds counts to each history based on number of n-grams
- Kneser-Ney
  - Modifies lower-order distributions of absolute discounting models

Good-Turing

- Let $\mathrm{w} = w_1 w_2 \ldots w_N$ be a random sample of size N
- Based on this sample, estimate word probabilities
  (i.e. if I sample one more word)
- Maximum likelihood: $\widehat{\mathrm{P}}(w|\mathrm{w}) = \frac{r}{N}$
  where $r = c(w, \mathrm{w})$ is the count of w in w
- Good-Turing:

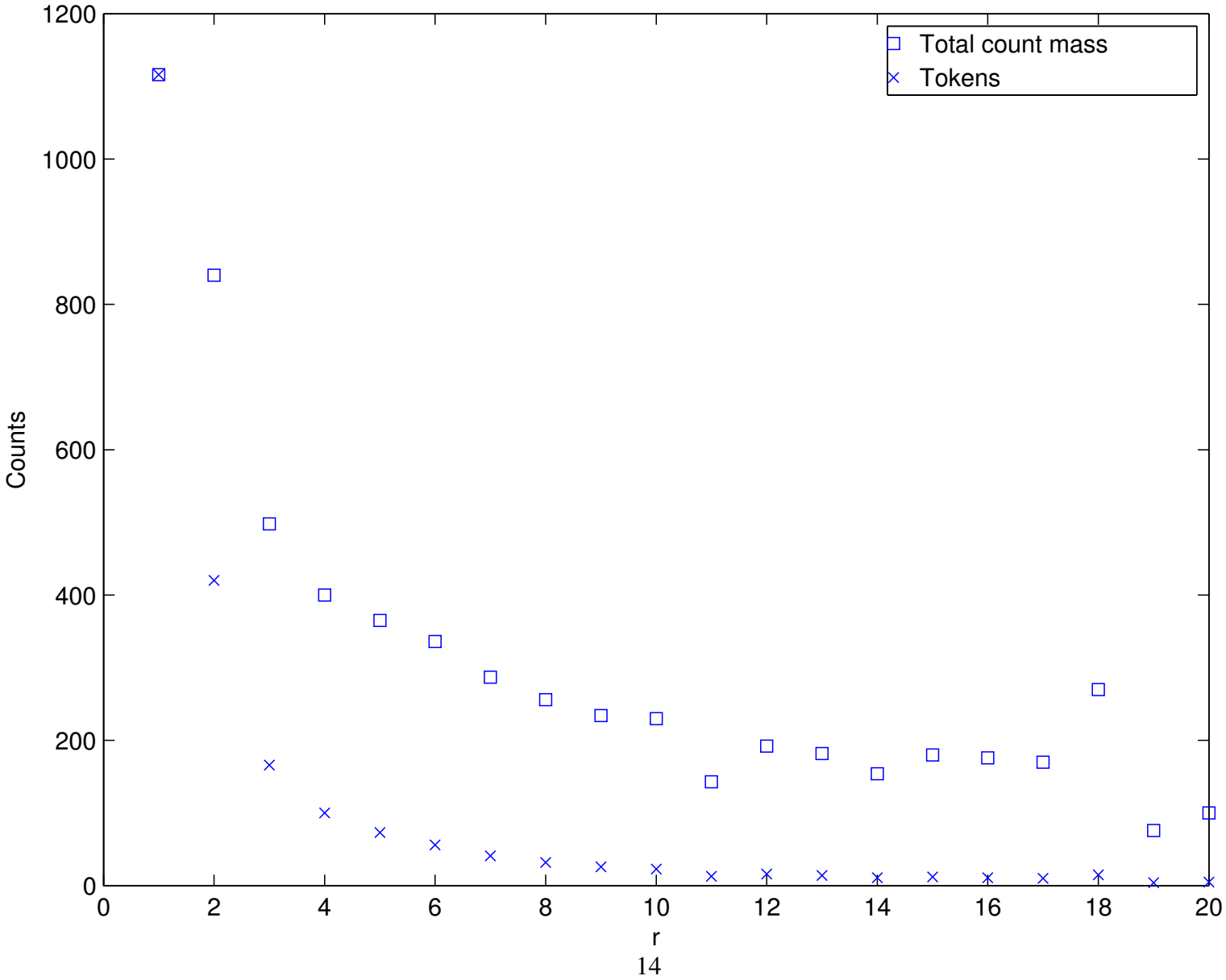
$$\widetilde{\mathrm{P}}_{gt}(w|\mathrm{w}) = \frac{r+1}{N}\frac{n_{r+1}}{n_r} \tag{1}$$

  where n_r is the number of distinct words that have count r in a sample

Good-Turing - intuition

- All items with the same count should have the same probability
- If w_{N+1} has count r in w, it now has count $r+1$
- We should expect something from the set n_r about as frequently as set n_{r+1} was observed.
- $(r+1)n_{r+1}$ is the total count of words that occur $r+1$ times in w, i.e. the total count mass
- $\frac{(r+1)n_{r+1}}{n_r}$ spreads the counts around evenly among the set n_r
- $\frac{r+1}{N}\frac{n_{r+1}}{n_r}$ normalizes the probabilities

Simple histogram plot from small text sample



Katz backoff

- Extends Good-Turing to higher order n-grams
  - and limits the count bins where discounting applies
- For $r > 0$, $\widetilde{\mathrm{P}}_{katz}(w_i|h) = \frac{rd_r}{C(h)}$
- If $r > k$ (typically $k = 5$), $d_r = 1$
- For $r \leq k$, $d_r \approx \frac{r+1}{r}\frac{n_{r+1}}{n_r}$
- Need to adjust for the fact that we stop at k:

$$d_r = \frac{\frac{r+1}{r}\frac{n_{r+1}}{n_r} - \frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}}$$

$$\text{if } k = 5, \text{ then } d_r = \frac{\frac{r+1}{r}\frac{n_{r+1}}{n_r} - \frac{6n_6}{n_1}}{1 - \frac{6n_6}{n_1}}$$

Katz backoff

- Assumption is that $\frac{r+1}{r}\frac{n_{r+1}}{n_r} < 1$ for $r \leq k$

  unigrams in a million words of Wall St. Journal:

r	n_r	rn_r	$\frac{(r+1)n_{r+1}}{rn_r}$	d_r	rd_r
1	20621	20621	0.6233	0.4369	0.4369
2	6427	12854	0.7620	0.6442	1.2884
3	3265	9795	0.8445	0.7675	2.3026
4	2068	8272	0.8928	0.8397	3.3587
5	1477	7385	0.9246	0.8872	4.4362
6	1138	6828		1	6

- Assumption holds for unigrams in corpus until $r = 10$

Jelinek Mercer (Deleted Interpolation)

- $\widetilde{\mathrm{P}}_{jm}(w_i|h) = \lambda_h \widehat{\mathrm{P}}(w_i|h) + (1-\lambda_h)\widetilde{\mathrm{P}}_{jm}(w_i|h')$
- Mixing parameter λ_h is typically estimated using EM on held out data (requires parameter tying across states)
- Note that our standard formulation:

$$\mathrm{P}(w \mid h) = \begin{cases} \widetilde{\mathrm{P}}(w \mid h) & \text{if } c(hw) > 0 \\ \alpha_h \mathrm{P}(w \mid h') & \text{otherwise} \end{cases}$$

holds in this case, with $\alpha_h = 1 - \lambda_h$

(will demonstrate explicitly later)

Absolute discounting

- Set $\widetilde{C}(hw_i) = \max(C(hw_i) - d, 0)$ for some d
- $\widetilde{\mathrm{P}}_{abs}(w_i|h) = \frac{\widetilde{C}(hw_i)}{C(h)} + (1 - \lambda_h)\widetilde{\mathrm{P}}_{abs}(w_i|h')$

  where $\lambda_h = \frac{\sum_{w_i} \widetilde{C}(hw_i)}{\sum_{w_i} C(hw_i)}$
- One proposed rule of thumb: $d = \frac{n_1}{n_1+2n_2}$
- From WSJ unigrams, this would be 0.6160

	1	2	3	4	5
Katz:	0.4369	1.2884	2.3026	3.3587	4.4362
Abs:	0.384	1.384	2.384	3.384	4.384

Witten Bell

- Mixing approach like deleted interpolation
- $\lambda_h = \frac{C(h)}{C(h) + k|\{w{:}C(hw){>}0\}|}$
- i.e., add k times # of n-grams following history to denominator
  - k is a metaparameter that is typically =1 (e.g., for your HW)
- Intuition: smoothes heavily if $C(h)$ is low or many low $C(hw_i)$
- Less smoothing if high counts and few alternatives
  - e.g. in a corpus about cars, if **Rolls** occurs frequently, it is likely to occur with **Royce**.

Kneser-Ney estimation

- This technique modifies only lower order distributions
  - Improve their utility within smoothing
- Intuition: **Royce** may occur frequently in a corpus but it is unlikely to follow anything but **Rolls**
  - Hence unigram probability should be small within bigram model
- A variant of absolute discounting
  - Highest order n-gram probs left the same

$$\widetilde{\mathrm{P}}_{abs}(w_i|h) = \frac{\widetilde{C}(hw_i)}{C(h)} + (1 - \lambda_h)\widetilde{\mathrm{P}}_{kn}(w_i|h')$$

where $\mathrm{P}_{kn}(w_i|h') \propto |\{w_j : C(w_j h' w_i) > 0\}|$

Smoothed n-gram models: unified framework

- All of these approaches are instances of

$$\mathrm{P}(w \mid h) = \begin{cases} \widetilde{\mathrm{P}}(w \mid h) & \text{if } c(hw) > 0 \\ \alpha_h \mathrm{P}(w \mid h') & \text{otherwise} \end{cases}$$

- To make things normalize

$$\alpha_h = \frac{1 - \sum_{w_i : C(hw_i) > 0} \widetilde{\mathrm{P}}(w_i | h)}{1 - \sum_{w_i : C(hw_i) > 0} \mathrm{P}(w_i | h')}$$

- In mixture approaches, $\alpha_h = 1 - \lambda_h$

Mixture α_h

- e.g. Jelinek Mercer:

$$
\begin{aligned}
\alpha_h &= \frac{1 - \sum_{w_i : C(hw_i) > 0} \widetilde{\mathrm{P}}(w_i|h)}{1 - \sum_{w_i : C(hw_i) > 0} \mathrm{P}(w_i|h')} \\
&= \frac{1 - \sum_{w_i : C(hw_i) > 0} (\lambda_h \widehat{\mathrm{P}}(w_i|h) + (1 - \lambda_h) \widetilde{\mathrm{P}}_{jm}(w_i|h'))}{1 - \sum_{w_i : C(hw_i) > 0} \widetilde{\mathrm{P}}_{jm}(w_i|h')} \\
&= \frac{1 - \lambda_h \sum_{w_i : C(hw_i) > 0} \widehat{\mathrm{P}}(w_i|h) - (1 - \lambda_h) \sum_{w_i : C(hw_i) > 0} \widetilde{\mathrm{P}}_{jm}(w_i|h')}{1 - \sum_{w_i : C(hw_i) > 0} \widetilde{\mathrm{P}}_{jm}(w_i|h')} \\
&= \frac{1 - \lambda_h - (1 - \lambda_h) \sum_{w_i : C(hw_i) > 0} \widetilde{\mathrm{P}}_{jm}(w_i|h')}{1 - \sum_{w_i : C(hw_i) > 0} \widetilde{\mathrm{P}}_{jm}(w_i|h')} \\
&= \frac{(1 - \lambda_h)(1 - \sum_{w_i : C(hw_i) > 0} \widetilde{\mathrm{P}}_{jm}(w_i|h'))}{1 - \sum_{w_i : C(hw_i) > 0} \widetilde{\mathrm{P}}_{jm}(w_i|h')} \\
&= 1 - \lambda_h
\end{aligned}
$$

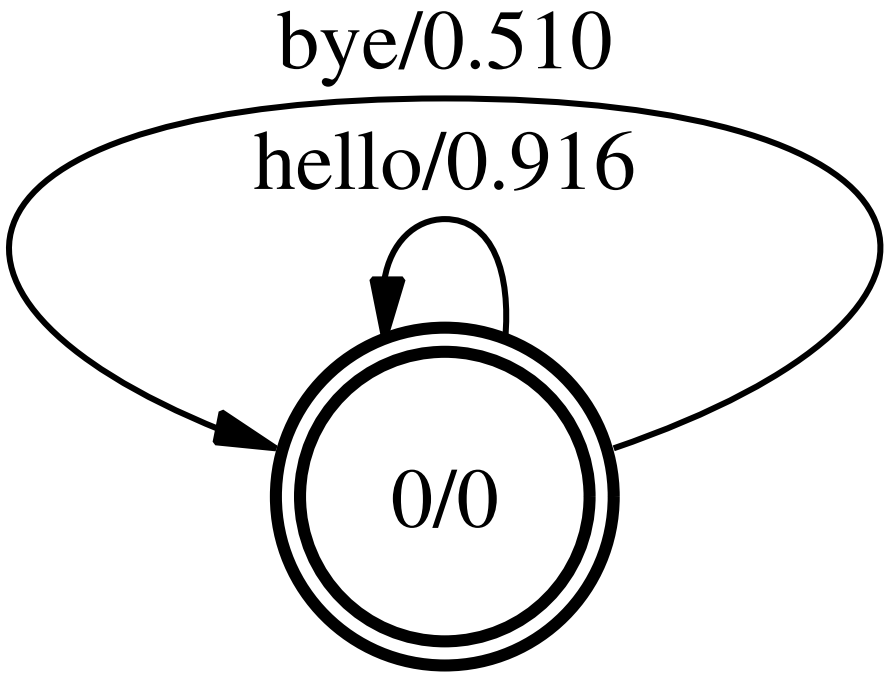
Encoding LMs as WFA

- N-gram models are finite-state and can be represented compactly via weighted finite-state automata (WFA)
- Effective encoding requires some kind of a back-off mechanism
  - Encoded as a "failure" transition
  - Approximated with epsilon transitions
- Encoded as WFA, they can compose with other models off-line
  - Graph optimizations yield more effective search
- Needs to accept all strings from Σ^*
- Important to pay attention to start and end of string

Running example

`corpus.txt`	`wl`	
hello	ϵ	0
bye	hello	1
hello	bye	2
bye bye		

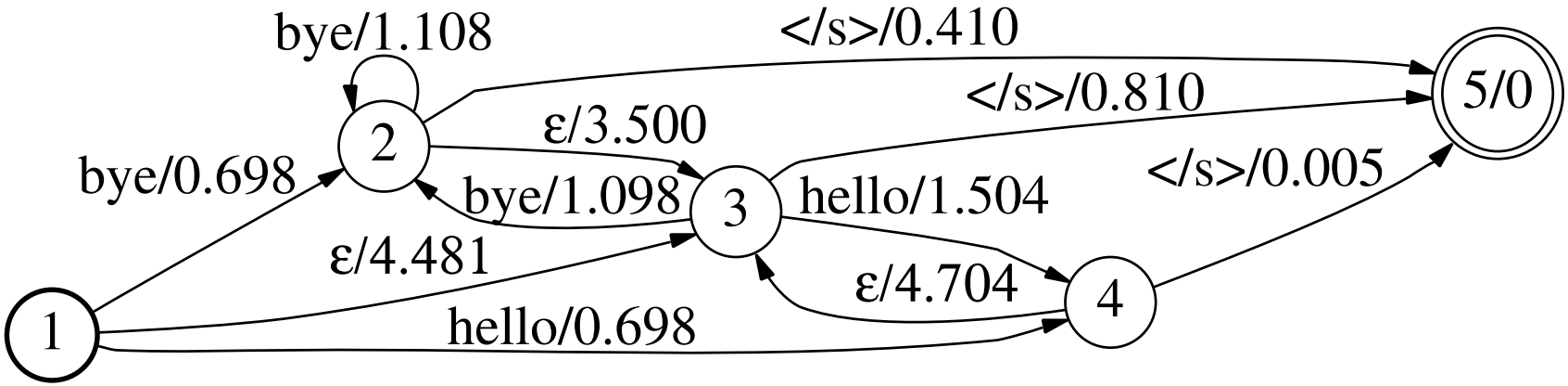
Unigram WFSA



Running example

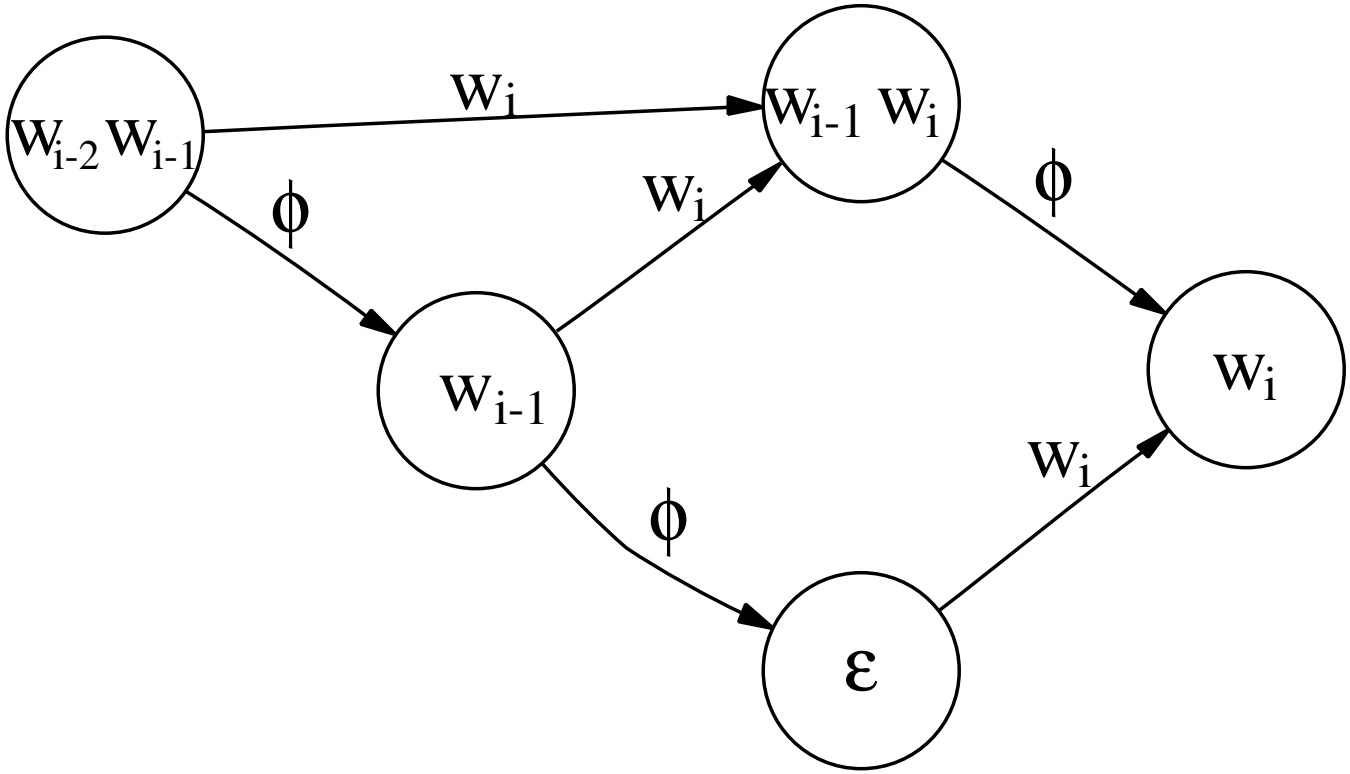
corpus.txt	wl	
<s> hello </s>	ϵ	0
<s> bye </s>	hello	1
<s> hello </s>	bye	2
<s> bye bye </s>	<s>	3
	</s>	4

Bigram WFSA



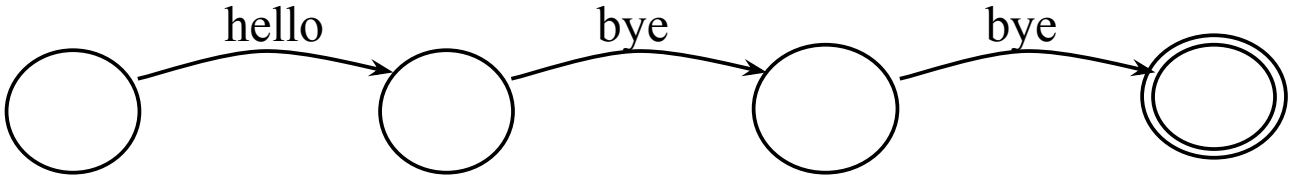
- state 1 encodes history: <s>
- state 2 encodes history: bye
- state 3 encodes history:
- state 4 encodes history: hello
- state 5 encodes history: </s>

Trigram WFSA



Using WFSA language models

- Encode input strings as WFSA
  - For example, encode a single string "hello bye bye" as a linear automaton



  - Or a word lattice encodes many strings on different paths
  - Note explicit </s> in language model examples, versus implicit end-of-string in linear automaton above
- Compose (intersect) with the language model to get score

Some intuitions on Kneser-Ney

- Build a language model automaton, randomly generate text
- Suppose we have a bigram model
  - Want randomly generated unigrams to occur about as frequently as they were observed in the corpus we trained on
- If we use relative frequency for unigram distribution, **Royce** will be generated too frequently in random car corpus
  - Generated with high frequency following **Rolls**
  - Unigram prob. high, hence also generated in other contexts
- Kneser-Ney constrains lower order models so that lower-order n-grams are not overgenerated in random corpus

Further notes on Kneser-Ney

- Modifies lower orders to match observed marginal distributions
  - e.g., expected frequency n-grams match observed frequencies
  - Similar methods are used to train maximum entropy models
  - This generally improves models (Goodman, 2001)
- Pruning of parameters in KN model generally causes problems
  - For various reasons, resulting pruned model is typically bad
  - Different methods for dealing with this (including general method from Roark et al., 2013, which is another talk)

VERY large models

- Training on petabyte scale resources results in many n-grams
- Distributed methods of training and access makes this feasible
- Typically, more data continues to improve models
- Traditional smoothing methods tricky using Map/Reduce
- Very simple, unnormalized variant ("Stupid Backoff") works well
  - Relative frequency for found n-grams; fixed α for rest
- Hash-based methods (Bloom and Bloomier filters) also used
  - Small number of false positives based on hash collisions
  - Mainly used in MT: "lookup table" rather than WFSA

Log linear modeling

- A general stochastic modeling approach that does not assume a particular WFSA structure
  - WFSA structure can be imposed for a particular feature set, e.g., n-grams
- Flexibility of modeling allows for many overlapping features
  - e.g., use unigram, bigram and trigram simultaneously
  - or use POS-tags or morphologically-derived features
- Generative or discriminative training can be used
- Commonly used models for other sequence processing problems

Log linear modeling

- Define a d-dimensional vector of features ϕ

  e.g. $\phi_{1000}(w_{i-2}w_{i-1}w_i)$ = 1 if w_{i-1} is **to** and w_i is **the**, 0 otherwise

- Estimate a d-dimensional parameter vector α

- Then

$$\mathrm{P}(w_i|w_{i-1}w_{i-2}) = \frac{e^{(\sum_{s=1}^{d} \alpha_s \phi_s(w_{i-2}w_{i-1}w_i)}}{Z(w_{i-1}w_{i-2})}$$

  Where

$$Z(w_{i-1}w_{i-2}) = \sum_{w'} e^{(\sum_{s=1}^{d} \alpha_s \phi_s(w_{i-2}w_{i-1}w'))}$$

- We can just consider log P:

$$\log \mathrm{P}(w_i|w_{i-1}w_{i-2}) = \sum_{s=1}^{d} \alpha_s \phi_s(w_{i-2}w_{i-1}w_i) - \log Z(w_{i-2}w_{i-2})$$

Global discriminative models

- Instead of normalizing locally (over next word) can normalize globally (over whole strings)
- Whole sentence models can be trained as follows
  - Generate recognizer output from training data
  - Move parameters to improve score of reference transcript
- Under such a scenario, global normalization tractable
  - Some methods don't bother to normalize, e.g., perceptron
- Often used in a reranking/rescoring scenario
  - Though with certain feature sets, can incorporate in first-pass

Other global models

- Syntactic language models
  - Strings scored via syntactic parsing or tagging
  - Scores can be derived from stochastic grammar itself; or by using features derived from the parse structures
- Recurrant neural networks
  - Outputs from earlier time fed as input at current time
  - Influence from arbitrarily far back in string (not finite state)
- Very large perplexity gains achievable from such methods
  - Stuck doing n-best reranking, hard to move error rates

Language modeling toolkits

- SRI language modeling toolkit (SRILM):
  http://www.speech.sri.com/projects/srilm/
- KenLM
  kheafield.com/code/kenlm/
- OpenGrm n-gram library:
  http://www.opengrm.org