Automatic Speech Recognition (CS753)

Lecture 15: Language Models (Part II)

Instructor: Preethi Jyothi

Mar 2, 2017

Recap

- Ngram language models are popularly used in various ML applications

- Language models are evaluated using the *perplexity* (normalized per-word cross-entropy) measure.

  - For a uniform unigram model over L words, perplexity = L.

- MLE estimates for Ngram models assume there are no unseen Ngrams

- Smoothing algorithms: **Discount** some probability mass from seen Ngrams and redistribute discounted mass to unseen events
  - Two different kinds of smoothing that combine higher-order and lower-order Ngram models: Backoff and Interpolation

Advanced Smoothing Techniques

- Good-Turing Discounting
- Katz Backoff Smoothing
- Absolute Discounting Interpolation
- Kneser-Ney Smoothing

Advanced Smoothing Techniques

- Good-Turing Discounting
- Katz Backoff Smoothing
- Absolute Discounting Interpolation
- Kneser-Ney Smoothing

Recall add-1/add-α smoothing (also viewed as discounting)

- Smoothing can be viewed as **discounting** (lowering) some probability mass from seen Ngrams and redistributing discounted mass to unseen events

- i.e. probability of a bigram with Laplace (add-1) smoothing

$$\Pr_{Lap}(w_i|w_{i-1}) = \frac{\pi(w_{i-1}, w_i) + 1}{\pi(w_{i-1}) + V}$$

- can be written as

$$\Pr_{Lap}(w_i|w_{i-1}) = \frac{\pi^*(w_{i-1}, w_i)}{\pi(w_{i-1})}$$

- where discounted count $\pi^*(w_{i-1}, w_i) = (\pi(w_{i-1}, w_i) + 1)\dfrac{\pi(w_{i-1})}{\pi(w_{i-1}) + V}$

Problems with Add-α Smoothing

- What's wrong with add-α smoothing?
- Assigns too much probability mass away from seen Ngrams to unseen events
- Does not discount high counts and low counts correctly
- Also, α is tricky to set
- Is there a more principled way to do this smoothing?
  A solution: Good-Turing estimation

Good-Turing estimation
(uses held-out data)

r	N_r	True r*	add-1 r*
1	2×10^6	0.448	2.8×10^{-11}
2	4×10^5	1.25	4.2×10^{-11}
3	2×10^5	2.24	5.7×10^{-11}
4	1×10^5	3.23	7.1×10^{-11}
5	7×10^4	4.21	8.5×10^{-11}

r = Count in a large corpus & N_r is the number of bigrams with r counts
True r^* is estimated on a *different* held-out corpus

- Add-1 smoothing hugely overestimates fraction of unseen events
- Good-Turing estimation uses held-out data to predict how to go from r to the true r^*

[CG91]: Church and Gale, “A comparison of enhanced Good-Turing…”, CSL, 1991

Good-Turing Estimation

- Intuition for Good-Turing estimation using leave-one-out validation:
  - Let N_r be the number of word types that occur *r* times in the entire corpus
  - Split a given set of N word tokens into a training set of (N-1) samples + 1 sample as the held-out set; repeat this process N times so that all N samples appear in the held-out set
  - In what fraction of these N trials is the held-out word unseen during training?

    N_1/N
  - In what fraction of these N trials is the held-out word seen exactly k times during training? $(k+1)N_{k+1}/N$
  - There are (≅)N_k words with training count k. Each should occur with probability:

    $(k+1)N_{k+1}/(N \times N_k)$
  - Expected count of each of the N_k words: $\mathbf{k^* = \theta(k) = (k+1)\, N_{k+1}/N_k}$

Good-Turing Smoothing

- Thus, Good-Turing smoothing states that for any Ngram that occurs r times, we should use an adjusted count $\theta(r) = (r + 1)N_{r+1}/N_r$
- Good-Turing smoothed counts for unseen events: $\theta(0) = N_1/N_0$
- Example: 10 bananas, 6 apples, 2 papayas, 1 guava, 1 pear
  - How likely are we to see a guava next? The GT estimate is $\theta(1)/N$
  - Here, $N = 20$, $N_2 = 1$, $N_1 = 2$. Computing $\theta(1)$: $\theta(1) = 2 \times 1/2 = 1$
  - Thus, $Pr_{GT}(guava) = \theta(1)/20 = 0.05$

Good-Turing estimates

r	N_r	$\theta(r)$	True r^*
0	7.47×10^{10}	.0000270	.0000270
1	2×10^6	0.446	0.448
2	4×10^5	1.26	1.25
3	2×10^5	2.24	2.24
4	1×10^5	3.24	3.23
5	7×10^4	4.22	4.21
6	5×10^4	5.19	5.23
7	3.5×10^4	6.21	6.21
8	2.7×10^4	7.24	7.21
9	2.2×10^4	8.25	8.26

Table showing frequencies of bigrams from 0 to 9

In this example, for $r > 0$, $\theta(r) \cong$ True r^* and $\theta(r)$ is always less than r

[CG91]: Church and Gale, "A comparison of enhanced Good-Turing…", CSL, 1991

Good-Turing Estimation

- One issue: For large r, many instances of $N_{r+1} = 0$!
    - This would lead to $\theta(r) = (r + 1)N_{r+1}/N_r$ being set to 0.
- Solution: Discount only for small counts $r <= k$ (e.g. $k = 9$) and $\theta(r) = r$ for $r > k$
- Another solution: Smooth N_r using a best-fit power law once counts start getting small
- Good-Turing smoothing tells us how to discount some probability mass to unseen events. Could we redistribute this mass across observed counts of lower-order Ngram events? Backoff!

Advanced Smoothing Techniques

- Good-Turing Discounting
- Katz Backoff Smoothing
- Absolute Discounting Interpolation
- Kneser-Ney Smoothing

Katz Smoothing

- Good-Turing discounting determines the volume of probability mass that is allocated to unseen events
- Katz Smoothing distributes this remaining mass proportionally across “smaller” Ngrams
  - i.e. no trigram found, use backoff probability of bigram and if no bigram found, use backoff probability of unigram

Katz Backoff Smoothing

- For a Katz bigram model, let us define:
  - $\Psi(w_{i-1}) = \{w: \pi(w_{i-1}, w) > 0\}$
- A bigram model with Katz smoothing can be written in terms of a unigram model as follows:

$$P_{\text{Katz}}(w_i|w_{i-1}) = \begin{cases} \frac{\pi^*(w_{i-1}, w_i)}{\pi(w_{i-1})} & \text{if } w_i \in \Psi(w_{i-1}) \\ \alpha(w_{i-1}) P_{\text{Katz}}(w_i) & \text{if } w_i \notin \Psi(w_{i-1}) \end{cases}$$

$$\text{where} \quad \alpha(w_{i-1}) = \frac{\left(1 - \sum_{w \in \Psi(w_{i-1})} \frac{\pi^*(w_{i-1}, w)}{\pi(w_{i-1})}\right)}{\sum_{w_i \notin \Psi(w_{i-1})} P_{\text{Katz}}(w_i)}$$

Katz Backoff Smoothing

$$P_{\text{Katz}}(w_i|w_{i-1}) = \begin{cases} \frac{\pi^*(w_{i-1},w_i)}{\pi(w_{i-1})} & \text{if } w_i \in \Psi(w_{i-1}) \\ \alpha(w_{i-1}) \boxed{P_{\text{Katz}}(w_i)} & \text{if } w_i \notin \Psi(w_{i-1}) \end{cases}$$

$$\text{where} \quad \alpha(w_{i-1}) = \frac{\boxed{\left(1 - \sum_{w \in \Psi(w_{i-1})} \frac{\pi^*(w_{i-1},w)}{\pi(w_{i-1})}\right)}}{\boxed{\sum_{w_i \notin \Psi(w_{i-1})} P_{\text{Katz}}(w_i)}}$$

- A bigram with a non-zero count is discounted using Good-Turing estimation
- The left-over probability mass from discounting for the unigram model …
- … is distributed over $w_i \notin \Psi(w_{i-1})$ proportionally to $P_{\text{Katz}}(w_i)$

Advanced Smoothing Techniques

- Good-Turing Discounting
- Katz Backoff Smoothing
- Absolute Discounting Interpolation
- Kneser-Ney Smoothing

Recall Good-Turing estimates

r	N_r	$\theta(r)$
0	7.47×10^{10}	.0000270
1	2×10^6	0.446
2	4×10^5	1.26
3	2×10^5	2.24
4	1×10^5	3.24
5	7×10^4	4.22
6	5×10^4	5.19
7	3.5×10^4	6.21
8	2.7×10^4	7.24
9	2.2×10^4	8.25

For $r > 0$, we observe that $\theta(r) \cong r - 0.75$ i.e. an absolute discounting

[CG91]: Church and Gale, "A comparison of enhanced Good-Turing…", CSL, 1991

Absolute Discounting Interpolation

- Absolute discounting motivated by Good-Turing estimation
- Just subtract a constant d from the non-zero counts to get the discounted count
- Also involves linear interpolation with lower-order models

$$\Pr_{\text{abs}}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda(w_{i-1})\Pr(w_i)$$

Advanced Smoothing Techniques

- Good-Turing Discounting
- Katz Backoff Smoothing
- Absolute Discounting Interpolation
- Kneser-Ney Smoothing

Kneser-Ney discounting

$$\Pr_{\mathrm{KN}}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda_{\mathrm{KN}}(w_{i-1})\Pr_{\mathrm{cont}}(w_i)$$

c.f., absolute discounting

$$\Pr_{\mathrm{abs}}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda(w_{i-1})\Pr(w_i)$$

Kneser-Ney discounting

$$\Pr_{\mathrm{KN}}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda_{\mathrm{KN}}(w_{i-1})\Pr_{\mathrm{cont}}(w_i)$$

Consider an example: "*Today I cooked some yellow <u>curry</u>*"

Suppose π(*yellow, curry*) = 0. $\Pr_{\mathrm{abs}}$[*w* | *yellow*] = λ(*yellow*)Pr(*w*)

Now, say Pr[*Francisco*] >> Pr[*curry*], as *San Francisco* is very common in our corpus.

But *Francisco* is not as common a "continuation" (follows only *San*) as *curry* is (*red curry, chicken curry*, *potato curry*, ...)

Moral: Should use probability of being a continuation!

c.f., absolute discounting

$$\Pr_{\mathrm{abs}}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda(w_{i-1})\Pr(w_i)$$

Kneser-Ney discounting

$$\Pr_{\text{KN}}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda_{\text{KN}}(w_{i-1})\Pr_{\text{cont}}(w_i)$$

$$\Pr_{\text{cont}}(w_i) = \frac{|\Phi(w_i)|}{|B|} \quad \text{and} \quad \lambda_{\text{KN}}(w_{i-1}) = \frac{d}{\pi(w_{i-1})}|\Psi(w_{i-1})|$$

$$\frac{d \cdot |\Psi(w_{i-1})| \cdot |\Phi(w_i)|}{\pi(w_{i-1}) \cdot |B|}$$

where

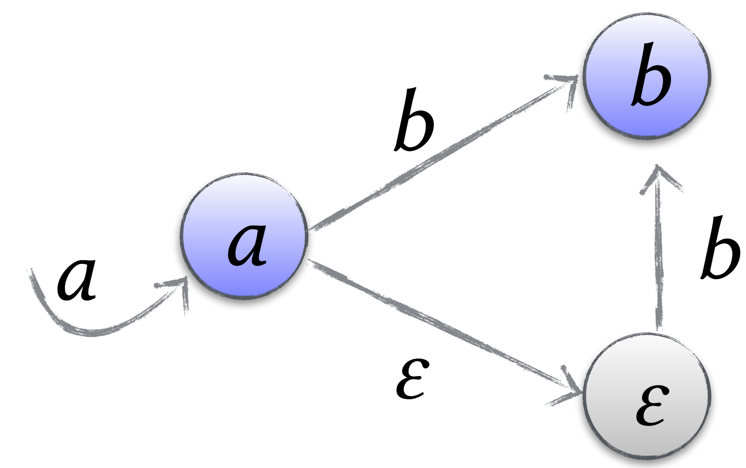
$$\Phi(w_i) = \{w_{i-1} : \pi(w_{i-1}, w_i) > 0\}$$
$$B = \{(w_{i-1}, w_i) : \pi(w_{i-1}, w_i) > 0\}$$

c.f., absolute discounting

$$\Pr_{\text{abs}}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda(w_{i-1})\Pr(w_i)$$

Kneser-Ney: An Alternate View

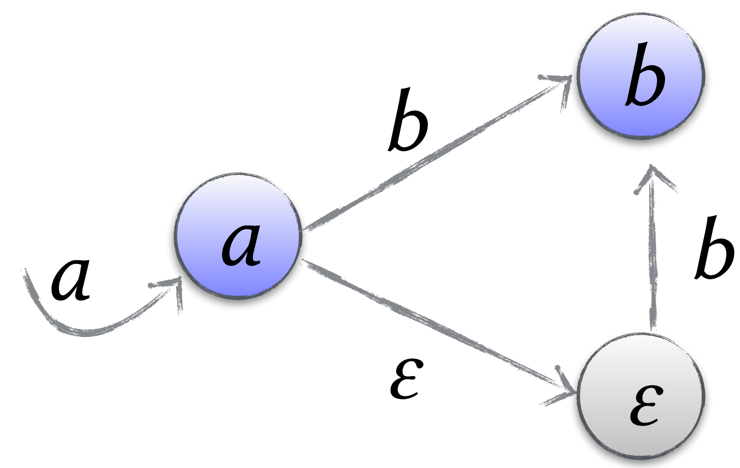
- A mix of bigram and unigram models



- A bigram *ab* could be generated in two ways:
  - In context *a*, output *b*, or
  - In context *a*, forget context and then output *b* (i.e., as "*aεb*")
- In a given set of bigrams, for each bigram *ab*, assume that d_{ab} of its occurrences were produced in the second way
- Will compute probabilities for each transition under this assumption

Kneser-Ney: An Alternate View

- Assuming $\pi(a,b)$ - d_{ab} occurrences as "*ab*", and d_{ab} occurrences as "*aεb*"
  - $\Pr[b|a] = [\pi(a,b) - d_{ab}] / \pi(a)$
  - $\Pr[\varepsilon|a] = [\Sigma_y d_{ay}] / \pi(a)$
  - $\Pr[b|\varepsilon] = [\Sigma_x d_{xb}] / [\Sigma_{xy} d_{xy}]$
  - $\Pr_{\mathrm{KN}}[b|a] = \Pr[b|a] + \Pr[\varepsilon|a] \cdot \Pr[b|\varepsilon]$



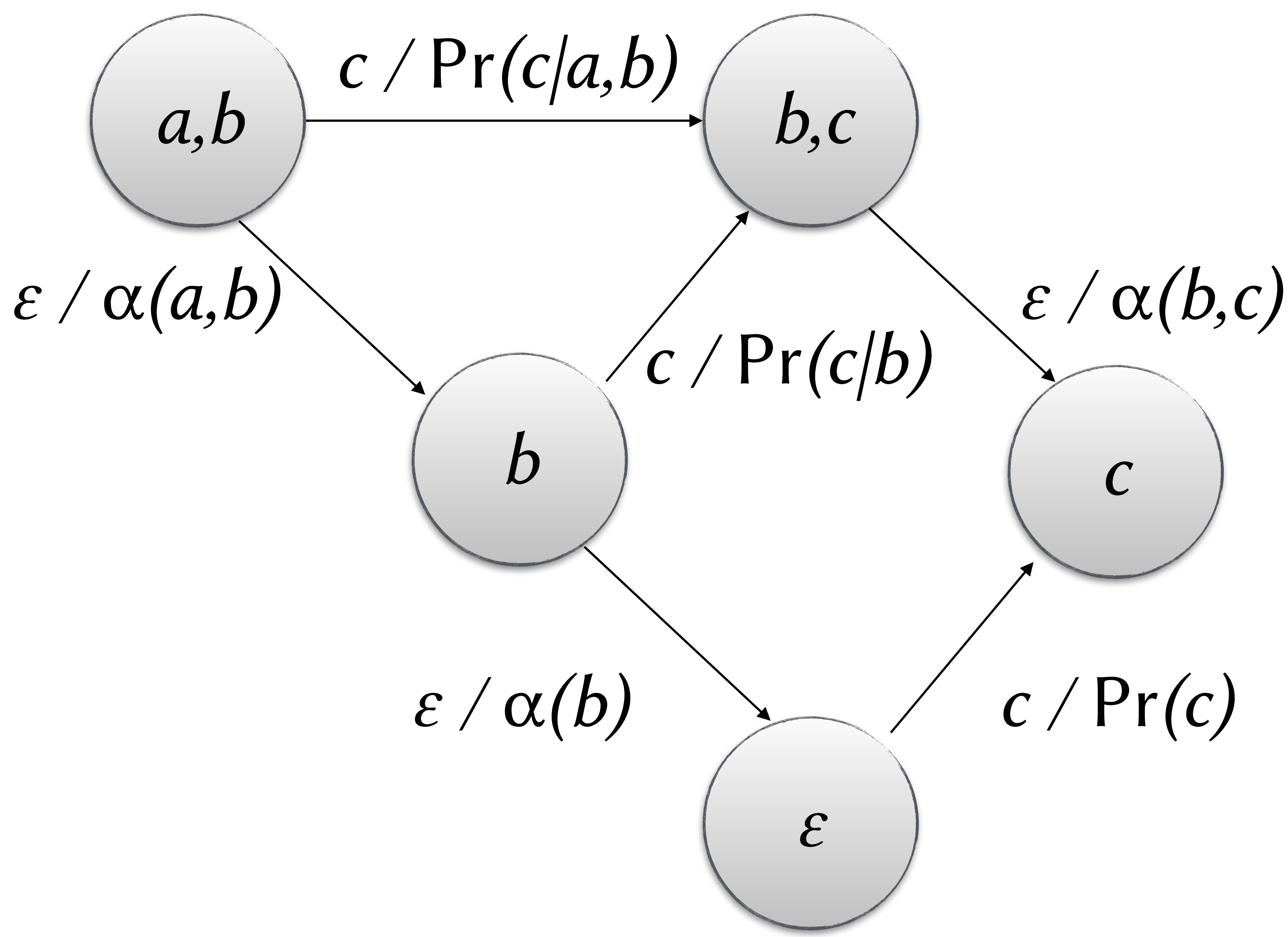
- Kneser-Ney: Take $d_{xy} = d$ for all bigrams *xy* that do appear (assuming they all appear at least d times — kosher, e.g., if $d = 1$)
  - Then $\Sigma_y d_{ay} = d \cdot |\Psi(a)|$, $\Sigma_x d_{xb} = d \cdot |\Phi(b)|$, and $\Sigma_{xy} d_{xy} = d \cdot |B|$
    where $\Psi(a) = \{y : \pi(a,y) > 0\}$, $\Phi(b) = \{x : \pi(x,b) > 0\}$, $B = \{xy : \pi(x,y) > 0\}$

$$\Pr_{\mathrm{KN}}(b|a) = \frac{\max\{\pi(a,b) - d, 0\}}{\pi(a)} + \frac{d \cdot |\Psi(a)| \cdot |\Phi(b)|}{\pi(a) \cdot |B|}$$

Ngram models as WFSAs

- With no optimizations, an Ngram over a vocabulary of V words defines a WFSA with V^{N-1} states and V^N edges.
- Example: Consider a trigram model for a two-word vocabulary, A B.
  - 4 states representing bigram histories, A_A, A_B, B_A, B_B
  - 8 arcs transitioning between these states
- Clearly not practical when V is large.
  - Resort to backoff language models

WFSA for backoff language model



Next class: Beyond Ngram LMs