Language Modeling (Part II)

Lecture 10

CS 753

Instructor: Preethi Jyothi

Unseen Ngrams

- By using estimates based on counts from large text corpora, there will still be many unseen bigrams/trigrams at test time that never appear in the training corpus
- If any unseen Ngram appears in a test sentence, the sentence will be assigned probability 0
- Problem with MLE estimates: Maximises the likelihood of the observed data by assuming anything unseen cannot happen and overfits to the training data
    - Smoothing methods: Reserve some probability mass to Ngrams that don’t occur in the training corpus

Add-one (Laplace) smoothing

Simple idea: Add one to all bigram counts. That means,

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

becomes

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

where V is the vocabulary size

Example: Bigram counts

No smoothing

	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

Laplace (Add-one) smoothing

	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

Example: Bigram probabilities

No smoothing

	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

Laplace (Add-one) smoothing

	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

Laplace smoothing moves too much probability mass to unseen events!

Add-α Smoothing

Instead of 1, add $\alpha < 1$ to each count

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

Choosing α:

- Train model on training set using different values of α
- Choose the value of α that minimizes cross entropy on the development set

Smoothing or 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 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 $\pi^*(w_{i-1}, w_i) = (\pi(w_{i-1}, w_i) + 1)\dfrac{\pi(w_{i-1})}{\pi(w_{i-1}) + V}$

Example: Bigram adjusted counts

No smoothing

	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

Laplace (Add-one) smoothing

	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

Advanced Smoothing Techniques

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

Advanced Smoothing Techniques

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

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	r* in heldout set	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
r* is estimated on a different held-out corpus

- Add-1 smoothing hugely overestimates fraction of unseen events
- Good-Turing estimation uses observed data to predict how to go from r to the heldout-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 words (tokens,bigrams,etc.) that occur r times
  - 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 $(\cong)N_k$ words with training count k.
    - Probability of each being chosen as held-out: $(k+1)N_{k+1}/(N \times N_k)$
  - Expected count of each of the N_k words in a corpus of size N: $k^* = \theta(k) = (k+1)\, N_{k+1}/N_k$

Good-Turing Estimates

r	N_r	r*-GT	r*-heldout
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 $r > 0$, r*-GT $\cong$ r*-heldout and r*-GT is always less than r

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

Good-Turing Smoothing

- Thus, Good-Turing smoothing states that for any Ngram that occurs r times, we should use an adjusted count $r^* = \theta(r) = (r + 1)N_{r+1}/N_r$
- Good-Turing smoothed counts for unseen events: $\theta(0) = N_1/N_0$
- Example: 10 bananas, 5 apples, 2 papayas, 1 melon, 1 guava, 1 pear
  - How likely are we to see a guava next? The GT estimate is $\theta(1)/N$
  - Here, $N = 20$, $N_2 = 1$, $N_1 = 3$. Computing $\theta(1)$: $\theta(1) = 2 \times 1/3 = 2/3$
  - Thus, $Pr_{GT}(\text{guava}) = \theta(1)/20 = 1/30 = 0.0333$

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?

Advanced Smoothing Techniques

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

Backoff and Interpolation

- General idea: It helps to use lesser context to generalise for contexts that the model doesn't know enough about
- Backoff:
  - Use trigram probabilities if there is sufficient evidence
  - Else use bigram or unigram probabilities
- Interpolation
  - Mix probability estimates combining trigram, bigram and unigram counts

Interpolation

- Linear interpolation: Linear combination of different Ngram models

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

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$

How to set the λ's?

Interpolation

- Linear interpolation: Linear combination of different Ngram models

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

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$

1. Estimate N-gram probabilities on a training set.

2. Then, search for λ's that maximises the probability of a held-out set

Advanced Smoothing Techniques

- Good-Turing Discounting
- Backoff and Interpolation
  - 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:
  - $\Psi(w_{i\text{-}1}) = \{w: \pi(w_{i\text{-}1}, w) > 0\}$
- A bigram model with Katz smoothing can be written in terms of a unigram model as follows:

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

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

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
- Backoff and Interpolation
  - 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

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

Advanced Smoothing Techniques

- Good-Turing Discounting
- Backoff and Interpolation
  - 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 curry”

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

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

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

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

where

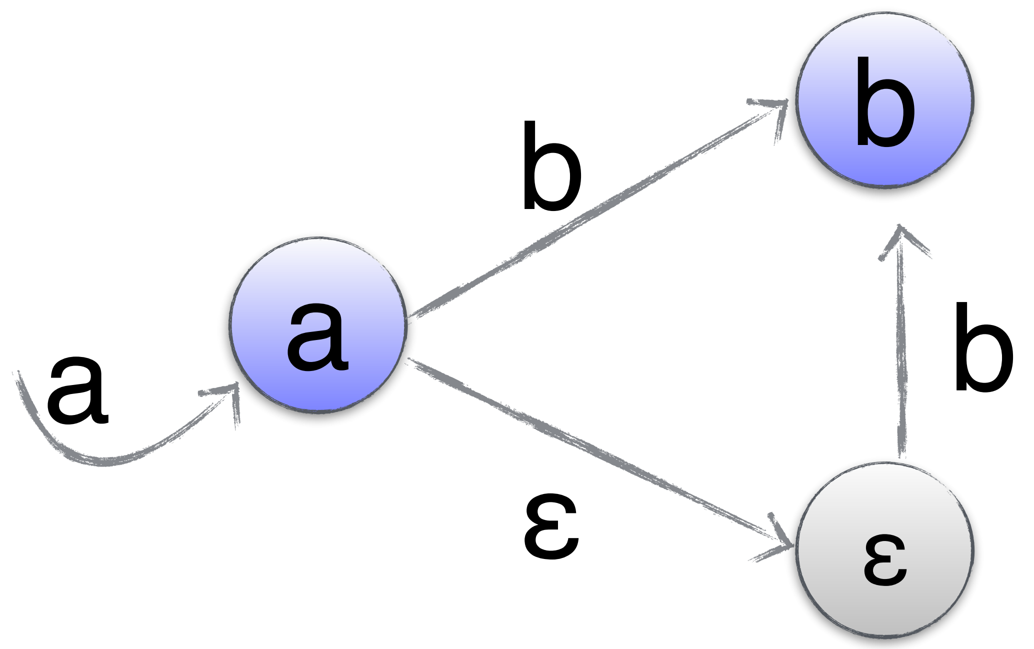
$$\Phi(w_i) = \{w_{i-1} : \pi(w_{i-1}, w_i) > 0\}$$
$$B = \{(w_{i-1}, w_i) : \pi(w_{i-1}, w_i) > 0\}$$

c.f., absolute discounting

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

Kneser-Ney: An Alternate View

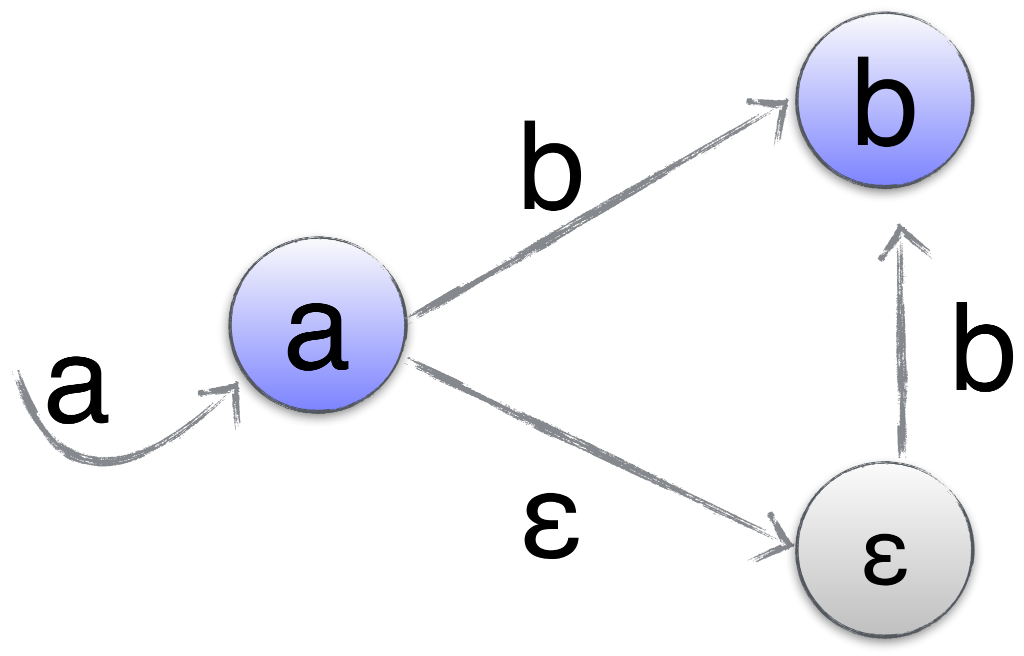
- A mix of bigram and unigram models
- A bigram ab could be generated in two ways:
  - In context a, output b, or
  - In context a, forget context and then output b (i.e., as “aεb”)



- 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"
    - $\Pr[b|a] = [\pi(a,b) - d_{ab}] / \pi(a)$
    - $\Pr[\varepsilon|a] = [\Sigma_y d_{ay}] / \pi(a)$
    - $\Pr[b|\varepsilon] = [\Sigma_x d_{xb}] / [\Sigma_{xy} d_{xy}]$
    - $\Pr_{KN}[b \mid a] = \Pr[b|a] + \Pr[\varepsilon|a] \cdot \Pr[b|\varepsilon]$



- 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 $\Psi(a) = \{y : \pi(a,y) > 0\}$, $\Phi(b) = \{x : \pi(x,b) > 0\}$, $B = \{xy : \pi(x,y) > 0\}$

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

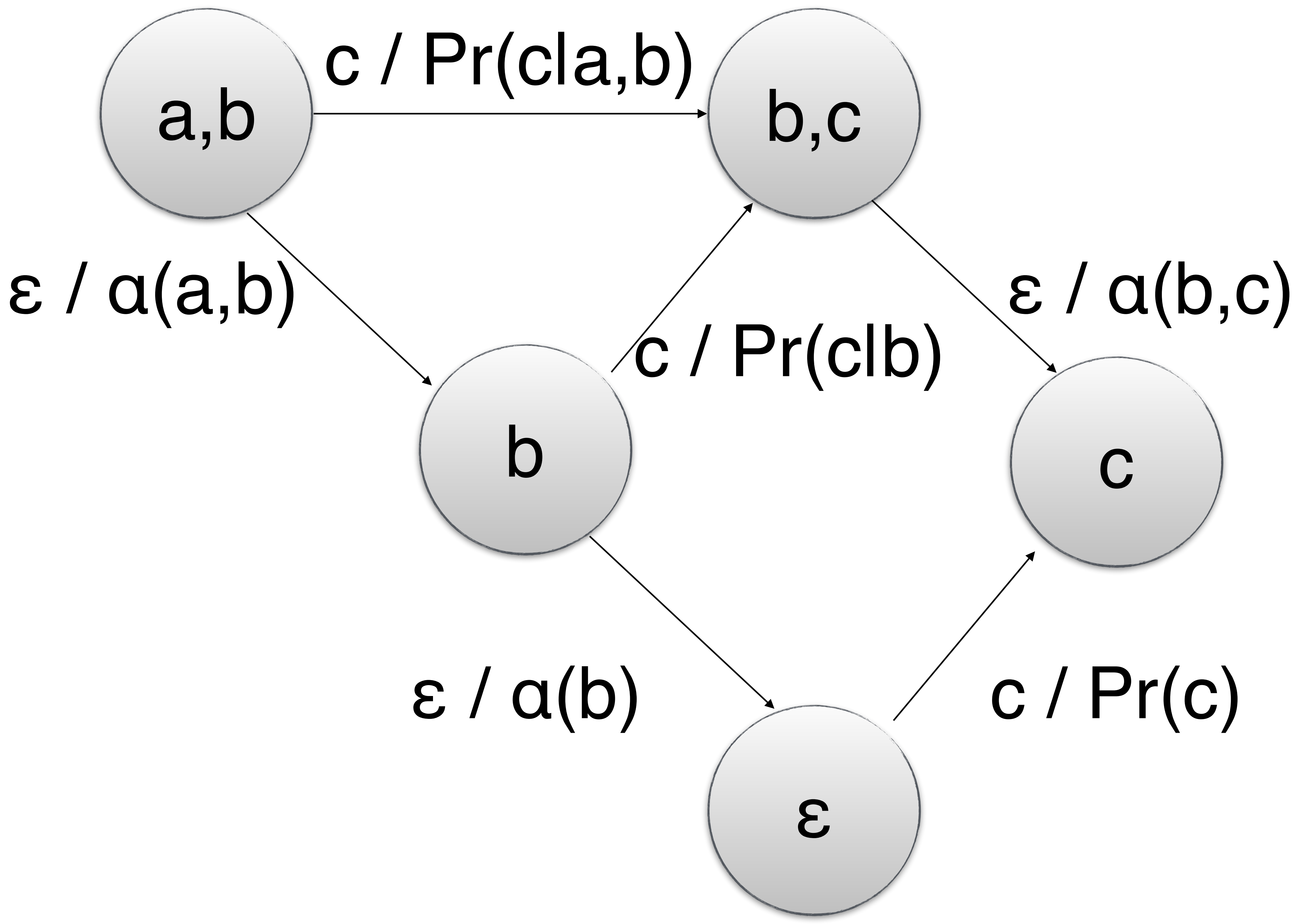
Ngram models as WFSAs

- 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



Putting it all together:
How do we recognise an utterance?

- A: speech utterance
- O_A: acoustic features corresponding to the utterance A

$$W^* = \arg\max_W \Pr(O_A|W)\Pr(W)$$

- Return the word sequence that jointly assigns the highest probability to O_A
- How do we estimate $\Pr(O_A|W)$ and $\Pr(W)$?
- How do we decode?

Acoustic model

$$W^* = \arg\max_W \boxed{\Pr(O_A|W)}\, \Pr(W)$$

$$\Pr(O_A|W) = \sum_Q \Pr(O_A, Q|W)$$

$$= \sum_{q_1^T, w_1^N} \prod_{t=1}^{T} \Pr(O_t|O_1^{t-1}, q_1^t, w_1^N)\, \Pr(q_t|q_1^{t-1}, w_1^N)$$

First-order HMM assumptions

$$\approx \sum_{q_1^T, w_1^N} \prod_{t=1}^{T} \Pr(O_t|q_t, w_1^N)\, \Pr(q_t|q_{t-1}, w_1^N)$$

Viterbi approximation

$$\approx \max_{q_1^T, w_1^N} \prod_{t=1}^{T} \Pr(O_t|q_t, w_1^N)\, \Pr(q_t|q_{t-1}, w_1^N)$$

Acoustic Model

Transition probabilities

$$\Pr(O_A|W) = \max_{q_1^T, w_1^N} \prod_{t=1}^{T} \Pr(O_t|q_t, w_1^N) \Pr(q_t|q_{t-1}, w_1^N)$$

Emission probabilities

Derived from a DNN or TDNN model

Modeled using a mixture of Gaussians

$$\Pr(O|q; w_1^N) = \sum_{\ell=1}^{L_q} c_{q\ell} \mathcal{N}(O|\mu_{q\ell}, \Sigma_{q\ell}; w_1^N)$$

$$\Pr(O\,|\,q; w_1^N) \propto \frac{\Pr(q\,|\,O; w_1^N)}{\Pr(q)}$$

Language Model

$$W^* = \arg\max_W \Pr(O_A|W)\,\Pr(W)$$

$$\begin{aligned}\Pr(W) &= \Pr(w_1, w_2, \ldots, w_N) \\ &= \Pr(w_1)\ldots\Pr(w_N|w_{N-m+1}^{N-1})\end{aligned}$$

m-gram language model

- Further optimized using smoothing and interpolation with lower-order Ngram models

Decoding

$$W^* = \arg\max_{W} \Pr(O_A|W)\Pr(W)$$

$$W^* = \arg\max_{w_1^N, N} \left\{ \left[\prod_{n=1}^{N} \Pr(w_n|w_{n-m+1}^{n-1}) \right] \left[\sum_{q_1^T, w_1^N} \prod_{t=1}^{T} \Pr(O_t|q_t, w_1^N)\Pr(q_t|q_{t-1}, w_1^N) \right] \right\}$$

$$\overset{\text{Viterbi}}{\approx} \arg\max_{w_1^N, N} \left\{ \left[\prod_{n=1}^{N} \Pr(w_n|w_{n-m+1}^{n-1}) \right] \left[\max_{q_1^T, w_1^N} \prod_{t=1}^{T} \Pr(O_t|q_t, w_1^N)\Pr(q_t|q_{t-1}, w_1^N) \right] \right\}$$

- Viterbi approximation divides the above optimisation problem into sub-problems that allows the efficient application of dynamic programming
- Search space still very huge for LVCSR tasks! Use approximate decoding techniques (A* decoding, beam-width decoding, etc.) to visit only promising parts of the search space