Natural Language Processing

Lecture 2: Language Modeling

The Language Modeling Problem

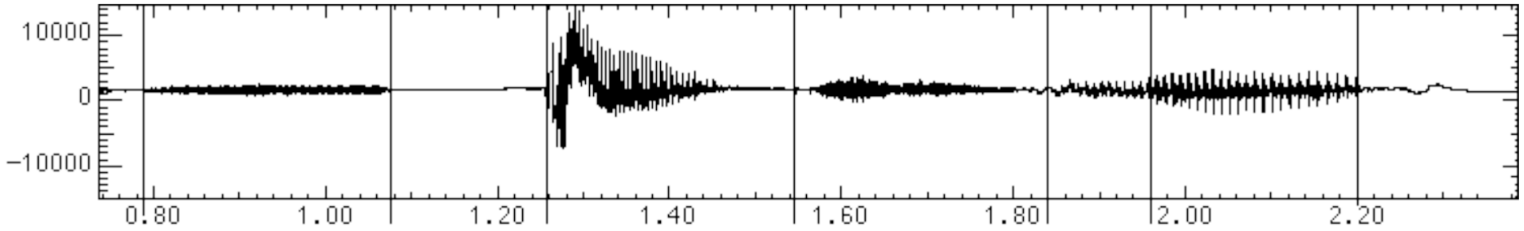
- **Goal:** given a sequence of words, what’s the distribution of the next word
  - Also known as Shannon’s game
- More formally:
  - Setup: assume a finite set of words (vocabulary) *V*
  - Data: a training set of examples
  - Problem: estimate the probability distribution over all sequences over V

$$\sum_{x \in \mathcal{V}^{\dagger}} p(x) = 1$$

Application: Speech Recognition

- Audio in, text out
- SOTA: 0.3% error for digit strings, 5% dictation, 50%+ TV

- "Wreck a nice beach?"
  - "Recognize speech"
- "Eye eight uh Jerry?"
  - "I ate a cherry"



The Noisy Channel Model

- Goal: predict sentence given acoustics

$$w^* = \arg\max_w P(w|a)$$

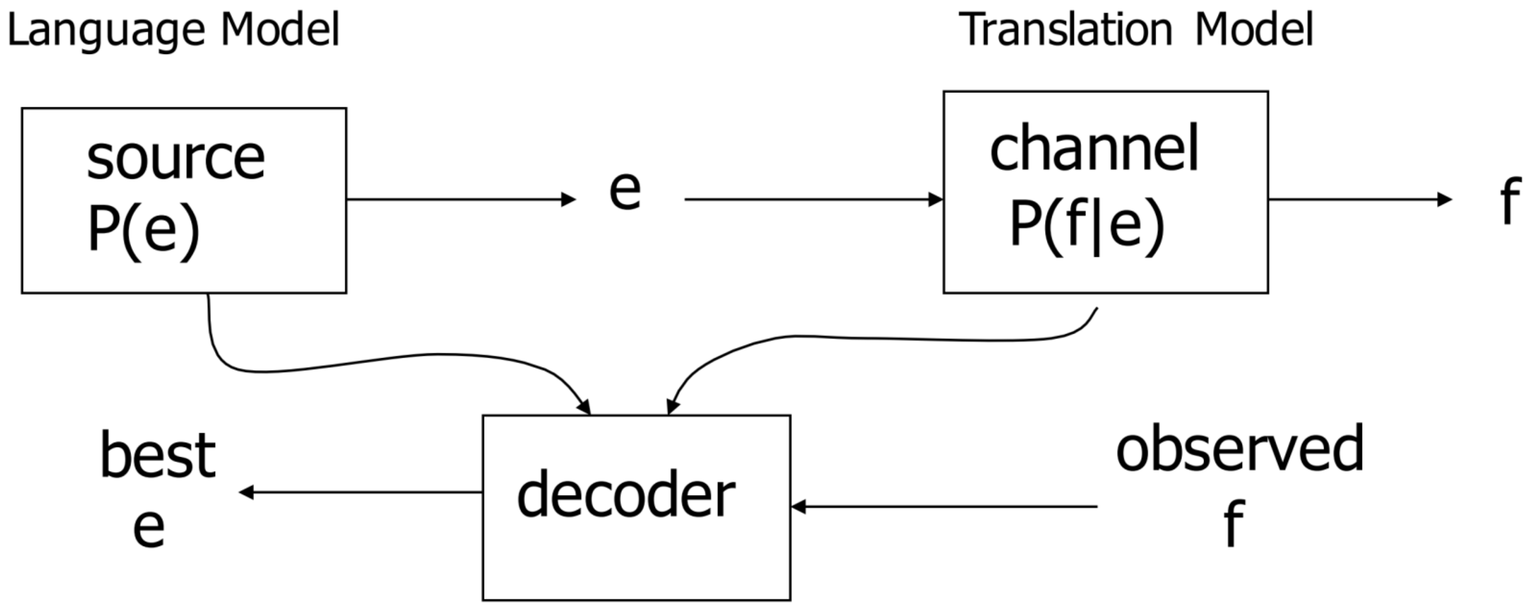
- The noisy channel approach:

$$\begin{aligned} w^* &= \arg\max_w P(w|a) \\ &= \arg\max_w \color{blue}{P(a|w)}\color{red}{P(w)}\color{black}{/P(a)} \\ &= \arg\max_w \underbrace{\color{blue}{P(a|w)}}_{\text{Acoustic Model}}\underbrace{\color{red}{P(w)}}_{\text{Language Model}} \end{aligned}$$

Acoustically Scored Hypotheses

the station signs are in deep in english	-14732
the stations signs are in deep in english	-14735
the station signs are in deep into english	-14739
the station 's signs are in deep in english	-14740
the station signs are in deep in the english	-14741
the station signs are indeed in english	-14757
the station 's signs are indeed in english	-14760
the station signs are indians in english	-14790
the station signs are indian in english	-14799
the stations signs are indians in english	-14807
the stations signs are indians and english	-14815

Statistical Machine Translation uses a Similar Architecture



$$\underset{e}{\operatorname{argmax}}\, P(e|f) = \underset{e}{\operatorname{argmax}}\, P(f|e)P(e)$$

Learning Language Models

- Goal: Assign useful probabilities *P(x)* to sentences x
- Input: many observations of training sentences *x*
- Output: system capable of computing *P(x)*

- Probabilities should broadly indicate plausibility of sentences:
  - *P(I saw a van) >> P(eyes awe of an)*
  - Not only grammaticality: P(artichokes intimidate zippers) ≈ 0
  - In principle, plausibility depends on the domain, context, speaker

Trivial Language Model

- Simplest option: empirical distribution over training sentences…

$$p(x_1 \dots x_n) = \frac{c(x_1 \dots x_n)}{N} \text{ for sentence } x = x_1 \dots x_n$$

- Problem: does not generalize (at all)
- We need to assign non-zero probability to previously unseen sentences!

Unigram Model

- Generative process: pick a word, pick a word, pick a word … until you pick STOP

$$p(x_1 \ldots x_n) = \prod_{i=1}^{n} p(x_i)$$

  - Problem: these models disregard context completely

Bigram Models

- Condition on previous single word
- Generative process:
  - pick START, pick a word conditioned on previous one, repeat until to pick STOP
- The model can be factored thusly:

$$p(x_1 \ldots x_n) = \prod_{i=1}^{n} p(x_i | x_{i-1})$$

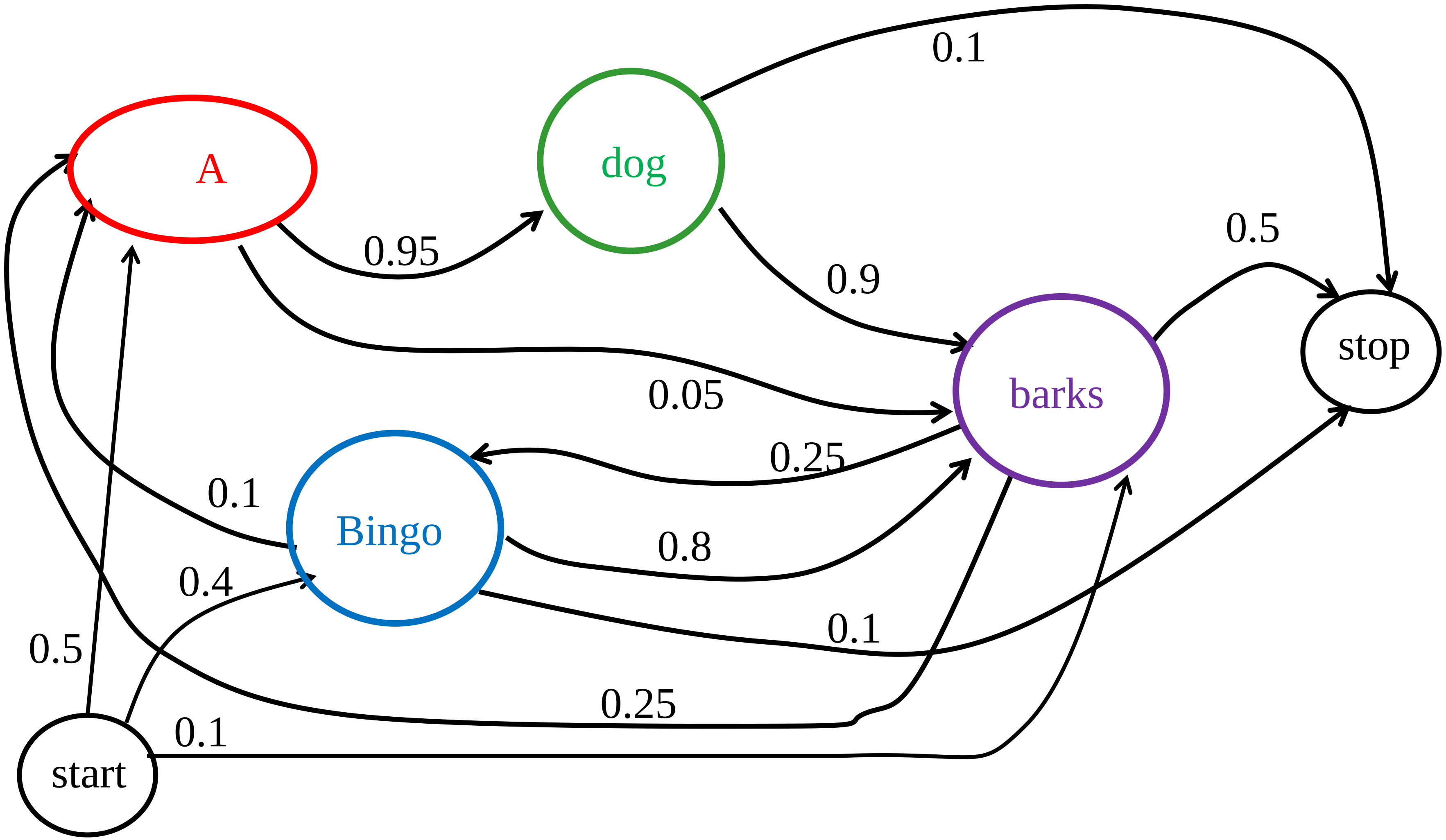
- This is equivalent to assuming the words of the sentence are (homogeneous, 1^{st} order) Markov chains, namely that:

$$p(x_i | x_{i-1} \ldots x_0) = p(x_i | x_{i-1})$$

Markov Model / Markov Chain

- A finite state automaton with probabilistic state transitions
- Makes Markov assumption that next state only depends on the current state and independent of previous history

Sample Markov Model for LM



Higher-order Markov Models can Help

198015222 the first
194623024 the same
168504105 the following
158562063 the world
…
14112454 the door

23135851162 the *

197302 close the window
191125 close the door
152500 close the gap
116451 close the thread
87298 close the deal

3785230 close the *

3380 please close the door
1601 please close the window
1164 please close the new
1159 please close the gate
…
0 please close the first

13951 please close the *

Higher-order Markov Language Models

- k-gram models are equivalent to bi-gram models where the state space (the words) are k-tuples of the bi-gram state space
- So we think of these models as modeling of the transition between k-tuple of states to k-tuple of state, where the first *k-1* coordinates of x_n next tuple must be equal to last *k-1* coordinates of x_{n-1}

$$p(x_1 \ldots x_n) = \prod_{i=1}^{n} q(x_i | x_{i-(k-1)} \ldots x_{i-1})$$

Learning Bigram Language Models

- The parameters of a Markov LM can be represented as a stochastic, positive-valued matrix (*transition matrix*)
  - Sometimes the distribution of the first symbol is represented separately, but we can think about the sequences as always starting with *n* instances of a special START symbol

$$A_{ij} = \Pr(xm = j | x_{m-1} = i)$$

- A maximum likelihood estimator for the transition matrix is obtained by counting:

$$\mathrm{q}_{\mathrm{ML}}(x_m = j | x_{m-1} = i) = \frac{count(i,j)}{\sum_j count(i,j)}$$

Zero Counts

- Training set:

  … denied the allegations

  … denied the reports

  … denied the claims

  … denied the request

- Test set:

  … denied the offer

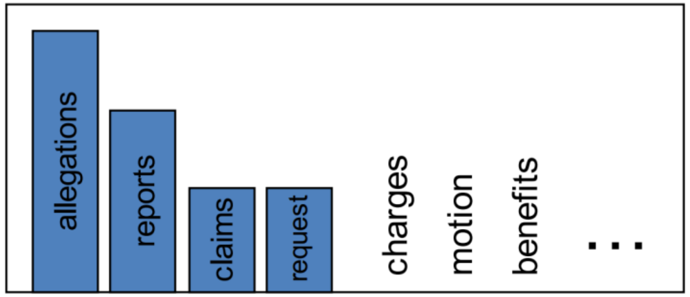
  … denied the loan

$$\mathrm{q}_{\mathrm{ML}}(offer|denied, the) = 0$$

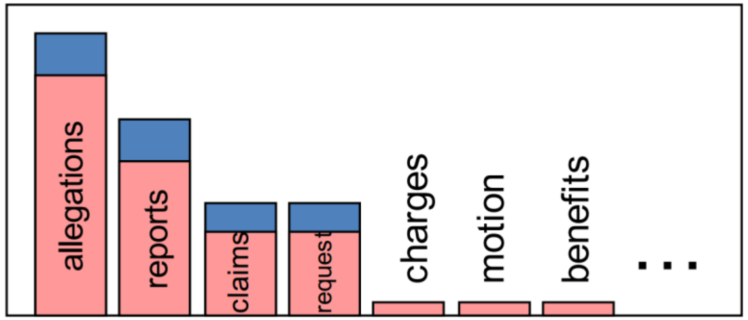
Zero Counts

- But, zero transition probabilities mean that the whole sentence receives a 0 probability!
- Therefore, zero probability estimates are generally avoided
- Methods for assigning probability mass to rare events are often called *smoothing methods*
  - Some have statistical motivation, others don't

P(w | denied the)
3 allegations
2 reports
1 claims
1 request
7 total



P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total



n-gram Smoothing

- The problem is not only zero counts
  - In general, where the event conditioned on has a low probability, the ML estimator is unreliable
- Estimating the probability of an n-gram is a fundamental task in NLP
  - Not only in language models
  - For instance, knowing the common n-grams a word participates in can be telling in terms of its meaning
- The problem becomes worse as n increases
  - The denominator grows exponentially with n

Add-δ (Laplace) Smoothing

- Classic solution: add counts

$$q_{add-\delta}(w) = \frac{c(w) + \delta}{\sum_{w'} (c(w') + \delta)} = \frac{c(w) + \delta}{c() + \delta|\mathcal{V}|}$$

- For a bigram distribution, can add counts shaped like the unigram:

$$q_{uni-\delta}(w|v) = \frac{c(v, w) + \delta q_{ML}(w)}{\sum_{w'} (c(v, w') + \delta q_{ML}(w'))} = \frac{c(v, w) + \delta q_{ML}(w)}{c(v) + \delta}$$

- Problem: doesn’t work well in practice for LMs

Back-off Models

- **The idea:** assume we want to estimate a trigram distribution for a word x_m given its two previous words x_{m-1} and x_{m-2}:

$$\mathrm{q}_{\mathrm{ML}}(x_m = j | x_{m-1} = i, x_{m-2} = l) = \frac{c(l,i,j)}{\sum_j c(l,i,j)}$$

- If the bigram (x_{m-2}, x_{m-1}) didn't appear enough times, we back-off to conditioning only on x_{m-1}:

$$\mathrm{q}_{\mathrm{ML}}(x_m = j | x_{m-1} = i) = \frac{c(i,j)}{\sum_j c(i,j)}$$

- If w_{i-1} is not frequent enough, we back-off to w_i's frequency:

$$\mathrm{q}_{\mathrm{ML}}(xm = j) = \frac{c(j)}{\sum_j c(j)}$$

Back-off: Linear Interpolation

- Interpolate the trigram, bigram and unigram distributions by a convex combination:

$$\begin{aligned} q(w_i \mid w_{i-2}, w_{i-1}) = \quad & \lambda_1 \times q_{\mathsf{ML}}(w_i \mid w_{i-2}, w_{i-1}) \\ & + \lambda_2 \times q_{\mathsf{ML}}(w_i \mid w_{i-1}) \\ & + \lambda_3 \times q_{\mathsf{ML}}(w_i) \end{aligned}$$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$, and $\lambda_i \geq 0$ for all i.

- In order to estimate the λ_i values, we hold out part of the training data, compute the N-gram probabilities q_{ML} on the rest of the training data, and search for the λ_i that yield the highest probability on the held-out set.

Back-off: Linear Interpolation

- That is, maximize *L,* the log-likelihood, where $c'(w_1,w_2,w_3)$ is the number of times the trigram (w_1,w_2,w_3) appeared in the validation set

$$L(\lambda_1, \lambda_2, \lambda_3) = \sum_{w_1,w_2,w_3} c'(w_1, w_2, w_3) \log q(w_3 \mid w_1, w_2)$$

such that $\lambda_1 + \lambda_2 + \lambda_3 = 1$, and $\lambda_i \geq 0$ for all i, and where

$$\begin{aligned} q(w_i \mid w_{i-2}, w_{i-1}) = \quad & \lambda_1 \times q_{\mathsf{ML}}(w_i \mid w_{i-2}, w_{i-1}) \\ & +\lambda_2 \times q_{\mathsf{ML}}(w_i \mid w_{i-1}) \\ & +\lambda_3 \times q_{\mathsf{ML}}(w_i) \end{aligned}$$

Back-off: Linear Interpolation

- It is common to take more ls to get a better fit
- Here is one example of how to do that. Define:

$$\Pi(w_{i-2}, w_{i-1}) = \begin{cases} 1 & \text{If Count}(w_{i-1}, w_{i-2}) = 0 \\ 2 & \text{If } 1 \leq \text{Count}(w_{i-1}, w_{i-2}) \leq 2 \\ 3 & \text{If } 3 \leq \text{Count}(w_{i-1}, w_{i-2}) \leq 5 \\ 4 & \text{Otherwise} \end{cases}$$

- And maximize:

$$\begin{aligned} q(w_i \mid w_{i-2}, w_{i-1}) = \quad & \lambda_1^{\Pi(w_{i-2}, w_{i-1})} \times q_{\text{ML}}(w_i \mid w_{i-2}, w_{i-1}) \\ & + \lambda_2^{\Pi(w_{i-2}, w_{i-1})} \times q_{\text{ML}}(w_i \mid w_{i-1}) \\ & + \lambda_3^{\Pi(w_{i-2}, w_{i-1})} \times q_{\text{ML}}(w_i) \end{aligned}$$

where $\lambda_1^{\Pi(w_{i-2}, w_{i-1})} + \lambda_2^{\Pi(w_{i-2}, w_{i-1})} + \lambda_3^{\Pi(w_{i-2}, w_{i-1})} = 1$,
and $\lambda_i^{\Pi(w_{i-2}, w_{i-1})} \geq 0$ for all i.

Discounting Methods

- Maximum likelihood estimates tend to be too high for low count items:

x	$\text{Count}(x)$	$q_{\text{ML}}(w_i \mid w_{i-1})$
the	48	
the, dog	15	15/48
the, woman	11	11/48
the, man	10	10/48
the, park	5	5/48
the, job	2	2/48
the, telescope	1	1/48
the, manual	1	1/48
the, afternoon	1	1/48
the, country	1	1/48
the, street	1	1/48

Discounting Methods

- One simple way to handle this is to discount all counts by a constant term: (say, 0.5)

- But what do we do with the missing probability mass?

x	Count(x)	Count*(x)	$\frac{\text{Count}^*(x)}{\text{Count(the)}}$
the	48		
the, dog	15	14.5	14.5/48
the, woman	11	10.5	10.5/48
the, man	10	9.5	9.5/48
the, park	5	4.5	4.5/48
the, job	2	1.5	1.5/48
the, telescope	1	0.5	0.5/48
the, manual	1	0.5	0.5/48
the, afternoon	1	0.5	0.5/48
the, country	1	0.5	0.5/48
the, street	1	0.5	0.5/48

Discounting Methods

- In a bigram model, the missing probability mass for a preceding word w_{i-1} is:

$$\alpha(w_{i-1}) = 1 - \sum_{w} \frac{\text{Count}^*(w_{i-1}, w)}{\text{Count}(w_{i-1})}$$

- In our example: $\alpha(w_{i-1}) = 10 \cdot 0.5/48$
- We will distribute this mass to bigrams with count 0
- But how?

Discounting + Unigram Interpolation

- First guess: smooth by reducing some fixed quantity from the bi-gram count, and interpolate with the unigram estimate:

discounted bigram

Interpolation weight

$$P_{Smooth}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - C}{c(w_{i-1})} + \lambda(w_{i-1}) P(w_i)$$

unigram

Kneser-Ney Smoothing

- Better estimate for the probabilities of unigrams:
  - What's the next word: *I can't see without my reading*___________?
  - "Francisco" is more common than "glasses"
  - … but "Francisco" always follows "San"
- So we want to interpolate with a measure of "How likely is *w* to appear as a novel continuation?"
  - For each word, count the number of bigram types it completes
  - Every bigram type was a novel continuation the first time it was seen

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

Kneser-Ney Smoothing

- Properly normalized:

$$P_{CONTINUATION}(w) = \frac{\left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|}{\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|}$$

- A frequent word (“Francisco”) occurring in only one context (“San”) will have a low continuation probability

Kneser-Ney Smoothing

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{CONTINUATION}(w_i)$$

λ is a normalizing constant; the probability mass we've discounted:

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left| \{ w : c(w_{i-1}, w) > 0 \} \right|$$

LM Evaluation

- Assume we have m sentences $s_1, s_2, \ldots, s_m$
- The common way to evaluate language models is by the probability they assign to held-out data
- More exactly, we use *perplexity*:

$$\text{Perplexity} = 2^{-l} \quad \text{where} \quad l = \frac{1}{M} \sum_{i=1}^{m} \log p(s_i)$$

M is the total number of words in the test data

Some Perplexity Values

- Perplexity measures the effective number of possibilities for the next word (on average)
- If our model predicts the probability of a word to be 1/N, regardless of the context (a uniform model), the perplexity will be *N*
- Perplexity values would change considerably between corpora, but they often vary between 50 and 500

- **Extrinsic evaluation** (by applying it to a downstream application) is usually conducted in parallel to perplexity evaluation

Questions?

We’ll also talk about Neural Network-based LM later on