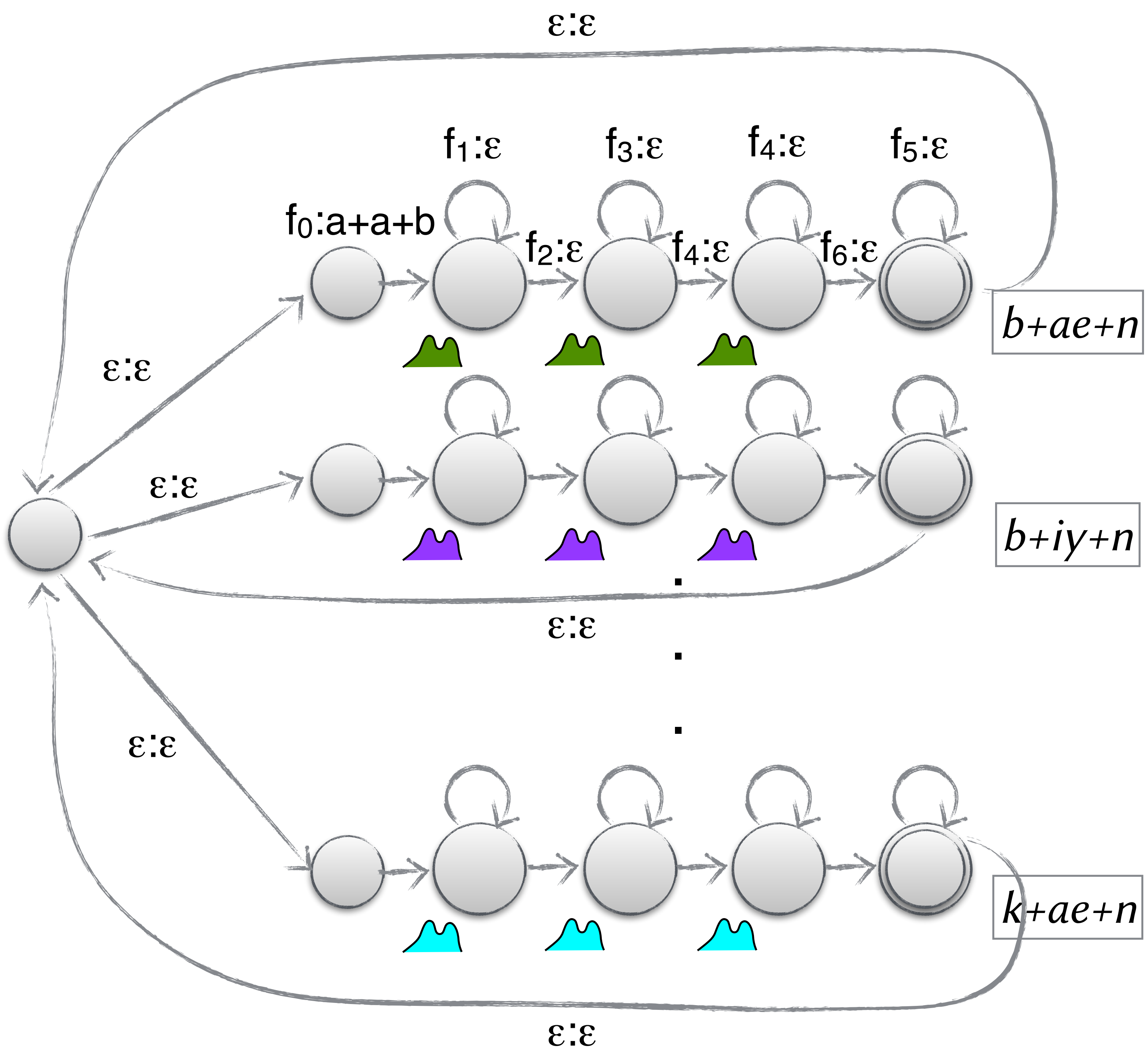
Automatic Speech Recognition (CS753)

Lecture 14: Language Models (Part I)

Instructor: Preethi Jyothi

Feb 27, 2017

So far, acoustic models…



Next, language models

- Language models
  - provide information about word reordering

    Pr("she class taught a") > Pr("she taught a class")

  - provide information about the most likely next word

    Pr("she taught a class") > Pr("she taught a speech")

Application of language models

- Speech recognition
  - Pr("she taught a class") > Pr("sheet or tuck lass")
- Machine translation
- Handwriting recognition/Optical character recognition
- Spelling correction of sentences
- Summarization, dialog generation, information retrieval, etc.

Popular Language Modelling Toolkits

- SRILM Toolkit:

  http://www.speech.sri.com/projects/srilm/

- KenLM Toolkit:

  https://kheafield.com/code/kenlm/

- OpenGrm NGram Library:

  http://opengrm.org/

Introduction to probabilistic LMs

Probabilistic or Statistical Language Models

- Given a word sequence, $W = \{w_1, \dots, w_n\}$, what is $\Pr(W)$?
- Decompose $\Pr(W)$ using the chain rule:

$$\Pr(w_1,w_2,\ldots,w_{n-1},w_n) = \Pr(w_1)\,\Pr(w_2|w_1)\,\Pr(w_3|w_1,w_2)\ldots\Pr(w_n|w_1,\ldots,w_{n-1})$$

- Sparse data with long word contexts: How do we estimate the probabilities $\Pr(w_n|w_1,\ldots,w_{n-1})$?

Estimating word probabilities

- Accumulate counts of words and word contexts
- Compute normalised counts to get word probabilities
- E.g. Pr("class | she taught a")

$$= \frac{\pi(\text{"she taught a class"})}{\pi(\text{"she taught a"})}$$

where π("…") refers to counts derived from a large English text corpus

- What is the obvious limitation here? We'll never see enough data

Simplifying Markov Assumption

- Markov chain:
  - Limited memory of previous word history: Only last m words are included

- 2-order language model (or bigram model)

$$\Pr(w_1, w_2, \ldots, w_{n-1}, w_n) \cong \Pr(w_1) \Pr(w_2|w_1) \Pr(w_3|w_2) \ldots \Pr(w_n|w_{n-1})$$

- 3-order language model (or trigram model)

$$\Pr(w_1, w_2, \ldots, w_{n-1}, w_n) \cong \Pr(w_1) \Pr(w_2|w_1) \Pr(w_3|w_1, w_2) \ldots \Pr(w_n|w_{n-2}, w_{n-1})$$

- Ngram model is an N-1th order Markov model

Estimating Ngram Probabilities

- Maximum Likelihood Estimates
  - Unigram model

$$\Pr_{ML}(w_1) = \frac{\pi(w_1)}{\sum_i \pi(w_i)}$$

  - Bigram model

$$\Pr_{ML}(w_2|w_1) = \frac{\pi(w_1, w_2)}{\sum_i \pi(w_1, w_i)}$$

$$\Pr(s = w_0, \ldots, w_n) = \Pr_{ML}(w_0) \prod_{i=1}^{n} \Pr_{ML}(w_i|w_{i-1})$$

Example

The dog chased a cat
The cat chased away a mouse
The mouse eats cheese

What is Pr("The cat chased a mouse")?

Pr("The cat chased a mouse") =

Pr("The") · Pr("cat|The") · Pr("chased|cat") · Pr("a|chased") · Pr("mouse|a") =

$3/15 \cdot 1/3 \cdot 1/1 \cdot 1/2 \cdot 1/2 = 1/60$

Example

The dog chased a cat
The cat chased away a mouse
The mouse eats cheese

What is Pr("The dog eats meat")?

Pr("The dog eats meat") =

Pr("The") · Pr("dog | The") · Pr("eats | dog") · Pr("meat | eats") =

$3/15 \cdot 1/3 \cdot 0/1 \cdot 0/1 = 0!$ Due to unseen bigrams

How do we deal with unseen bigrams? We'll come back to it.

Open vs. closed vocabulary task

- Closed vocabulary task: Use a fixed vocabulary, V. We know all the words in advance.
- More realistic setting, we don't know all the words in advance. Open vocabulary task. Encounter out-of-vocabulary (OOV) words during test time.
- Create an unknown word: <UNK>
    - Estimating <UNK> probabilities: Determine a vocabulary V. Change all words in the training set not in V to <UNK>
    - Now train its probabilities like a regular word
    - At test time, use <UNK> probabilities for words not in training

Evaluating Language Models

- Extrinsic evaluation:
  - To compare Ngram models A and B, use both within a specific speech recognition system (keeping all other components the same)
  - Compare word error rates (WERs) for A and B
  - Time-consuming process!

Intrinsic Evaluation

- Evaluate the language model in a standalone manner
- How likely does the model consider the text in a test set?
- How closely does the model approximate the actual (test set) distribution?
    - Same measure can be used to address both questions — perplexity!

Measures of LM quality

- How likely does the model consider the text in a test set?
- How closely does the model approximate the actual (test set) distribution?
    - Same measure can be used to address both questions — perplexity!

Perplexity (I)

- How likely does the model consider the text in a test set?
  - Perplexity(test) = $1/\Pr_{model}[\text{text}]$
  - Normalized by text length:
    - Perplexity(test) = $(1/\Pr_{model}[\text{text}])^{1/N}$ where N = number of tokens in test
    - e.g. If model predicts i.i.d. words from a dictionary of size L, per word perplexity = $(1/(1/L)^N)^{1/N} = L$

Intuition for Perplexity

- Shannon's guessing game builds intuition for perplexity
  - What is the surprisal factor in predicting the next word?
    - At the stall, I had tea and __________

biscuits	*0.1*
samosa	*0.1*
coffee	*0.01*
rice	*0.001*
⋮	
but	*0.00000000001*

- A better language model would assign a higher probability to the actual word that fills the blank (and hence lead to lesser surprisal/perplexity)

Measures of LM quality

- How likely does the model consider the text in a test set?
- How closely does the model approximate the actual (test set) distribution?
  - Same measure can be used to address both questions — perplexity!

Perplexity (II)

- How closely does the model approximate the actual (test set) distribution?
    - KL-divergence between two distributions X and Y
    $$D_{KL}(X||Y) = \Sigma_\sigma \Pr_X[\sigma] \log (\Pr_X[\sigma]/\Pr_Y[\sigma])$$
    - Equals zero iff X = Y ; Otherwise, positive

- How to measure $D_{KL}(X||Y)$? We don't know X!
    - $D_{KL}(X||Y) = \boxed{\Sigma_\sigma \Pr_X[\sigma] \log(1/\Pr_Y[\sigma])} - H(X)$
    where $H(X) = -\Sigma_\sigma \Pr_X[\sigma] \log \Pr_X[\sigma]$
    - Empirical cross entropy:
    $$\frac{1}{|test|} \sum_{\sigma \in test} \log(\frac{1}{\Pr_y[\sigma]})$$

Cross entropy between X and Y

Perplexity vs. Empirical Cross Entropy

- Empirical Cross Entropy (ECE)

$$\frac{1}{|\#sents|}\sum_{\sigma\in test}\log(\frac{1}{\Pr_{model}[\sigma]})$$

- Normalized Empirical Cross Entropy = ECE/(avg. length) =

$$\frac{1}{|\#words/\#sents|}\frac{1}{|\#sents|}\sum_{\sigma\in test}\log(\frac{1}{\Pr_{model}[\sigma]}) =$$
$$\frac{1}{N}\sum_{\sigma}\log(\frac{1}{\Pr_{model}[\sigma]})$$

- How does $\frac{1}{N}\sum_{\sigma}\log(\frac{1}{\Pr_{model}[\sigma]})$ relate to perplexity?

Perplexity vs. Empirical Cross-Entropy

$$\begin{aligned}\log(\text{perplexity}) &= \frac{1}{N}\log\frac{1}{\Pr[test]}\\ &= \frac{1}{N}\log\prod_{\sigma}\left(\frac{1}{\Pr_{model}[\sigma]}\right)\\ &= \frac{1}{N}\sum_{\sigma}\log\left(\frac{1}{\Pr_{model}[\sigma]}\right)\end{aligned}$$

Thus, perplexity = $2^{(\text{normalized cross entropy})}$

Example perplexities for Ngram models trained on WSJ (80M words):

Unigram: 962, Bigram: 170, Trigram: 109

Introduction to smoothing of LMs

Recall example

The dog chased a cat
The cat chased away a mouse
The mouse eats cheese

What is Pr("The dog eats meat")?

Pr("The dog eats meat") =

Pr("The") · Pr("dog|The") · Pr("eats|dog") · Pr("meat|eats") =

3/15 · 1/3 · 0/1 · 0/1 = 0!   Due to unseen bigrams

Unseen Ngrams

- Even with MLE estimates based on counts from large text corpora, there will be many unseen bigrams/trigrams that never appear in the 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

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

Correct?

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

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

No, $\Sigma_{w_i}\Pr_{Lap}(w_i|w_{i\text{-}1})$ must equal 1. Change denominator s.t.

$$\sum_{w_i} \frac{\pi(w_{i-1}, w_i) + 1}{\pi(w_{i-1}) + x} = 1$$

Solve for x: $x = V$ where V is the vocabulary size

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

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

Backoff

- In a backoff model, if the Ngram has zero counts, we backoff to the (N-1)gram or lower order Ngram models

- Katz Backoff:

$$P_{\mathrm{BO}}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n|w_{n-N+1}^{n-1}), & \text{if } C(w_{n-N+1}^{n}) > 0 \\ \alpha(w_{n-N+1}^{n-1})P_{\mathrm{BO}}(w_n|w_{n-N+2}^{n-1}), & \text{otherwise.} \end{cases}$$

- where $P^*(w_n|w_{n-N+1}^{n-1})$ is the discounted probability and α's are appropriately normalised backoff weights

Interpolation

- Linear interpolation: Linear combination of different Ngram models

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

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

- Instead of a fixed value, λ's could also be conditioned on the context

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

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

- Instead of a fixed value, λ's could also be conditioned on the context

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

Estimate N-gram probabilities on a training set. Then, search for λ's that maximise the probability of a held-out set, $\Sigma_n \log \hat{P}(w_n|w_{n-1})$

Smoothing for Web-scale N-grams

- "Stupid backoff" [B07]
- Don't apply any discounting and instead directly use relative counts
- Works well on very large web-scale datasets

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if} \quad \text{count}(w_{i-k+1}^{i}) > 0 \\ 0.4 S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

[B07]: Brants et al., "Large Language Models in Machine Translation", ACL, 2007

Next class: Advanced Smoothing & Beyond Ngram LMs