Recap: Smoothing for Language Models

- N-gram LMs reduce sparsity by assuming each word only depends on a fixed-length history.
- But even this assumption isn’t enough: we still encounter lots of unseen N-grams in a test set or new corpus.
- If we use MLE, we’ll assign 0 probability to unseen items: useless as an LM.
- **Smoothing** solves this problem: move probability mass from seen items to unseen items.

Smoothing Methods so Far

- **Add-\(\alpha\) smoothing**: \((\alpha = 1 \text{ or } < 1)\) very simple, but no good when vocabulary size is large.
- **Good-Turing smoothing**:
  - If \(n\) is count of history, then \(P_{GT} = \frac{c^*}{n}\) where

\[
  c^* = (c + 1) \frac{N_{c+1}}{N_c}
\]

  - \(N_c\) number of N-grams that occur exactly \(c\) times in corpus
  - \(N_0\) total number of unseen N-grams
  - Ex. for trigram probability \(P_{GT}(\text{three} | \text{I spent})\), then \(n\) is count of \text{I spent} and \(c\) is count of \text{I spent three}.

Good-Turing smoothing
Problems with Good-Turing

• Assumes we know the vocabulary size (no unseen words) [but again, use UNK: see J&M 4.3.2]

• Doesn’t allow “holes” in the counts (if $N_i > 0$, $N_{i-1} > 0$) [can estimate using linear regression: see J&M 4.5.3]

• Applies discounts even to high-frequency items [but see J&M 4.5.3]

• But there’s a more fundamental problem...

Remaining problem

• In training corpus, suppose we see Scottish beer but neither of
  – Scottish beer drinkers
  – Scottish beer eaters

• If we build a trigram model smoothed with Add-\(\alpha\) or G-T, which example has higher probability?

Interpolation

• Combine higher and lower order \(N\)-gram models, since they have different strengths and weaknesses:
  – high-order \(N\)-grams are sensitive to more context, but have sparse counts
  – low-order \(N\)-grams have limited context, but robust counts

• If \(P_N\) is \(N\)-gram estimate (from MLE, GT, etc; \(N = 1 − 3\)), use:

\[
P_{\text{INT}}(w_3|w_1, w_2) = \lambda_1 P_1(w_3) + \lambda_2 P_2(w_3|w_2) + \lambda_3 P_3(w_3|w_1, w_2)
\]

\[
P_{\text{INT}}(\text{three}|\text{I, spent}) = \lambda_1 P_1(\text{three}) + \lambda_2 P_2(\text{three}|\text{spent}) + \lambda_3 P_3(\text{three}|\text{I, spent})
\]
Interpolation

- Note that $\lambda_i$s must sum to 1:

$$1 = \sum_{w_3} P_{\text{INT}}(w_3|w_1, w_2)$$

$$= \sum_{w_3} [\lambda_1 P_1(w_3) + \lambda_2 P_2(w_3|w_2) + \lambda_3 P_3(w_3|w_1, w_2)]$$

$$= \lambda_1 \sum_{w_3} P_1(w_3) + \lambda_2 \sum_{w_3} P_2(w_3|w_2) + \lambda_3 \sum_{w_3} P_3(w_3|w_1, w_2)$$

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

Fitting the interpolation parameters

- In general, any weighted combination of distributions is called a mixture model.

- So $\lambda_i$s are interpolation parameters or mixture weights.

- The values of the $\lambda_i$s are chosen to optimize perplexity on a held-out data set.

Katz Back-Off

- Solve the problem in a similar way to Good-Turing smoothing.

- Discount the trigram-based probability estimates.

- This leaves some probability mass to share among the estimates from the lower-order model(s).

- Katz backoff: Good-Turing discount the observed counts, but

- instead of distributing that mass uniformly over unseen items, use it for backoff estimates.

Back-Off Formulae

- Trust the highest order language model that contains $N$-gram

$$P_{BO}(w_i|w_{i-N+1}, \ldots, w_{i-1}) =$$

$$P^*(w_i|w_{i-N+1}, \ldots, w_{i-1})$$

if $\text{count}(w_{i-N+1}, \ldots, w_i) > 0$

$$\alpha(w_{i-N+1}, \ldots, w_{i-1}) P_{BO}(w_i|w_{i-N+2}, \ldots, w_{i-1})$$

else
Back-Off

- Requires
  - adjusted prediction model $P^*(w_i|w_{i-N+1}, ..., w_{i-1})$
  - backoff weights $\alpha(w_1, ..., w_{N-1})$

- Exact equations get complicated to make probabilities sum to 1.
- See textbook for details if interested.

Diversity of histories matters!

- A real example: the word York
  - fairly frequent word in Europarl corpus, occurs 477 times
  - as frequent as foods, indicates and providers
  - in unigram language model: a respectable probability

- However, it almost always directly follows New (473 times)

- So, in unseen bigram contexts, York should have low probability
  - lower than predicted by unigram model as used in interpolation/backoff.

Do our smoothing methods work here?

Example from MacKay and Peto (1995):

Imagine, you see, that the language, you see, has, you see, a frequently occurring couplet, ‘you see’, you see, in which the second word of the couplet, see, follows the first word, you, with very high probability, you see. Then the marginal statistics, you see, are going to become hugely dominated, you see, by the words you and see, with equal frequency, you see.

- $P(\text{see})$ and $P(\text{you})$ both high, but see nearly always follows you.
- So $P(\text{see}|\text{novel})$ should be much lower than $P(\text{you}|\text{novel})$.

Kneser-Ney Smoothing

- Kneser-Ney smoothing takes diversity of histories into account
- Count of distinct histories for a word:

$$N_{1+}(\bullet w_i) = |\{w_{i-1} : c(w_{i-1}, w_i) > 0\}|$$

- Recall: maximum likelihood est. of unigram language model:

$$P_{ML}(w_i) = \frac{C(w_i)}{\sum w C(w)}$$

- In KN smoothing, replace raw counts with count of histories:

$$P_{KN}(w_i) = \frac{N_{1+}(\bullet w_i)}{\sum w N_{1+}(\bullet w)}$$
Kneser-Ney in practice

- Original version used backoff, later “modified Kneser-Ney” introduced using interpolation.

- Fairly complex equations, but until recently the best smoothing method for word n-grams.

- See Chen and Goodman (1999) for extensive comparisons of KN and other smoothing methods.

- KN (and other methods) implemented in language modelling toolkits like SRILM (classic), KenLM (good for really big models), OpenGrm Ngram library (uses finite state transducers), etc.

Are we done with smoothing yet?

We’ve considered methods that predict rare/unseen words using

- Uniform probabilities (add-α, Good-Turing)

- Probabilities from lower-order n-grams (interpolation, backoff)

- Probability of appearing in new contexts (Kneser-Ney)

What’s left?

Word similarity

- Suppose we have two words with \( C(w_1) \gg C(w_2) \)
  - salmon
  - swordfish

- Can \( P(\text{salmon}|\text{caught two}) \) tell us anything about \( P(\text{swordfish}|\text{caught two}) \)?

- \( N \)-gram models: no.

Word similarity in language modeling

- Early version: class-based language models (J&M 4.9.2)
  - Define classes \( c \) of words, by hand or automatically
  - \( P_{\text{CL}}(w_i|w_{i-1}) = P(c_i|c_{i-1})P(w_i|c_i) \) (an HMM)

- Recent version: distributed language models
  - Use neural networks to project words into a continuous space, so words that appear in similar contexts have similar representations (e.g., Mikolov 2012).

Distributed word representations

• Each word represented as high-dimensional vector (50-500 dims)
  E.g., salmon is [0.1, 2.3, 0.6, −4.7, . . .]

• Similar words represented by similar vectors
  E.g., swordfish is [0.3, 2.2, 1.2, −3.6, . . .]

• More about this later in the course.

Training the model

• Goal: learn word representations (embeddings) such that words that behave similarly are close together in high-dimensional space.

• 2-dimensional example:

Using the model

Want to compute $P(w_1 \ldots w_n)$ for a new sequence.

• N-gram LM: again, relatively quick
• distributed LM: often prohibitively slow for real applications
• An active area of research for distributed LMs

Training the model

• N-gram LM: collect counts, maybe optimize some parameters
  – (Relatively) quick, especially these days (minutes-hours)

• distributed LM: learn the representation for each word
  – Use ML methods like neural networks that iteratively improve embeddings
  – Can be extremely time-consuming (hours-days)
  – Learned embeddings capture both semantic and syntactic similarity.
Other Topics in Language Modeling

Many active research areas!

- Modeling issues:
  - Morpheme-based language models: *preempt* vs. *preregistration*
  - Syntactic language models (more later)
  - Domain adaptation: when only a small corpus is available in the domain of interest

- Implementation issues:
  - Speed: both to train, and to use in real-time applications like translation and speech recognition.
  - Disk space and memory: especially important for mobile devices

---

### Noisy channel model

- We imagine that someone tries to communicate a sequence to us, but noise is introduced. We only see the output sequence.

\[
\begin{align*}
\text{symbol sequence} & \quad \text{noisy/errorful encoding} & \quad \text{output sequence} \\
\ P(Y) & \quad P(X|Y) & \quad P(X)
\end{align*}
\]

Application

<table>
<thead>
<tr>
<th>Y</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speech recognition</td>
<td>words in ( L_1 )</td>
</tr>
<tr>
<td>Machine translation</td>
<td>intended words in ( L_2 )</td>
</tr>
<tr>
<td>Spelling correction</td>
<td>typed words</td>
</tr>
</tbody>
</table>

---

Noisy channel model

- We imagine that someone tries to communicate a sequence to us, but noise is introduced. We only see the output sequence.

\[
\begin{align*}
\text{symbol sequence} & \quad \text{noisy/errorful encoding} & \quad \text{output sequence} \\
\ P(Y) & \quad P(X|Y) & \quad P(X)
\end{align*}
\]
Example: spelling correction

- \( P(Y) \): Distribution over the words the user intended to type. A language model!
- \( P(X|Y) \): Distribution describing what user is likely to type, given what they meant. Could incorporate information about common spelling errors, key positions, etc. Call it a noise model.
- \( P(X) \): Resulting distribution over what we actually see.
- Given some particular observation \( x \) (say, effert), we want to recover the most probable \( y \) that was intended.

Noisy channel as probabilistic inference

- Mathematically, what we want is \( \text{argmax}_y P(y|x) \).
  - Read as “the \( y \) that maximizes \( P(y|x) \)”
- Rewrite using Bayes’ Rule:
  \[
  \text{argmax}_y P(y|x) = \text{argmax}_y \frac{P(x|y)P(y)}{P(x)} = \text{argmax}_y P(x|y)P(y)
  \]

You may be wondering

- If we can train \( P(X|Y) \), why can’t we just train \( P(Y|X) \)? Who needs Bayes’ Rule?
  - Answer 1: sometimes we do train \( P(Y|X) \) directly. Stay tuned...
  - Answer 2: training \( P(X|Y) \) or \( P(Y|X) \) requires input/output pairs, which are often limited:
    - Misspelled words with their corrections; transcribed speech; translated text
    
    But LMs can be trained on huge unannotated corpora: a better model. Can help improve overall performance.
Model versus algorithm

- We defined a probabilistic model, which says what we should do.
  - E.g., for spelling correction: given trained LM and noise model (we haven’t said yet how to acquire the noise model), find the intended word that is most probable given the observed word.
- We haven’t considered how we would do that.
  - A search problem: there are (infinitely) many possible inputs that could have generated what we saw; which one is best?
  - We need to design an algorithm that can solve the problem.

Summary

- Different smoothing methods account for different aspects of sparse data and word behaviour.
  - Interpolation/backoff: leverage advantages of both higher and lower order N-grams.
  - Kneser-Ney smoothing: accounts for diversity of history.
  - Distributed representations: account for word similarity.
- Noisy channel model combines LM with noise model to define the “best” solution for many applications.
- Next time, will consider how to find this solution.