Natural Language Processing
CS 6840

Lecture 02

Razvan C. Bunescu
School of Electrical Engineering and Computer Science
bunescu@ohio.edu
Language Models i.e. N-gram Models

• **Next word prediction:**
  - Sue swallowed the large green ________
    - tree, car, desk, pill, frog, alien, …
  - How? Predict next word \( w^* \) given history \( w_1, \ldots, w_{n-1} \):
    \[
    w^* = \arg \max_{w_n \in V} p(w_n \mid w_1, \ldots, w_{n-1})
    \]

• **Sentence probability estimation:**
  - \( p(“Sue swallowed the large green pill”) = ? \)
  - (related to) / (reformulation of) next word prediction.
    \[
    p(w_1, \ldots, w_n) = p(w_1) p(w_2 \mid w_1) p(w_3 \mid w_1, w_2) \ldots p(w_n \mid w_1, \ldots, w_{n-1})
    \]
N-Gram Models

• Predict next word $w^*$ given history $w_1, \ldots, w_{n-1}$:
  
  $$w^* = \arg \max_{w_n \in V} p(w_n | w_1, \ldots, w_{n-1})$$

  – Unigram Model: $p(w_n)$
  – Bigram Model: $p(w_n | w_{n-1})$
  – Trigram Model: $p(w_n | w_{n-2}, w_{n-1})$

• Building N-Gram models:
  – How can we estimate $p(w_n | w_1, \ldots, w_{n-1})$?
N-gram Models: Statistical Estimation

- Reduce to estimating the probability distribution of n-grams:
  \[ p(w_n | w_1, \ldots, w_{n-1}) = \frac{p(w_1, \ldots, w_n)}{p(w_1, \ldots, w_{n-1})} \]

- Use a training text to estimate n-gram distributions:
  - Assume sequence of \( N \) words:
    - pad with \( n-1 \) dummy symbols to the beginning \( \Rightarrow N \) n-grams.
  - Assume the occurrence of each particular n-gram is a random variable with Bernoulli distribution (quite untrue, but usable!).
  - Use **Maximum Likelihood Estimate** (here, *relative frequency*):
    \[ p(w_1, \ldots, w_n) = \frac{C(w_1, \ldots, w_n)}{N} \]
    freq. of n-gram in training corpus
N-gram Models: MLE

\[ p(w_1, \ldots, w_n) = \frac{C(w_1, \ldots, w_n)}{N} \]

Why is this only an estimate?

\[ \Rightarrow p(w_n \mid w_1, \ldots, w_{n-1}) = \frac{C(w_1, \ldots, w_n)}{C(w_1, \ldots, w_{n-1})} \]

- MLE is in general unsuitable for statistical inference in NLP:
  - Sparsity leads to unreliable estimates for rare events!
  - Can alleviate sparsity effects (but not eliminate):
    - Increase \( N \) (corpus size) – how much for sufficient trigram counts?
    - Decrease \( n \) (history size) – how much for accurate probabilities?
      - go from unlimited to limited history/memory.
Zipf’s Law

Frequency vs. rank for all words in Moby Dick
Zipf’s Law (Log Scale)

Moby Dick:
- 44% hapax legomena
- 17% dis legomena

“Honorificabilitudinitatibus”:
- Shakespeare’s *hapax legomenon*
- longest word with alternating vowels and consonants
Markov Chains

• Assume next word depends only on $k$ previous words:
  – $k^{th}$ order Markov assumption
    \[ p(w_n \mid w_1, \ldots, w_{n-1}) \approx p(w_n \mid w_{n-k}, \ldots, w_{n-1}) \]
  – $1^{st}$ order Markov model:
    \[ p(w_n \mid w_1, \ldots, w_{n-1}) \approx p(w_n \mid w_{n-1}) \iff p(w_1, \ldots, w_n) \approx ? \]

• Assume vocabulary $|V| = 30K$:
  – 0-gram LM (uniform) \( \Rightarrow \) 1 params: $p(w) = 1/|V|$,
  – 1-gram LM (unigram) \( \Rightarrow \) $3 \times 10^4$ params: $p(w_n)$
  – 2-gram LM (bigram) \( \Rightarrow \) $9 \times 10^8$ params: $p(w_n \mid w_{n-1})$
  – 3-gram LM (trigram) \( \Rightarrow \) $2.7 \times 10^{13}$ params: $p(w_n \mid w_{n-2}, w_{n-1})$
Out Of Vocabulary (OOV) Words

Also, it is not practical to calculate n-grams for all words. Possible approaches:

1) Use only the most common words:
   - replace 1-count words (hapax legomena) by <UNK>.
     • half of the types, but only a fraction of the tokens (Zipf’s law).

2) Use a predefined vocabulary.
   - replace OOV words by <UNK>.

3) Replace the first occurrence of every word type in the training data by <UNK>.
Text Preparation

• Separate punctuation, include as words.

• Case sensitive or case insensitive:
  – better: use name recognition.

• Replace numbers with <num>.

• Include sentence boundaries:
  <s> Nothing exists except atoms and empty space . </s>
  <s> Everything else is opinion . </s>
  <s> Democritus ( <num> BC - <num> BC ) </s>
Example: MLE Bigram Model

<s> Nothing exists except atoms and empty space . </s>
<s> Everything else is opinion . </s>
<s> A word is a shadow of a deed . </s>
<s> A word is a shadow of a deed . </s>
<s> A word is a shadow of a deed . </s>
<s> Democritus ( <num> BC - <num> BC ) </s>

\[
p(\text{nothing} \mid <s>) \quad p(a \mid \text{is}) \quad p(\text{empty} \mid \text{and})
\]

\[
p(\text{shadow} \mid a)
\]

\[
p(\text{deed} \mid \text{shadow of a})
\]
Berkeley Restaurant Project Corpus

- Questions about a database of restaurants:
  - can you tell me about any good cantonese restaurants close by
  - mid priced thai food is what i’m looking for
  - tell me about chez panisse
  - can you give me a listing of the kinds of food that are available
  - i’m looking for a good place to eat breakfast
  - when is caffe venezia open during the day
  - ...

- Vocabulary $|V| = 1446$
- 9332 sentences.
### Unigam and Bigram Counts

<table>
<thead>
<tr>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>2533</td>
<td>927</td>
<td>2417</td>
<td>746</td>
<td>158</td>
<td>1093</td>
<td>341</td>
<td>278</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>5</td>
<td>827</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>want</td>
<td>2</td>
<td>0</td>
<td>608</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>to</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>686</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>211</td>
</tr>
<tr>
<td>eat</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>16</td>
<td>2</td>
<td>42</td>
<td>0</td>
</tr>
<tr>
<td>chinese</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>82</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>food</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lunch</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spend</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Bigram Probabilities: MLE

- Divide bigram counts by prefix unigram counts to get MLE estimates of probabilities.

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0.002</td>
<td>0.33</td>
<td>0</td>
<td>0.0036</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00079</td>
</tr>
<tr>
<td>want</td>
<td>0.0022</td>
<td>0</td>
<td>0.66</td>
<td>0.0011</td>
<td>0.0065</td>
<td>0</td>
<td>0</td>
<td>0.0054</td>
</tr>
<tr>
<td>to</td>
<td>0.00083</td>
<td>0</td>
<td>0.0027</td>
<td>0.28</td>
<td>0.00083</td>
<td>0</td>
<td>0</td>
<td>0.0025</td>
</tr>
<tr>
<td>eat</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.021</td>
<td>0</td>
<td>0</td>
<td>0.056</td>
</tr>
<tr>
<td>chinese</td>
<td>0.0063</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>food</td>
<td>0.014</td>
<td>0</td>
<td>0.014</td>
<td>0</td>
<td>0.00092</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lunch</td>
<td>0.0059</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spend</td>
<td>0.0036</td>
<td>0</td>
<td>0.0036</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Shannon’s Game

- Sample a random bigram ($<s>$, $w$) according to its probability
- **do:**
  - **Sample** a random bigram ($w$, $x$) according to its probability
  - **set** $w \leftarrow x$
- **until** we sample a bigram ($w$, $<s>$)

I want to eat Chinese food

How exactly do we sample?
Shannon’s Game: Shakespeare

- \( N = 884,647 \) tokens, \( V = 29,066 \) types
- Shakespeare produced 300,000 bigram types out of \( V^2 = 844 \) million possible bigrams...
  - **Sparsity**: 99.96% of the possible bigrams were never seen (have zero entries in the table)
  - This is the biggest problem in language modeling!
- Possible quadrigrams \( V^4 = 7 \times 10^{17} \):
  - only five possible continuations of “It cannot be but”:
    - that, I, he, thou, so
  \[ \Rightarrow \] generated text will look a lot like Shakespeare.
# Shannon’s Game: Shakespeare

<table>
<thead>
<tr>
<th>Unigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>• To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have</td>
</tr>
<tr>
<td>• Every enter now severally so, let</td>
</tr>
<tr>
<td>• Hill he late speaks; or! a more to leg less first you enter</td>
</tr>
<tr>
<td>• Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>• What means, sir. I confess she? then all sorts, he is trim, captain.</td>
</tr>
<tr>
<td>• Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.</td>
</tr>
<tr>
<td>• What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?</td>
</tr>
<tr>
<td>• Enter Menenius, if it so many good direction found’st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Sweet prince, Falstaff shall die. Harry of Monmouth’s grave.</td>
</tr>
<tr>
<td>• This shall forbid it should be branded, if renown made it empty.</td>
</tr>
<tr>
<td>• Indeed the duke; and had a very good friend.</td>
</tr>
<tr>
<td>• Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ’tis done.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quadrigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>• King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in;</td>
</tr>
<tr>
<td>• Will you not tell me who I am?</td>
</tr>
<tr>
<td>• It cannot be but so.</td>
</tr>
<tr>
<td>• Indeed the short and the long. Marry, ’tis a noble Lepidus.</td>
</tr>
</tbody>
</table>
Shannon’s Game: WSJ

unigram: Months the my and issue of year foreign new exchange’s september were recession exchange new endorsed a acquire to six executives

bigram: Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

trigram: They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions
Building and Evaluating LMs

• Split corpus into training, held-out/development, test:
  – Training used to estimate the parameters.
  – Held-out used to tune hyper-parameters.
  – Test used to evaluate the model.
  – Typical split: 80%, 10%, 10%.

• Evaluation:
  – Intrinsic:
    • perplexity
    • cross-entropy
Intrinsic Evaluation: Perplexity

• The normalized inverse of the probability of the test set:

\[ PP(W) = P(w_1w_2\ldots w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1w_2\ldots w_N)}} \]

• Minimize perplexity ⇔ Maximize probability of test set.

\[ PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1\ldots w_{i-1})}} \]

\[ PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}} \]

chain rule

bigram model

Lecture 01
Sparsity: Low to Zero Counts

• Shakespeare corpus:
  – 300,000 bigram types out of $V^2 = 844$ million possible bigrams.
  – 99.96% of the possible bigrams were never seen (have zero entries in the table).

  • worse for: trigrams, quadrigrams, …

\[
p(w_n \mid w_{n-2}, w_{n-1}) = \frac{C(w_{n-2}, w_{n-1}, w_n)}{C(w_{n-2}, w_{n-1})} = \begin{cases} 
  NaN, \text{ for unseen bigram} \\
  0, \text{ for unseen trigram}
\end{cases}
\]

⇒ $p(w_1, \ldots, w_n)$ is 0, or undefined ⇒ unusable LM
Sparsity

• Types of zeros:
  – Legitimate: Things that really can’t or shouldn’t happen.
  – Rare events: If only the training corpus had been a little bigger …

• Zipf’s Law (long tail phenomenon):
  – A small number of events occur with high frequency.
  – A large number of events occur with low frequency.
  – You can quickly collect statistics on the high frequency events.
  – You might have to wait an arbitrarily long time to get valid statistics on low frequency events.
Dealing with Sparsity

• Sparsity effects:
  – Bad estimates: we have no counts at all for the vast bulk of things we want to estimate!

• Solutions:
  1. **Discounting / Smoothing:**
     • Decrease the probability of previously seen events;
     • There is some probability mass left over for unseen events!
  2. **Neural Language Models:**
     • Learn word embeddings that capture semantic similarity;
     • Compute probabilities based on word meanings through word embeddings.
Discounting/Smoothing

• **Discounting** = decrease the probability of previously seen events:
  ⇒ there is some probability mass left over for unseen events.

• **Smoothing**: the resulting probability distribution is smoother.

• Smoothing methods:
  – Lidstone’s Law: Laplace’s Law, Jeffrey Perks Law
  – Good-Turing, Simple Good-Turing
  – Linear Interpolation (Jelinek-Mercer): Witten-Bell
  – Back-off Models (Katz)
  – Absolute Discounting: Kneser-Ney
Laplace Smoothing (Add-One)

• Simply add 1 to all counts:
  - Example: Laplace smoothing for unigrams:

\[ p_{\text{MLE}}(w_n) = \frac{C_n}{N} \quad \text{MLE estimate} \]

\[ p_{\text{Lap}}(w_n) = \frac{C_n + 1}{N + V} \quad \text{Laplace estimate} \]

\[ C^*_n = (C_n + 1) \frac{N}{N + V} \quad \text{Discounted counts} \]

\[ d_n = \frac{C^*_n}{C_n} \quad \text{Discount ratio} \]
Laplace Smoothing (Add-One)

- Laplace smoothing for bigrams:

\[ P_{MLE}(w_n | w_{n-1}) = \frac{C(w_{n-1}, w_n)}{C(w_{n-1})} \]  

MLE estimate

\[ P_{Lap}(w_n | w_{n-1}) = \frac{C(w_{n-1}, w_n) + 1}{C(w_{n-1}) + V} \]  

Discounted counts

\[ C^*(w_n, w_{n-1}) = \frac{[C(w_{n-1}, w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V} \]

\[ P_{Lap}(w_{n-1}, w_n) = \frac{C(w_{n-1}, w_n) + 1}{N + V^2} \]

Laplace estimate
Laplace Smoothing: Bigram Counts

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>i</strong></td>
<td>6</td>
<td>828</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>want</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>to</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>687</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>212</td>
</tr>
<tr>
<td>eat</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>17</td>
<td>3</td>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td>chinese</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>83</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>food</td>
<td>16</td>
<td>1</td>
<td>16</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>lunch</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>spend</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Laplace Smoothing: Bigram Probabilities

\[ p_{\text{Lap}}(w_n \mid w_{n-1}) = \frac{C(w_{n-1}, w_n) + 1}{C(w_{n-1}) + V} \]

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0.0015</td>
<td>0.21</td>
<td>0.00025</td>
<td>0.0025</td>
<td>0.00025</td>
<td>0.00025</td>
<td>0.00025</td>
<td>0.00075</td>
</tr>
<tr>
<td>want</td>
<td>0.0013</td>
<td>0.00042</td>
<td>0.26</td>
<td>0.00084</td>
<td>0.0029</td>
<td>0.0029</td>
<td>0.0025</td>
<td>0.00084</td>
</tr>
<tr>
<td>to</td>
<td>0.00078</td>
<td>0.00026</td>
<td>0.0013</td>
<td>0.18</td>
<td>0.00078</td>
<td>0.00026</td>
<td>0.0018</td>
<td>0.055</td>
</tr>
<tr>
<td>eat</td>
<td>0.00046</td>
<td>0.00046</td>
<td>0.0014</td>
<td>0.00046</td>
<td>0.0078</td>
<td>0.0014</td>
<td>0.02</td>
<td>0.00046</td>
</tr>
<tr>
<td>chinese</td>
<td>0.0012</td>
<td>0.00062</td>
<td>0.00062</td>
<td>0.00062</td>
<td>0.00062</td>
<td>0.052</td>
<td>0.0012</td>
<td>0.00062</td>
</tr>
<tr>
<td>food</td>
<td>0.0063</td>
<td>0.00039</td>
<td>0.0063</td>
<td>0.00039</td>
<td>0.00079</td>
<td>0.002</td>
<td>0.00039</td>
<td>0.00039</td>
</tr>
<tr>
<td>lunch</td>
<td>0.0017</td>
<td>0.00056</td>
<td>0.00056</td>
<td>0.00056</td>
<td>0.00056</td>
<td>0.0011</td>
<td>0.00056</td>
<td>0.00056</td>
</tr>
<tr>
<td>spend</td>
<td>0.0012</td>
<td>0.00058</td>
<td>0.0012</td>
<td>0.00058</td>
<td>0.00058</td>
<td>0.00058</td>
<td>0.00058</td>
<td>0.00058</td>
</tr>
</tbody>
</table>
Laplace Smoothing: Discounted Counts

\[ C^*(w_n, w_{n-1}) = \frac{[C(w_{n-1}, w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V} \]

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>3.8</td>
<td>527</td>
<td>0.64</td>
<td>6.4</td>
<td>0.64</td>
<td>0.64</td>
<td>0.64</td>
<td>1.9</td>
</tr>
<tr>
<td>want</td>
<td>1.2</td>
<td>0.39</td>
<td>238</td>
<td>0.78</td>
<td>2.7</td>
<td>2.7</td>
<td>2.3</td>
<td>0.78</td>
</tr>
<tr>
<td>to</td>
<td>1.9</td>
<td>0.63</td>
<td>3.1</td>
<td>430</td>
<td>1.9</td>
<td>0.63</td>
<td>4.4</td>
<td>133</td>
</tr>
<tr>
<td>eat</td>
<td>0.34</td>
<td>0.34</td>
<td>1</td>
<td>0.34</td>
<td>5.8</td>
<td>1</td>
<td>15</td>
<td>0.34</td>
</tr>
<tr>
<td>chinese</td>
<td>0.2</td>
<td>0.098</td>
<td>0.098</td>
<td>0.098</td>
<td>0.098</td>
<td>8.2</td>
<td>0.2</td>
<td>0.098</td>
</tr>
<tr>
<td>food</td>
<td>6.9</td>
<td>0.43</td>
<td>6.9</td>
<td>0.43</td>
<td>0.86</td>
<td>2.2</td>
<td>0.43</td>
<td>0.43</td>
</tr>
<tr>
<td>lunch</td>
<td>0.57</td>
<td>0.19</td>
<td>0.19</td>
<td>0.19</td>
<td>0.19</td>
<td>0.38</td>
<td>0.19</td>
<td>0.19</td>
</tr>
<tr>
<td>spend</td>
<td>0.32</td>
<td>0.16</td>
<td>0.32</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
</tr>
</tbody>
</table>
Laplace Smoothing

• Laplace gives too much probability mass to unseen events:
  – $C$(want to) went from 609 to 238!
  – $p$(to|want) from .66 to .26!

• Exercise (Church and Gale, 1991):
  – AP newswire, 44 million corpus, 400,653 vocabulary.
  – Training on half, i.e. 22 million words, $V = 273,266$ word types:
    • compute Laplace prob. for:
      – unseen bigrams ($N_0 = 74$ million).
      – happax legomena ($N_1 = 2$ million).
    • how much prob. mass is allocated to unseen bigrams?
Lidstone’s Law / Jeffrey Perks Law

- Pretend each n-gram occurs $\delta$ more time than it does:

$$p_{Lid}(w_n | w_{n-1}) = \frac{C(w_{n-1}, w_n) + \delta}{C(w_{n-1}) + \delta V} \quad p_{Lid}(w_{n-1}, w_n) = \frac{C(w_{n-1}, w_n) + \delta}{N + \delta \ast V^2}$$

- $\delta = 0 \Rightarrow$ MLE
- $\delta = 1 \Rightarrow$ Laplace’s Law
- $\delta = 1/2 \Rightarrow$ Jeffreys-Perks Law
  
  • Expected Likelihood Estimation (ELE)

- Objections:
  - need a good way to guess an appropriate value for $\delta$.
  - linear in the MLE counts, not always a good match for empirical distribution at low frequencies.
Good-Turing Discounting

- \( N_c = \text{frequency of frequency } c = \sum_{x: \text{count}(x) = c}^{1} \)  
  = the # of different n-grams seen \( c \) times.
- \( N = \text{total number of n-grams}. \)
- \( P_{GT}(n\text{-gram’s with frequency } c) = (c + 1) \frac{N_{c+1}}{N} \)
- Equivalent with:
  - \( P_{GT}(n\text{-gram with frequency } c) = \frac{c^*}{N} \)
  - \( c^* \) is new count for bin \( N_c \):
    \[ c^* = (c + 1) \frac{N_{c+1}}{N_c} \]
Good-Turing Discounting: Josh Goodman

Intuition

• Imagine you are fishing:
  – There are 8 species: carp, perch, whitefish, trout, salmon, eel, catfish, bass.
• You have caught:
  – 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish

\[ N_1 = 3, \quad N = 18 \implies N_1/N = 3/18 \]

• How likely is it that the next fish caught is from a new species (one not seen in our previous catch)?
  – \( N_1 = 3, \quad N = 18 \implies N_1/N = 3/18 \)
• Assuming so, how likely is it that next species is trout?
  – Must be less than 1/18 …
Good-Turing Discounting: Josh Goodman
Intuition

<table>
<thead>
<tr>
<th></th>
<th>unseen (bass or catfish)</th>
<th>trout</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>MLE $p$</td>
<td>$p = \frac{0}{18} = 0$</td>
<td>$\frac{1}{18}$</td>
</tr>
<tr>
<td>$c^*$</td>
<td></td>
<td>$c^*(\text{trout}) = 2 \times \frac{N_2}{N_1} = 2 \times \frac{1}{3} = .67$</td>
</tr>
<tr>
<td>GT $p^*_\text{GT}$</td>
<td>$p^*_\text{GT}(\text{unseen}) = \frac{N_1}{N} = \frac{3}{18} = .17$</td>
<td>$p^*_\text{GT}(\text{trout}) = \frac{.67}{18} = \frac{1}{27} = .037$</td>
</tr>
</tbody>
</table>

- We know $N_0 = 2$:
  - unknown species: catfish, bass.

- Therefore $P_{GT}(\text{catfish}) = \frac{1}{N_0} \times \frac{N_1}{N} = \frac{1}{2} \times \frac{3}{18} = 0.085$
Good-Turing Discount

- Bigram frequencies of frequencies and GT estimates:

<table>
<thead>
<tr>
<th>Series</th>
<th>AP Newswire</th>
<th>Berkeley Restaurant—</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$c$ (MLE)</td>
<td>$N_c$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>74,671,100,000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2,018,046</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>449,721</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>188,933</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>105,668</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>68,379</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>48,190</td>
</tr>
</tbody>
</table>

Lecture 01
Simple Good-Turing

• **Problem:**
  – \( P_{GT} \) (most frequent \( n \)-gram) = ?

• **Solutions:**
  1) Use GT reestimation only for frequencies \( c < k \) (\( k = 10 \) or 5).
  2) Fit a function \( S_c \) through the observed \((c, N_c)\), then use \( S_c \) instead:
     \[
     S_c = a \times c^b \iff \log S_c = a + b \log c
     \]
     \[
     c^* = (c + 1) \frac{S_{c+1}}{S_c}
     \]
     *Linear Regression*
  3) **Simple GT** (Gale and Sampson, 1995): Do both. Renormalize!

*Zipf’s Law*
Simple Good-Turing

- Renormalization (Katz, 1987):
  \[
  c^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}}, \quad \text{for } 1 \leq c \leq k.
  \]

- In practice, GT discounting used with:
  - Interpolation.
  - Backoff.
Interpolation and Backoff

- If we are estimating trigram \( p(z|x,y) \), but \( C(xyz) = 0 \):
  - use bigram \( p(z|y) \)
  - or even unigram \( p(z) \)

- How to combine this trigram, bigram, unigram info in a valid fashion?
  - **Interpolation**: mix all three.
  - **Backoff**: use trigram if you have it, otherwise bigram, otherwise unigram.
Interpolation

- Simple linear interpolation:

\[ P_{li}(w_n \mid w_{n-2} w_{n-1}) = \lambda_1 P(w_n \mid w_{n-2} w_{n-1}) + \lambda_2 P(w_n \mid w_{n-1}) + \lambda_3 P(w_n) \]

\[ \lambda_1 + \lambda_2 + \lambda_3 = 1 \]

- Conditional linear interpolation:

\[ P_{li}(w_n \mid w_{n-2} w_{n-1}) = \lambda_1 (w_{n-2} w_{n-1}) P(w_n \mid w_{n-2} w_{n-1}) + \lambda_2 (w_{n-2} w_{n-1}) P(w_n \mid w_{n-1}) + \lambda_3 (w_{n-2} w_{n-1}) P(w_n) \]
Setting Simple and Context-Dependent Weights $\lambda$

• Choose $\lambda$’s which maximize the probability of some held-out / development corpus:
  – Fix the $n$-gram probabilities.
  – Search for $\lambda$’s that maximize probability for held-out set.
    • Can use Expectation Maximization (EM) to do this search:
      – iterative learning algorithm [JM, Chapter 6].
      – converges to locally optimal $\lambda$’s.
    • Use Powell’s algorithm [Chen and Goodman, 1996].
Katz Backoff

Why use discounted probabilities $P^*$ instead of MLE? Why use $\alpha$?

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

$$P_{\text{katz}}(z|x,y) = \begin{cases} 
\alpha(x,y)P_{\text{katz}}(z|y), & \text{else if } C(x,y) > 0 \\
P^*(z), & \text{otherwise.}
\end{cases}$$

$$P_{\text{katz}}(z|y) = \begin{cases} 
P^*(z|y), & \text{if } C(y,z) > 0 \\
\alpha(y)P^*(z), & \text{otherwise.}
\end{cases}$$
Katz Backoff

- Need $\sum_z p_{Katz}(z \mid x, y) = 1$
- But MLE probabilities already sum up to 1: $\sum p(z \mid x, y) = 1$
  - If we used MLE probabilities but backed off to lower order model when MLE prob is zero, we would be adding extra probability mass, and thus total Katz probability would be greater than 1.
- Hence, need to use discounted $P^*$:

$$P^*(w_n \mid w_{n-N+1}^{n-1}) = \frac{c^* (w_n^{n-N+1})}{c(w_{n-N+1})}$$

- $\alpha$ is needed to make ensure the lower order probability mass sums up to the the amount saved by discounting.

Exact derivation of $\alpha$’s in Section 4.7.1.
Absolute Discounting

- Good-Turing estimates on AP corpus:
  - GT estimate $c^*$ ends up as MLE count - a constant $\delta$
    - $c^* = c - 0.75$

- **Absolute Discounting**: subtract the same discount $d$ from each count:
  
  $$P_{ad}(w_n | w_{n-1}) = \begin{cases} \frac{C(w_{n-1}w_n) - \delta}{C(w_{n-1})}, & \text{if } C(w_{n-1}w_n) > 0 \\ \alpha(w_{n-1})P_{MLE}(w_n), & \text{otherwise} \end{cases}$$

  - Compute backoff coefficients through normalization constraint.
  - Choose $\delta$ using held-out estimation.
  - In practice, keep distinct $\delta$’s for small counts.
Absolute Discounting: Kneser-Ney

• I can’t see without my reading _______
  – suppose \( C(\text{reading glasses}) = C(\text{reading Francisco}) = 0 \)
  • glasses? Francisco?
  • AD prefers Francisco: \( P(\text{Francisco}) > P(\text{glasses}) \).
  • But Francisco only appears in “San Francisco”, whereas “glasses” appears in many more different contexts ⇒ “glasses” more likely to appear in a new/unseen context!

• Instead of unigram MLE, use “continuation probability”:

\[
P_{\text{cont}}(w_n) = \frac{|\{w_{n-1} : C(w_{n-1}w_n) > 0\}|}{\sum_{w_n} |\{w_{n-1} : C(w_{n-1}w_n) > 0\}|}
\]
Kneser-Ney: Backoff vs. Interpolation

- **Backoff Kneser-Ney for bigrams grammars:**

\[
P_{KN}(w_n \mid w_{n-1}) = \begin{cases} 
\frac{C(w_{n-1}w_n) - \delta}{C(w_{n-1})}, & \text{if } C(w_{n-1}w_n) > 0 \\
\alpha(w_{n-1}) \frac{\{w_{n-1} : C(w_{n-1}w_n) > 0\}}{\sum_{w_n} \{w_{n-1} : C(w_{n-1}w_n) > 0\}}, & \text{otherwise}
\end{cases}
\]

- **Interpolated Kneser-Ney works better:**

\[
P_{KN}(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n) - \delta}{C(w_{n-1})} + \beta(w_{n-1}) \frac{\{w_{n-1} : C(w_{n-1}w_n) > 0\}}{\sum_{w_n} \{w_{n-1} : C(w_{n-1}w_n) > 0\}}
\]

Most commonly used smoothing method.
Using Long Distance Information

• Cache language models:
  - people tend to repeat words they have used before.
  - interpolate N-gram model with unigram model from preceding part of the corpus.

• Topic-based LMs.
  \[ p(w|h) = \sum_t p(w|t)p(t|h) \]

• Variable-length N-grams.
  - when a longer phrase is particularly frequent.

• LMs based on syntactic structures.
Class-based N-grams

- Assume (each) word belongs to a class / cluster:
  1. Classes manually assigned e.g. city, airline, month, weekday.

\[ P(w_n | w_{n-1}) \approx P(w_n | c_n)P(c_n | c_{n-1}) \]

\[ \Rightarrow P(\text{Shanghai} | \text{to}) = P(\text{Shanghai} | \text{city}) \times P(\text{city} | \text{to}) \]

- Useful for dealing with sparsity:
  - Usually mixed with regular word-based N-grams.
  - Better: use neural language models, deal with sparsity through learned word embeddings:
    - Brown clustering can be used to create word embeddings ...

Lecture 01
Sparse Representations of Words

- **Sparse (one-hot) vector representation:**
  - $V$ is the vocabulary
  - Each word $w$ is mapped to a unique $\text{id}(w)$ between 1 and $|V|$.
    - i.e. the position of the word in the vocabulary.
  - Represent a word $w$ using a "one-hot" vector $\mathbf{w}$ of length $|V|$:  
    - $\mathbf{w}[i] = 1$, if $i = \text{id}(w)$.
    - $\mathbf{w}[i] = 0$, otherwise

- **Example:**
  - Suppose $\text{id}(\text{frog}) = 2$ and $\text{id}(\text{turtle}) = 4$. Then:
    - $\mathbf{w}(\text{frog}) = [0, 1, 0, 0, 0, ..., 0]$
    - $\mathbf{w}(\text{turtle}) = [0, 0, 0, 1, 0, ..., 0]$
Neural Language Modeling with Sparse Word Representations

2. (N)LM with (Naive) Softmax Regression:

AI systems use deep ...... algorithms

\[ P(w \mid \text{deep, algorithms}) \quad \text{for each word } w \in V \]

- Need \(|W| = 2 \times |V| \times |V|\) parameters!
Sparse vs. Dense Representations of Words

- **Sparse representations:**
  - Each word $w$ is a sparse vector $w \in \{0,1\}^{|V|}$ or $\mathbb{R}^{|V|}$.
  - Using words as features leads to a large number of parameters.
    - $\text{sim}(\text{frog}, \text{turtle}) = 0$.

- **Dense representations:**
  - Each word $w$ is a dense vector $w \in \mathbb{R}^k$, where $k \ll |V|$.
  - Can use unsupervised learning:
    - Use Harris’ *Distributional Hypothesis* [Word, 1954]
      - Words that appear in the same contexts tend to have similar meanings.
    - $\text{sim}(\text{frog}, \text{turtle}) > \text{sim}(\text{frog}, \text{magpie}) > 0$
Neural Language Modeling with Dense Word Representations

- **Neural Language Modeling:**
  - Associate each word \( w \) with its distributed representation \( Uw \).
  - \( w \) is the *sparse (one-hot) representation* of word \( w \).
  - \( Uw \) is the *dense representation* of word \( w \):
    - i.e. word embedding.
    - i.e. distributed representation.
  - \( U \) is the projection or embedding matrix:
    - its columns are the word embeddings.
  - Use a neural network to simultaneously learn the word embeddings (\( U \)) and the softmax parameters (\( W \)).
Neural Language Modeling with Dense Word Representations

- Softmax on top of a hidden layer of size $h$ per word:

$$P(w | \text{deep, algorithms}) \iff \text{for each word } w \text{ in } V$$

$\#\text{params} = |V| \times h + 2h \times |V|$
A Neural Probabilistic Language Model

[Bengio et al., JMLR 2003]

- Fighting the Curse of Dimensionality with Distributed Representations:

1. Associate with each word in the vocabulary a distributed word feature vector (a real-valued vector in $\mathbb{R}^m$),

2. Express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence, and

3. Learn simultaneously the word feature vectors and the parameters of that probability function.

- Generalize i.e. transfer probability mass from seen sentence (first) to unseen sentences (the rest):

  The cat is walking in the bedroom
  A dog was running in a room
  The cat is running in a room
  A dog is walking in a bedroom
  The dog was walking in the room
i-th output = P(w_t = i | context)

softmax

most computation here

tanh

C(w_{t-n+1})

C(w_{t-2})

C(w_{t-1})

Table look-up in C

index for w_{t-n+1}

index for w_{t-2}

index for w_{t-1}

Matrix C

shared parameters across words
<table>
<thead>
<tr>
<th>Model</th>
<th>n</th>
<th>c</th>
<th>h</th>
<th>m</th>
<th>direct</th>
<th>mix</th>
<th>train.</th>
<th>valid.</th>
<th>test.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP1</td>
<td>5</td>
<td>50</td>
<td>60</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>182</td>
<td>284</td>
<td>268</td>
</tr>
<tr>
<td>MLP2</td>
<td>5</td>
<td>50</td>
<td>60</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>275</td>
<td>310</td>
<td>317</td>
</tr>
<tr>
<td>MLP3</td>
<td>5</td>
<td>0</td>
<td>60</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>201</td>
<td>327</td>
<td>310</td>
</tr>
<tr>
<td>MLP4</td>
<td>5</td>
<td>0</td>
<td>60</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>286</td>
<td>272</td>
<td>272</td>
</tr>
<tr>
<td>MLP5</td>
<td>5</td>
<td>50</td>
<td>30</td>
<td>yes</td>
<td>no</td>
<td></td>
<td>209</td>
<td>296</td>
<td>279</td>
</tr>
<tr>
<td>MLP6</td>
<td>5</td>
<td>50</td>
<td>30</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>273</td>
<td>259</td>
<td>259</td>
</tr>
<tr>
<td>MLP7</td>
<td>3</td>
<td>50</td>
<td>30</td>
<td>yes</td>
<td>no</td>
<td></td>
<td>210</td>
<td>309</td>
<td>293</td>
</tr>
<tr>
<td>MLP8</td>
<td>3</td>
<td>50</td>
<td>30</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>284</td>
<td>270</td>
<td>270</td>
</tr>
<tr>
<td>MLP9</td>
<td>5</td>
<td>100</td>
<td>30</td>
<td>no</td>
<td>no</td>
<td></td>
<td>175</td>
<td>280</td>
<td>276</td>
</tr>
<tr>
<td>MLP10</td>
<td>5</td>
<td>100</td>
<td>30</td>
<td>no</td>
<td>yes</td>
<td></td>
<td>265</td>
<td></td>
<td>252</td>
</tr>
<tr>
<td>Del. Int.</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>31</td>
<td>352</td>
<td>336</td>
</tr>
<tr>
<td>Kneser-Ney back-off</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>334</td>
<td>323</td>
<td></td>
</tr>
<tr>
<td>Kneser-Ney back-off</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>332</td>
<td>321</td>
<td></td>
</tr>
<tr>
<td>Kneser-Ney back-off</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>332</td>
<td>321</td>
<td></td>
</tr>
<tr>
<td>class-based back-off</td>
<td>3</td>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>348</td>
<td>334</td>
<td></td>
</tr>
<tr>
<td>class-based back-off</td>
<td>3</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>354</td>
<td>340</td>
<td></td>
</tr>
<tr>
<td>class-based back-off</td>
<td>3</td>
<td>500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>326</td>
<td>312</td>
<td></td>
</tr>
<tr>
<td>class-based back-off</td>
<td>3</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>335</td>
<td>319</td>
<td></td>
</tr>
<tr>
<td>class-based back-off</td>
<td>3</td>
<td>2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>343</td>
<td>326</td>
<td></td>
</tr>
<tr>
<td>class-based back-off</td>
<td>4</td>
<td>500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>327</td>
<td>312</td>
<td></td>
</tr>
<tr>
<td>class-based back-off</td>
<td>5</td>
<td>500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>327</td>
<td>312</td>
<td></td>
</tr>
</tbody>
</table>
Neural vs. N-gram Language Models

• **Neural Probabilistic LMs** are very slow at training and testing:
  – Hierarchical Probabilistic Neural Network Language Model [Morin & Bengio, AISTATS’05]:
    • replace unstructured vocabulary with a binary tree representing a clustering of words.
    • two orders of magnitude faster.
    • performed considerably worse than original NPLM, but still better than class-based trigrams.

• **Recurrent Neural Network LMs** [Mikolov, 2010]
  – No need to specify the context length.
  – SOA performance on Penn Treebank (PTB).
<table>
<thead>
<tr>
<th>Model</th>
<th>Perplexity</th>
<th>Entropy reduction over baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>individual +KN5</td>
<td>+KN5+cache</td>
</tr>
<tr>
<td>3-gram, Good-Turing smoothing (GT3)</td>
<td>165.2</td>
<td>-</td>
</tr>
<tr>
<td>5-gram, Good-Turing smoothing (GT5)</td>
<td>162.3</td>
<td>-</td>
</tr>
<tr>
<td>3-gram, Kneser-Ney smoothing (KN3)</td>
<td>148.3</td>
<td>-</td>
</tr>
<tr>
<td>5-gram, Kneser-Ney smoothing (KN5)</td>
<td>141.2</td>
<td>-</td>
</tr>
<tr>
<td>5-gram, Kneser-Ney smoothing + cache</td>
<td>125.7</td>
<td>-</td>
</tr>
<tr>
<td>PAQ8010t</td>
<td>131.1</td>
<td>-</td>
</tr>
<tr>
<td>Maximum entropy 5-gram model</td>
<td>142.1</td>
<td>0.4%</td>
</tr>
<tr>
<td>Random clusterings LM</td>
<td>170.1</td>
<td>2.3%</td>
</tr>
<tr>
<td>Random forest LM</td>
<td>131.9</td>
<td>1.5%</td>
</tr>
<tr>
<td>Structured LM</td>
<td>146.1</td>
<td>2.4%</td>
</tr>
<tr>
<td>Within and across sentence boundary LM</td>
<td>116.6</td>
<td>5.0%</td>
</tr>
<tr>
<td>Log-bilinear LM</td>
<td>144.5</td>
<td>4.1%</td>
</tr>
<tr>
<td>Feedforward neural network LM [50]</td>
<td>140.2</td>
<td>3.8%</td>
</tr>
<tr>
<td>Feedforward neural network LM [40]</td>
<td>141.8</td>
<td>4.2%</td>
</tr>
<tr>
<td>Syntactical neural network LM</td>
<td>131.3</td>
<td>5.0%</td>
</tr>
<tr>
<td>Recurrent neural network LM</td>
<td>124.7</td>
<td>5.8%</td>
</tr>
<tr>
<td>Dynamically evaluated RNNLM</td>
<td>123.2</td>
<td>6.4%</td>
</tr>
<tr>
<td>Combination of static RNNLMs</td>
<td>102.1</td>
<td>7.9%</td>
</tr>
<tr>
<td>Combination of dynamic RNNLMs</td>
<td>101.0</td>
<td>8.5%</td>
</tr>
</tbody>
</table>
LM Toolkits

• SRLIM toolkit [Stolke, 2002].

• Cambridge-CMU toolkit [Clarkson and Rosenfeld, 1997].

• NLTK [Bird et al., 2009].

• RNNLM Toolkit [Mikolov, 2010-12].