Language Modeling (and the Noisy Channel)
The Noisy Channel

- Prototypical case:
  - Input: 0,1,1,1,0,1,0,1,...
  - Output (noisy): 0,1,1,0,0,1,1,0,...

- Model: probability of error (noise):
  - Example: $p(0|1) = .3 \quad p(1|1) = .7 \quad p(1|0) = .4 \quad p(0|0) = .6$

- The Task:
  - known: the noisy output; want to know: the input (decoding)
Noisy Channel Applications

- OCR
  - text → print (adds noise), scan/camera → image
- Handwriting recognition
  - text → neurons, muscles (“noise”), scan/digitize → image
- Speech recognition (dictation, commands, etc.)
  - text → conversion to acoustic signal (“noise”) → acoustic waves
- Machine Translation
  - text in target language → translation (“noise”) → source language
- Also: Part of Speech Tagging
  - sequence of tags → selection of word forms (“noise”) → text
Noisy Channel: The Golden Rule of ...

Recall:

\[ p(A|B) = \frac{p(B|A)p(A)}{p(B)} \]  (Bayes formula)

\[ A_{\text{best}} = \arg\max_A p(B|A)p(A) \]  (The Golden Rule)

- \( p(B|A) \): the acoustic/image/translation/lexical model
  - application-specific name
  - will explore later

- \( p(A) \): *the language model*
Probabilistic Language Models

• Today’s goal: assign a probability to a sentence
  • Machine Translation:
    • \( P(\text{high winds tonite}) > P(\text{large winds tonite}) \)
  • Spell Correction
    • The office is about fifteen minuets from my house
      ▶ \( P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from}) \)
  • Speech Recognition
    • \( P(\text{I saw a van}) \gg P(\text{eyes awe of an}) \)
  • + Summarization, question-answering, etc., etc.!!
Probabilistic Language Modeling

- **Goal:** compute the probability of a sentence or sequence of words:
  - \( P(W) = P(w_1, w_2, w_3, w_4, w_5...w_n) \)
- **Related task:** probability of an upcoming word:
  - \( P(w_5 | w_1, w_2, w_3, w_4) \)
- **A model that computes either of these:**
  - \( P(W) \) or \( P(w_n | w_1, w_2...w_{n-1}) \) is called a language model.
- **Better:** the grammar
  - But language model or LM is standard
The Perfect Language Model

- **Sequence of word forms** [forget about tagging for the moment]
- Notation: \( A \sim W = (w_1, w_2, w_3, \ldots, w_d) \)
- The big (modeling) question:
  \[
p(W) = ?
  \]
- Well, we know (Bayes/chain rule \(\rightarrow\)):
  \[
p(W) = p(w_1, w_2, w_3, \ldots, w_d) = \\
p(w_1) \times p(w_2 | w_1) \times p(w_3 | w_1, w_2) \times \ldots \times p(w_d | w_1, w_2, \ldots, w_{d-1})
  \]
- Not practical (even short \( W \rightarrow \) too many parameters)
Markov Chain

- Unlimited memory:
  - for $w_i$, we know all its predecessors $w_1, w_2, w_3, \ldots, w_{i-1}$

- Limited memory:
  - we disregard “too old” predecessors
  - remember only $k$ previous words: $w_{i-k}, w_{i-k+1}, \ldots, w_{i-1}$
  - called “$k$th order Markov approximation”

- + stationary character (no change over time):

\[ p(W) \approx \prod_{i=1}^{d} p(w_i | w_{i-k}, w_{i-k+1}, \ldots, w_{i-1}), \quad d = |W| \]
n-gram Language Models

- \((n-1)^{th}\) order Markov approximation \(\rightarrow\) n-gram LM:

\[ p(W) = \prod_{i=1}^{d} p(w_i|w_{i-n+1},w_{i-n+2},\ldots,w_{i-1}) \]

- In particular (assume vocabulary \(|V| = 60k\)):
  - 0-gram LM: uniform model, \(p(w) = 1/|V|\), 1 parameter
  - 1-gram LM: unigram model, \(p(w)\), \(6 \times 10^4\) parameters
  - 2-gram LM: bigram model, \(p(w_i|w_{i-1})\) \(3.6 \times 10^9\) parameters
  - 3-gram LM: trigram model, \(p(w_i|w_{i-2},w_{i-1})\) \(2.16 \times 10^{14}\) parameters
LM: Observations

- How large $n$?
  - nothing is enough (theoretically)
  - but anyway: as much as possible ($\rightarrow$ close to “perfect” model)
  - empirically: 3
    - parameter estimation? (reliability, data availability, storage space, …)
  - 4 is too much: $|V|=60k \rightarrow 1.296 \times 10^{19}$ parameters
  - but: 6-7 would be (almost) ideal (having enough data): in fact, one can recover original from 7-grams!

- Reliability $\sim (1 / \text{Detail})$ ($\rightarrow$ need compromise) (detail=many gram)
- For now, keep word forms (no “linguistic” processing)
Parameter Estimation

- **Parameter**: numerical value needed to compute $p(w|h)$
- **From data (how else?)**
- **Data preparation:**
  - get rid of formatting etc. (“text cleaning”)
  - define words (separate but include punctuation, call it “word”)
  - define sentence boundaries (insert “words” `<s>` and `</s>`)  
  - letter case: keep, discard, or be smart:
    - name recognition
    - number type identification
    [these are huge problems per se!]
  - numbers: keep, replace by `<num>`, or be smart (form ~ pronunciation)
Maximum Likelihood Estimate

- **MLE: Relative Frequency...**
  - ...best predicts the data at hand (the “training data”)

- **Trigrams from Training Data T:**
  - count sequences of three words in T: \( c_3(w_{i-2}, w_{i-1}, w_i) \)
    - [NB: notation: just saying that the three words follow each other]
  - count sequences of two words in T: \( c_2(w_{i-1}, w_i) \):
    - either use \( c_2(y, z) = \sum_w c_3(y, z, w) \)
    - or count differently at the beginning (& end) of data!

\[
p(w_i | w_{i-2}, w_{i-1}) = \text{est.} \frac{c_3(w_{i-2}, w_{i-1}, w_i)}{c_2(w_{i-2}, w_{i-1})}
\]
Character Language Model

- Use individual characters instead of words:

\[ p(W) = \prod_{i=1}^{df} p(c_i | c_{i-n+1}, c_{i-n+2}, \ldots, c_{i-1}) \]

- Same formulas etc.

- Might consider 4-grams, 5-grams or even more

- Good only for language comparison

- Transform cross-entropy between letter- and word-based models:

\[ H_S(p_c) = H_S(p_w) / \text{avg. # of characters/word in } S \]
LM: an Example

- Training data:
  
  \(<s> <s> He can buy the can of soda.\)

- Unigram: \(p_1(He) = p_1(buy) = p_1(the) = p_1(of) = p_1(soda) = p_1(.) = .125\)
  \(p_1(can) = .25\)

- Bigram: \(p_2(He | <s>) = 1, p_2(can | He) = 1, p_2(buy | can) = .5,\)
  \(p_2(of | can) = .5, p_2(the | buy) = 1,\ldots\)

- Trigram: \(p_3(He | <s>, <s>) = 1, p_3(can | <s>, He) = 1,\)
  \(p_3(buy | He, can) = 1, p_3(of | the, can) = 1, \ldots, p_3(.) | of, soda) = 1.\)

- (normalized for all n-grams) Entropy: \(H(p_1) = 2.75, H(p_2) = .25, H(p_3) = 0\) ← Great?!
Language Modeling Toolkits

- SRILM
Google N-Gram Release, August 2006

All Our N-gram are Belong to You

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects,

... that’s why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.
Evaluation: How good is our model?

- Does our language model prefer good sentences to bad ones?
  - Assign higher probability to “real” or “frequently observed” sentences
    - Than “ungrammatical” or “rarely observed” sentences?
- We train parameters of our model on a training set.
- We test the model’s performance on data we haven’t seen.
  - A test set is an unseen dataset that is different from our training set, totally unused.
  - An evaluation metric tells us how well our model does on the test set.
Extrinsic evaluation of N-gram models

- Best evaluation for comparing models A and B
  - Put each model in a task
    - spelling corrector, speech recognizer, MT system
  - Run the task, get an accuracy for A and for B
    - How many misspelled words corrected properly
    - How many words translated correctly
  - Compare accuracy for A and B
Difficulty of extrinsic (in-vivo) evaluation of N-gram models

- Extrinsic evaluation
  - Time-consuming; can take days or weeks
- So
  - Sometimes use intrinsic evaluation: perplexity
  - Bad approximation
    - unless the test data looks just like the training data
    - So generally only useful in pilot experiments
- But is helpful to think about.
Intuition of Perplexity

The Shannon Game:
- How well can we predict the next word?
  - I always order pizza with cheese and ____
  - The 33rd President of the US was ____
  - I saw a ____

- Unigrams are terrible at this game. (Why?)

- A better model of a text
  - is one which assigns a higher probability to the word that actually occurs

mushrooms 0.1
pepperoni 0.1
anchovies 0.01
...
fried rice 0.0001
...
and 1e-100
Perplexity

The best language model is one that best predicts an unseen test set:

- Gives the highest $P(sentence)$
- Perplexity is the probability of the test set, normalized by the number of words:

$$PP(W) = \sqrt[\frac{N}{P(w_1 w_2 \ldots w_N)}}$$

- Chain rule:

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

- For bigrams:

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

Minimizing perplexity is the same as maximizing probability
Perplexity as branching factor

- Let’s suppose a sentence consisting of random digits
- What is the perplexity of this sentence according to a model that assign \( P=1/10 \) to each digit?

\[
PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}
\]

\[
= \left( \frac{1}{10} \right)^{-\frac{1}{N}}
\]

\[
= 1^{-1}
\]

\[
= \frac{1}{10}
\]

\[
= 10
\]
Lower perplexity = better model

- Training 38 million words, test 1.5 million words, WSJ

<table>
<thead>
<tr>
<th>N-gram Order</th>
<th>Unigram</th>
<th>Bigram</th>
<th>Trigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perplexity</td>
<td>962</td>
<td>170</td>
<td>109</td>
</tr>
</tbody>
</table>
The wall street journal

**Unigram**

Months the my and issue of year foreign new exchange’s september were recession ex-
change 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
LM: an Example

- Training data:

  \(<s> <s> \text{He can buy the can of soda.}\)

- Unigram: \(p_1(\text{He}) = p_1(\text{buy}) = p_1(\text{the}) = p_1(\text{of}) = p_1(\text{soda}) = p_1(\text{.}) = 0.125\)
  \(p_1(\text{can}) = 0.25\)

- Bigram: \(p_2(\text{He} | <s>) = 1, p_2(\text{can} | \text{He}) = 1, p_2(\text{buy} | \text{can}) = 0.5,\)
  \(p_2(\text{of} | \text{can}) = 0.5, p_2(\text{the} | \text{buy}) = 1,...\)

- Trigram: \(p_3(\text{He} | <s>, <s>) = 1, p_3(\text{can} | <s>, \text{He}) = 1,\)
  \(p_3(\text{buy} | \text{He}, \text{can}) = 1, p_3(\text{of} | \text{the}, \text{can}) = 1, ..., p_3(\text{.} | \text{of}, \text{soda}) = 1.\)

- (normalized for all n-grams) Entropy: \(H(p_1) = 2.75, H(p_2) = 0.25, H(p_3) = 0 \quad \text{Great?!}\)
LM: an Example (The Problem)

- Cross-entropy:
- \( S = \langle s \rangle \langle s \rangle \text{ It was the greatest buy of all. (test data)} \)
- Even \( H_S(p_1) \) fails (= \( H_S(p_2) = H_S(p_3) = \infty \)), because:
  - all unigrams but \( p_1(\text{the}), p_1(\text{buy}), p_1(\text{of}) \) and \( p_1(.) \) are 0.
  - all bigram probabilities are 0.
  - all trigram probabilities are 0.
- We want: to make all probabilities non-zero. \( \Rightarrow \) data sparseness handling
Why do we need Nonzero Probs?

- To avoid infinite Cross Entropy:
  - happens when an event is found in test data which has not been seen in training data
    \[ H(p) = \infty: \text{prevents comparing data with } \geq 0 \text{ "errors"} \]

- To make the system more robust
  - low count estimates:
    - they typically happen for “detailed” but relatively rare appearances
  - high count estimates: reliable but less “detailed”
Eliminating the Zero Probabilities: Smoothing

- Get new \( p'(w) \) (same \( \Omega \)): almost \( p(w) \) but no zeros
- Discount \( w \) for (some) \( p(w) > 0 \): new \( p'(w) < p(w) \)

\[
\sum_{w \in \text{discounted}} (p(w) - p'(w)) = D
\]

- Distribute \( D \) to all \( w \); \( p(w) = 0 \): new \( p'(w) > p(w) \)
  - possibly also to other \( w \) with low \( p(w) \)
- For some \( w \) (possibly): \( p'(w) = p(w) \)
- Make sure \( \sum_{w \in \Omega} p'(w) = 1 \)
- There are many ways of **smoothing**
Smoothing by Adding 1 (Laplace)

Simplest but not really usable:

- Predicting words \(w\) from a vocabulary \(V\), training data \(T\):
  \[ p'(w|h) = \frac{(c(h,w) + 1)}{(c(h) + |V|)} \]
  for non-conditional distributions: \[ p'(w) = \frac{(c(w) + 1)}{|T| + |V|} \]
- Problem if \(|V| > c(h)\) (as is often the case; even >> c(h)!) 

Example:  Training data:  \(<s>\) what is it what is small ?  \(|T| = 8\)

- \(V = \{ \text{what, is, it, small, ?, <s>, flying, birds, are, a, bird, . } \}, |V| = 12\)
- \(p(\text{it}) = .125, p(\text{what}) = .25, p(.) = 0\)  \(p(\text{what is it?}) = .25^2 \times .125^2 \cong .001\)
  \(p(\text{it is flying.}) = .125 \times .25 \times 0^2 = 0\)
- \(p'(\text{it}) = .1, p'(\text{what}) = .15, p'(.) = .05\)  \(p'(\text{what is it?}) = .15^2 \times .1^2 \cong .0002\)
  \(p'(\text{it is flying.}) = .1 \times .15 \times .05^2 \cong .00004\)

(assume word independence!)
Adding \textit{less than} 1

Equally simple:

- Predicting words $w$ from a vocabulary $V$, training data $T$:
  \[ p'(w|h) = \frac{(c(h,w) + \lambda)}{(c(h) + \lambda |V|)}, \lambda < 1 \]
  - for non-conditional distributions: \[ p'(w) = \frac{(c(w) + \lambda)}{|T| + \lambda |V|} \]

Example: Training data: \(<s>\) what is it what is small ? \(|T| = 8\)

- $V = \{ \text{what, is, it, small, ?, <s>, flying, birds, are, a, bird, . } \}$, \(|V| = 12\)
- $p(\text{it}) = .125$, $p(\text{what}) = .25$, $p(.) = 0$  \( p(\text{what is it?}) = .25^2 \times .125^2 \approx .001 \)
  \[ p(\text{it is flying.}) = .125 \times .25 \times .01^2 = 0 \]

- Use $\lambda = .1$:
  - $p'(\text{it}) \approx .12$, $p'(\text{what}) \approx .23$, $p'(.) \approx .01$  \( p'(\text{what is it?}) = .23^2 \times .12^2 \approx .0007 \)
  \[ p'(\text{it is flying.}) = .12 \times .23 \times .01^2 \approx .000003 \]
Language Modeling

Advanced: Good Turing Smoothing
Reminder: Add-1 (Laplace) Smoothing

\[ P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V} \]
More general formulations: Add-k

\[ P_{Add-k}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_{i-1}) + kV} \]

\[ P_{Add-k}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + m \left( \frac{1}{V} \right)}{c(w_{i-1}) + m} \]
Unigram prior smoothing

\[
P_{Add-k}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + m\left(\frac{1}{V}\right)}{c(w_{i-1}) + m}
\]

\[
P_{UnigramPrior}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + mP(w_i)}{c(w_{i-1}) + m}
\]
Advanced smoothing algorithms

- Intuition used by many smoothing algorithms
  - Good-Turing
  - Kneser-Ney
  - Witten-Bell

- Use the count of things we’ve seen once
  - to help estimate the count of things we’ve never seen
Notation: $N_c = \text{Frequency of frequency } c$

- $N_c =$ the count of things we’ve seen $c$ times
- Sam I am I am Sam I do not eat

\begin{align*}
I & \quad 3 \\
\text{sam} & \quad 2 \\
\text{am} & \quad 2 \\
\text{do} & \quad 1 \\
\text{not} & \quad 1 \\
\text{eat} & \quad 1 \\
\end{align*}

\begin{align*}
N_1 & = 3 \\
N_2 & = 2 \\
N_3 & = 1
\end{align*}
You are fishing (a scenario from Josh Goodman), and caught:
- 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish

How likely is it that next species is trout?
- 1/18

How likely is it that next species is new (i.e. catfish or bass)
- Let’s use our estimate of things-we-saw-once to estimate the new things.
- 3/18 (because $N_1 = 3$)

Assuming so, how likely is it that next species is trout?
- Must be less than 1/18 -> discounted by 3/18!!
- How to estimate?
Good Turing calculations

\[ P_{GT}^*(\text{things with zero frequency}) = \frac{N_1}{N} \]

- Unseen (bass or catfish)
  - \( c = 0: \)  
    - MLE \( p = 0/18 = 0 \)
  - \( P_{GT}^*(\text{unseen}) = N_1/N = 3/18 \)

- Seen once (trout)
  - \( c = 1 \)
  - MLE \( p = 1/18 \)
  - \( C^*(\text{trout}) = 2 * N_2/N_1 \)
    - \( = 2 * 1/3 \)
    - \( = 2/3 \)
  - \( P_{GT}^*(\text{trout}) = 2/3 / 18 = 1/27 \) 
  
For only low count
Language Modeling

Advanced:
Kneser-Ney Smoothing
Resulting Good-Turing numbers

- Numbers from Church and Gale (1991)
- 22 million words of AP Newswire

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

- It sure looks like \( C^* = (c - .75) \)

<table>
<thead>
<tr>
<th>Count c</th>
<th>Good Turing c*</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.0000270</td>
</tr>
<tr>
<td>1</td>
<td>0.446</td>
</tr>
<tr>
<td>2</td>
<td>1.26</td>
</tr>
<tr>
<td>3</td>
<td>2.24</td>
</tr>
<tr>
<td>4</td>
<td>3.24</td>
</tr>
<tr>
<td>5</td>
<td>4.22</td>
</tr>
<tr>
<td>6</td>
<td>5.19</td>
</tr>
<tr>
<td>7</td>
<td>6.21</td>
</tr>
<tr>
<td>8</td>
<td>7.24</td>
</tr>
<tr>
<td>9</td>
<td>8.25</td>
</tr>
</tbody>
</table>
Absolute Discounting

Interpolation

- Save ourselves some time and just subtract 0.75 (or some d)!

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1}) P(w)$$

- (Maybe keeping a couple extra values of d for counts 1 and 2)
- But should we really just use the regular unigram P(w)?
Kneser-Ney Smoothing

- Alternative metaphor: The number of # of word types seen to precede w

\[ \left| \left\{ w_{i-1} : c(w_{i-1}, w) > 0 \right\} \right| \]

- Normalized by the # of words preceding all words:

\[ P_{\text{CONTINUATION}}(w) = \frac{\left| \left\{ w_{i-1} : c(w_{i-1}, w) > 0 \right\} \right|}{\sum_{w'} \left| \left\{ w'_{i-1} : c(w'_{i-1}, w') > 0 \right\} \right|} \]

- A frequent word (Francisco) occurring in only one context (San) will have a low continuation probability
Kneser-Ney Smoothing

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

\( \lambda \) is a normalizing constant; the probability mass we’ve discounted

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

The number of word types that can follow \( w_{i-1} \)

= # of word types we discounted

= # of times we applied normalized discount
Kneser-Ney Smoothing: Recursive formulation

\[ P_{KN}(w_i \mid w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i-1}) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1})P_{KN}(w_i \mid w_{i-n+2}^{i-1}) \]

\[ c_{KN}(\bullet) = \begin{cases} 
\text{count}(\bullet) & \text{for the highest order} \\
\text{continuation count}(\bullet) & \text{for lower order}
\end{cases} \]

Continuation count = Number of unique single word contexts for \( \bullet \)
Backoff and Interpolation

- Sometimes it helps to use less context
  - Condition on less context for contexts you haven’t learned much about

- Backoff:
  - use trigram if you have good evidence,
  - otherwise bigram, otherwise unigram

- Interpolation:
  - mix unigram, bigram, trigram

- Interpolation works better
Smoothing by Combination: Linear Interpolation

Combine what?
- distributions of various level of detail vs. reliability

n-gram models:
- use (n-1)gram, (n-2)gram, ..., uniform

Simplest possible combination:
- sum of probabilities, normalize:
  - \( p(0|0) = .8, \quad p(1|0) = .2, \quad p(0|1) = 1, \quad p(1|1) = 0, \quad p(0) = .4, \quad p(1) = .6: \)
  - \( p'(0|0) = .6, \quad p'(1|0) = .4, \quad p'(0|1) = .7, \quad p'(1|1) = .3 \)
  - \( p'(0|0) = 0.5p(0|0) + 0.5p(0) \)
Typical n-gram LM Smoothing

- Weight in less detailed distributions using $\lambda=(\lambda_0, \lambda_1, \lambda_2, \lambda_3)$:
  
  $p'_{\lambda}(w_i | w_{i-2}, w_{i-1}) = \lambda_3 p_3(w_i | w_{i-2}, w_{i-1}) + $ 
  
  $\lambda_2 p_2(w_i | w_{i-1}) + \lambda_1 p_1(w_i) + \lambda_0 / |V|$

- Normalize:
  
  $\lambda_i > 0, \sum_{i=0..n} \lambda_i = 1$ is sufficient ($\lambda_0 = 1 - \sum_{i=1..n} \lambda_i$) (n=3)

- Estimation using MLE:
  
  - fix the $p_3$, $p_2$, $p_1$ and $|V|$ parameters as estimated from the training data
  - then find such $\{\lambda_i\}$ which minimizes the cross entropy (maximizes probability of data):
    
    $-(1/|D|)\sum_{i=1..|D|} \log_2(p'_{\lambda}(w_i | h_i))$

    $\lambda_i$ Hyper-parameter (EM algorithm)
Held-out Data

- **What data to use? (to estimate \( \lambda \))**
  - (bad) try the training data \( T \): but we will always get \( \lambda_3 = 1 \)
    - why? (let \( p_{iT} \) be an i-gram distribution estimated using relative freq. from \( T \))
    - minimizing \( H_T(p'_\lambda) \) over a vector \( \lambda \), \( p'_\lambda = \lambda_3 p_{3T} + \lambda_2 p_{2T} + \lambda_1 p_{1T} + \lambda_0 / |V| \)
      - remember: \( H_T(p'_\lambda) = H(p_{3T}) + D(p_{3T} || p'_\lambda) \); (\( p_{3T} \) fixed \( \rightarrow \) \( H(p_{3T}) \) fixed, best)
      - which \( p'_\lambda \) minimizes \( H_T(p'_\lambda) \)? Obviously, \( p'_\lambda \) for which \( D(p_{3T} || p'_\lambda) = 0 \)
      - ...and that's \( p_{3T} \) (because \( D(p || p) = 0 \), as we know).
      - ...and certainly \( p'_\lambda = p_{3T} \) if \( \lambda_3 = 1 \) (maybe in some other cases, too).
    - \( (p'_\lambda = 1 \times p_{3T} + 0 \times p_{2T} + 0 \times p_{1T} + 0 / |V|) \)
  - thus: do not use the training data for estimation of \( \lambda \)!
    - must hold out part of the training data (**heldout** data, \( H \)):
    - ...call the remaining data the (true/raw) **training** data, \( T \)
    - the **test** data \( S \) (e.g., for comparison purposes): still different data!
Mutual Information and Word Classes (class n-gram)
The Problem

- Not enough data
  - Language Modeling: we do not see “correct” n-grams
    - solution so far: smoothing
  - suppose we see:
    - short homework, short assignment, simple homework
  - but not:
    - simple assignment
  - What happens to our (bigram) LM?
    - $p(\text{homework} \mid \text{simple}) = \text{high probability}$
    - $p(\text{assignment} \mid \text{simple}) = \text{low probability (smoothed with } p(\text{assignment}))$
- They should be much closer!
Word Classes

- Observation: similar words behave in a similar way
  - trigram LM:
    - in the … (all nouns/adj);
    - catch a … (all things which can be caught, incl. their accompanying adjectives);
  - trigram LM, conditioning:
    - a … homework (any attribute of homework: short, simple, late, difficult),
    - … the woods (any verb that has the woods as an object: walk, cut, save)
  - trigram LM: both:
    - a (short, long, difficult, …) (homework, assignment, task, job, …)
Solution

- Use the Word Classes as the “reliability” measure
- Example: we see
  - short homework, short assignment, simple homework
  - but not:
    - simple assignment
- Cluster into classes:
  - (short, simple) (homework, assignment)
    - covers “simple assignment”, too
- Gaining: realistic estimates for unseen n-grams
- Loosing: accuracy (level of detail) within classes
The New Model

- Rewrite the n-gram LM using classes:
  - Was: \([k = 1\ldots n]\)
    - \(p_k(w_i|h_i) = \frac{c(h_i,w_i)}{c(h_i)}\) \[\text{history: (k-1) words}\]
  - Introduce classes:
    \[
p_k(w_i|h_i) = p(w_i|c_i) p_k(c_i|h_i) \]
    - history: \text{classes}, too: [for trigram: \(h_i = c_{i-2},c_{i-1}\), bigram: \(h_i = c_{i-1}\)]
  - Smoothing as usual
    - over \(p_k(w_i|h_i)\), where each is defined as above (except uniform which stays at \(1/|V|\))
Training Data

- Suppose we already have a mapping:
  - \( r: V \rightarrow C \) assigning each word its class \( (c_i = r(w_i)) \)

- Expand the training data:
  - \( T = (w_1, w_2, ..., w_{|T|}) \) into
  - \( T_C = (<w_1, r(w_1)>, <w_2, r(w_2)>, ..., <w_{|T|}, r(w_{|T|})>) \)

- Effectively, we have two streams of data:
  - word stream: \( w_1, w_2, ..., w_{|T|} \)
  - class stream: \( c_1, c_2, ..., c_{|T|} \) (def. as \( c_i = r(w_i) \))

- Expand Heldout, Test data too
Training the New Model

- As expected, using ML estimates:
  - \[ p(w_i|c_i) = p(w_i|r(w_i)) = \frac{c(w_i)}{c(r(w_i))} = \frac{c(w_i)}{c(c_i)} \]
  - !!! \( c(w_i,c_i) = c(w_i) \) [since \( c_i \) determined by \( w_i \)]

- \( p_k(c_i|h_i) \):
  - \( p_3(c_i|h_i) = p_3(c_i|c_{i-2},c_{i-1}) = \frac{c(c_{i-2},c_{i-1},c_i)}{c(c_{i-2},c_{i-1})} \)
  - \( p_2(c_i|h_i) = p_2(c_i|c_{i-1}) = \frac{c(c_{i-1},c_i)}{c(c_{i-1})} \)
  - \( p_1(c_i|h_i) = p_1(c_i) = \frac{c(c_i)}{|T|} \)

- Then smooth as usual
  - not the \( p(w_i|c_i) \) nor \( p_k(c_i|h_i) \) individually, but the \( p_k(w_i|h_i) \)
Word Classes in Applications

- Word Sense Disambiguation: context not seen [enough(-times)]
- Parsing: verb-subject, verb-object relations
- Speech recognition (acoustic model): need more instances of [rare(r)] sequences of phonemes
- Machine Translation: translation equivalent selection [for rare(r) words]
Spelling Correction and the Noisy Channel

The Spelling Correction Task
Spelling Tasks

- Spelling Error Detection
- Spelling Error Correction:
  - Autocorrect
    - hte → the
  - Suggest a correction
  - Suggestion lists
Types of spelling errors

- Non-word Errors
  - graffe → giraffe
- Real-word Errors
  - Typographical errors
    - three → there
  - Cognitive Errors (homophones)
    - piece → peace,
    - too → two
Non-word spelling errors

- Non-word spelling error detection:
  - Any word not in a *dictionary* is an error
  - The larger the dictionary the better

- Non-word spelling error correction:
  - Generate *candidates*: real words that are similar to error
  - Choose the one which is best:
    - Shortest weighted edit distance
    - Highest noisy channel probability
Real word spelling errors

- For each word $w$, generate candidate set:
  - Find candidate words with similar *pronunciations*
  - Find candidate words with similar *spelling*
  - Include $w$ in candidate set
- Choose best candidate
  - Noisy Channel
  - Classifier
Spelling Correction and the Noisy Channel

The Noisy Channel Model of Spelling
Noisy Channel

- We see an observation $x$ of a misspelled word
- Find the correct word $w$

\[
\hat{w} = \arg\max_{w \in V} P(w | x)
\]

\[
= \arg\max_{w \in V} \frac{P(x | w)P(w)}{P(x)}
\]

\[
= \arg\max_{w \in V} P(x | w)P(w)
\]
Computing error probability: confusion matrix

\[ \text{del}[x,y]: \quad \text{count}(xy \text{ typed as } x) \]
\[ \text{ins}[x,y]: \quad \text{count}(x \text{ typed as } xy) \]
\[ \text{sub}[x,y]: \quad \text{count}(x \text{ typed as } y) \]
\[ \text{trans}[x,y]: \quad \text{count}(xy \text{ typed as } yx) \]

Insertion and deletion conditioned on previous character
Channel model

\[ P(x|w) = \begin{cases} 
\frac{\text{del}[w_{i-1},w_i]}{\text{count}[w_{i-1}w_i]}, & \text{if deletion} \\
\frac{\text{ins}[w_{i-1},x_i]}{\text{count}[w_{i-1}]}, & \text{if insertion} \\
\frac{\text{sub}[x_i,w_i]}{\text{count}[w_i]}, & \text{if substitution} \\
\frac{\text{trans}[w_i,w_{i+1}]}{\text{count}[w_iw_{i+1}]}, & \text{if transposition}
\end{cases} \]
<table>
<thead>
<tr>
<th>Candidate Correction</th>
<th>Correct Letter</th>
<th>Error Letter</th>
<th>x(\mid)w</th>
<th>(P(x\mid\text{word}))</th>
<th>(P(\text{word}))</th>
<th>(10^9 P(x\mid w) P(w))</th>
</tr>
</thead>
<tbody>
<tr>
<td>actress</td>
<td>t</td>
<td>-</td>
<td>c\mid ct</td>
<td>.000117</td>
<td>.0000231</td>
<td>2.7</td>
</tr>
<tr>
<td>cress</td>
<td>-</td>
<td>a</td>
<td>a\mid #</td>
<td>.00000144</td>
<td>.000000544</td>
<td>.00078</td>
</tr>
<tr>
<td>caress</td>
<td>ca</td>
<td>ac</td>
<td>ac\mid ca</td>
<td>.00000164</td>
<td>.00000170</td>
<td>.0028</td>
</tr>
<tr>
<td>access</td>
<td>c</td>
<td>r</td>
<td>r\mid c</td>
<td>.000000209</td>
<td>.0000916</td>
<td>.019</td>
</tr>
<tr>
<td>across</td>
<td>o</td>
<td>e</td>
<td>e\mid o</td>
<td>.0000093</td>
<td>.000299</td>
<td>2.8</td>
</tr>
<tr>
<td>acres</td>
<td>-</td>
<td>s</td>
<td>es\mid e</td>
<td>.0000321</td>
<td>.0000318</td>
<td>1.0</td>
</tr>
<tr>
<td>acres</td>
<td>-</td>
<td>s</td>
<td>ss\mid s</td>
<td>.0000342</td>
<td>.0000318</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Spelling Correction and the Noisy Channel

Real-Word Spelling Correction
Real-word spelling errors

- ...leaving in about fifteen *minuets* to go to her house.
- The design *an* construction of the system...
- Can they *lave* him my messages?
- The study was conducted mainly *be* John Black.

- 25-40% of spelling errors are real words  *Kukich 1992*
Solving real-world spelling errors

- For each word in sentence
  - Generate candidate set
    - the word itself
    - all single-letter edits that are English words
    - words that are homophones
  - Choose best candidates
    - Noisy channel model
    - Task-specific classifier
Noisy channel for real-word spell correction

- Given a sentence $w_1, w_2, w_3, \ldots, w_n$
- Generate a set of candidates for each word $w_i$
  - $\text{Candidate}(w_1) = \{w_1, w'_1, w''_1, w'''_1, \ldots\}$
  - $\text{Candidate}(w_2) = \{w_2, w'_2, w''_2, w'''_2, \ldots\}$
  - $\text{Candidate}(w_n) = \{w_n, w'_n, w''_n, w'''_n, \ldots\}$
- Choose the sequence $W$ that maximizes $P(W)$
Noisy channel for real-word spell correction
Simplification: One error per sentence

- Out of all possible sentences with one word replaced:
  - $w_1, w_2', w_3, w_4$ two offthew
  - $w_1, w_2, w_3', w_4$ two of the
  - $w_1', w_2, w_3, w_4$ too ofthew
  - ...

- Choose the sequence $W$ that maximizes $P(W)$