Noisy Channel, N-grams & Smoothing

Lecture #9

Computational Linguistics

CMPSCI 591N, Spring 2006

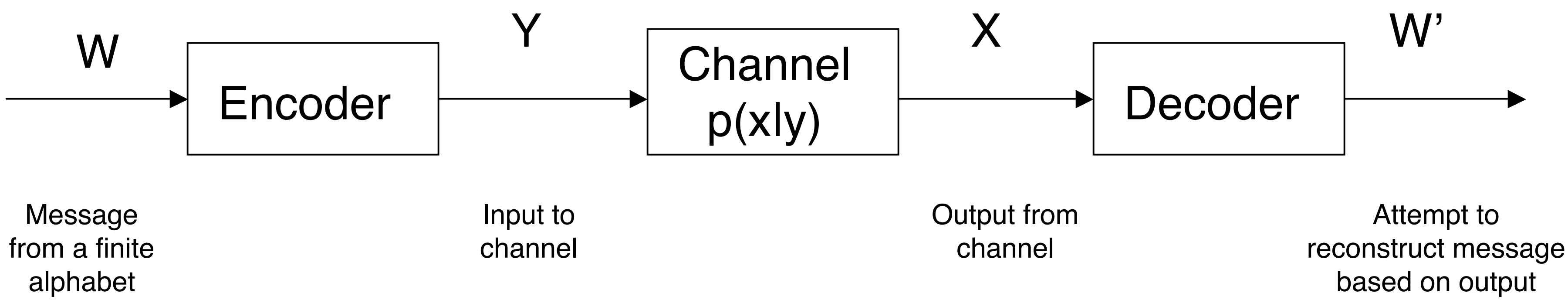
University of Massachusetts Amherst

Andrew McCallum

Today's Main Points

- Course feedback.
- Application of the Noisy Channel Model
- Markov Models
  - including Markov property definition
- Smoothing
  - Laplace, (Lidstone's, Held-out, Good-Turing.)

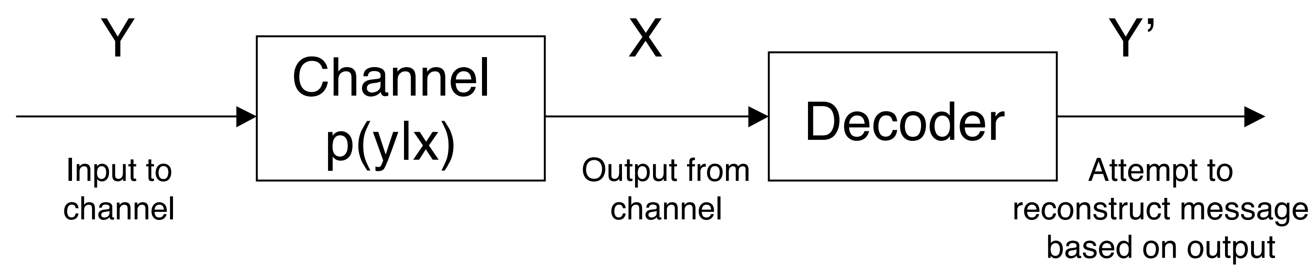
Noisy Channel Model



- Optimize Encoder for throughput and accuracy.
  - compression: remove all redundancy
  - accuracy: adding controlled redundancy
- Capacity: rate at which can transmit information with arbitrarily low probability of error in W’

$$C = \max_{p(Y)} I(X;Y)$$

Noisy Channel in NLP



$$Y' = \arg\max_y p(y|x) = \arg\max_y \frac{p(y)p(x|y)}{p(x)} = \arg\max_y \underbrace{p(y)}_{\text{language model}}\underbrace{p(x|y)}_{\text{channel prob.}}$$

Application	Input	Output	p(y)	p(x|y)
Machine Translation	L1 word sequences	L2 word sequences	p(L1) in a language model	translation model
Optical Character Recognition (OCR)	actual text	text with OCR errors	prob of language text	model of OCR errors
Part of Speech (POS) tagging	POS tag sequences	English word sequence	prob of POS sequences	probability of word given tag
Speech Recognition	word sequences	acoustic speech signal	prob of word sequences	acoustic model
Document classification	class label	word sequence in document	class prior probability	p(L1) from each class

Probabilistic Language Modeling

- Assigns probability p(t) to a word sequence
  $t = w_1 w_2 w_3 w_4 w_5 w_6 \ldots$
- Chain rule and joint/conditional probabilities for text t:

$$p(t) = p(w_1 ... w_n) = p(w_1) ... p(w_n | w_1, ... w_{n-1})$$

$$= \prod_{i=1}^{n} p(w_i | w_1 ... w_{i-1})$$

where

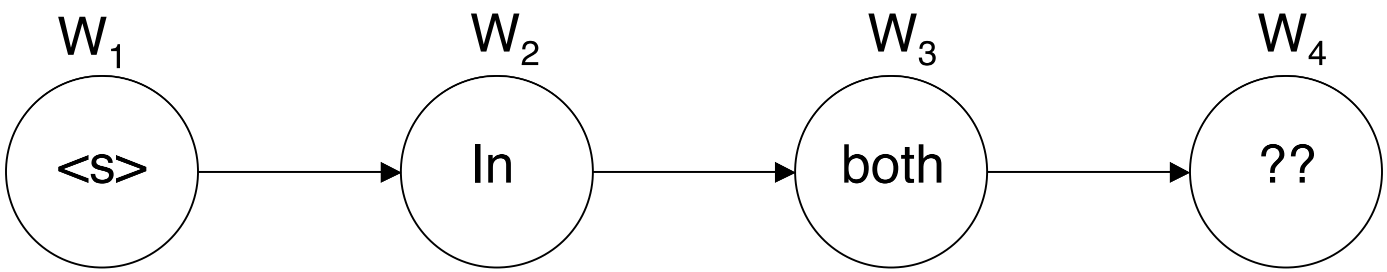
$$p(w_k | w_1 ... w_{k-1}) = \frac{p(w_1 ... w_k)}{p(w_1 ... w_{k-1})} \sim \frac{C(w_1 ... w_k)}{C(w_1 ... w_{k-1})}$$

The chain rule leads to a history-based model:
we predict following things from past things.

n-gram models
the classic example of a statistical model of language

- Each word is predicted according to a conditional distribution based on <u>limited</u> context
- Conditional Probability Table (CPT): p(X|"both")
  - p(of|both) = 0.066
  - p(to|both) = 0.041
  - p(in|both) = 0.038
- a.k.a. Markov (chain) models
  - sequences of random variables in which the future variable is determined by the present variable, but is independent of the way in which the present state arose from its predecessors

1-gram model



First-order Markov model, $P(w_t|w_{t-1})$

- Simplest linear graphical model
- Words are random variables, arrows are direct dependencies between them (CPTs)
- These simple engineering models have been amazingly successful.

n-th order Markov models

- First order Markov assumption = bigram

$$p(w_k|w_1...w_{k-1}) \simeq p(w_k|w_{k-1}) = \frac{p(w_{k-1}w_k)}{p(w_{k-1})}$$

- Similarly, n-th order Markov assumption

- Most commonly, trigram (2nd order)

$$p(w_k|w_1...w_{k-1}) \simeq p(w_k|w_{k-2}w_{k-1}) = \frac{p(w_{k-2}w_{k-1}w_k)}{p(w_{k-2}w_{k-1})}$$

Andrei Andreyevich Markov

1856 - 1922

- Graduate of Saint Petersburg University (1878), where he began a professor in 1886.
- Mathematician, teacher political activist
  - In 1913, when the government celebrated the 300th anniversary of the House of Romanov family, Markov organized a counter-celebration of the 200th anniversary of Bernoulli's discovery of the Law of Large Numbers.
- Markov was also interested in poetry and he made studies of poetic style.

Markov's Model

- Took 20,000 characters from Pushkin's *Eugene Onegin* to see if it could be approximated by a simple chain of characters.

	vowel	consonant
vowel	0.128	0.872
consonant	0.663	0.337

Markov Approximations to English

- Zero-order approximation, P(c)
  - XFOML RXKXRJFFUJ ZLPWCFWKCRJ FFJEYVKCQSGHYD QPAAMKBZAACIBZLHJQD
- First-order approximation, P(c|c)
  - OCRO HLI RGWR NWIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA
- Second-order approximation, P(c|c,c)
  - ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY ACHIN D ILONASIVE TUCOOWE AT TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE

[From Shannon's original paper]

Markov Approximations to English (cont.)

- Third-order approximation, P(w|w,w,w)
  - IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME OF DEMONSTURES OF THE REPTABIN IS REGOACTIONA OF CRE
- Markov Random Field with 1000 “features”
  - WAS REASER IN THERE TO WILL WAS BY HOMES THING BE RELOVERATED THER WHICH CONSISTS AT FORES ANDITING WITH PROVERAL THE CHESTRAING FOR HAVE TO INTRALLY OF QUT DIVERAL THIS OFFECT INATEVER THIFER CONTRANDED STATER

[Della Pietra, Della Pietra & Lafferty, 1997]

Word-based Approximations

- First-order approximation
  – representing and speedily is an good apt or come can different natural here he the a in came the to of to expert gray come to furnishes the line message had be
- Second-order approximation
  – the head and in frontal attack on an English writer that the character of this point is therefore another method for the letters that the time of who ever told the problem for an unexpected

Shannon's comment (1948): *"It would be interesting if further approximations could be constructed, but the labor involved becomes enormous at the next stage."*

n-gram models

- Core language model for the engineering task of better predicting the next word:
  - Speech recognition
  - OCR
  - Context-sensitive spelling correction
- It has only recently that improvements have been made for these tasks [Alshawi ‘96, Wu ‘97]
- But linguistically, they are appallingly simple and naïve.

Why might n-gram models not work?

- Relationships (say between subject and verb) can be arbitrarily distant and convoluted, as linguists love to point out:
  - The **man** on the sidewalk, without pausing to look at what was happening down the street, and quite oblivious to the situation that was about to befall him, confidently **strode** into the center of the road.

Why do they work?

- That kind of thing doesn’t happen much
- Collins (1997)
  - 74% of dependencies (in the Penn Treebank, WSJ) are with an adjacent word (95% with one less than 5 words away), once one treats simple NPs as units

Evaluation of language models

- Best evaluation of probability model is task-based!
- As substitute for evaluating one component, standardly use corpus per-word cross entropy:

$$H(X,p) = -\frac{1}{n}\sum_{i=1}^{n} \log_2 p(w_i|w_1, ..., w_{i-1})$$

- Or perplexity
  - units = average number of choices, scaled for uniform distr.
  - high = unpredictable

$$PP(X,p) = 2^{H(X,p)} = \left[\prod_{i=1}^{n} p(w_i|w_1, ..., w_{i-1})\right]^{-1/n}$$

Parameter Estimation

Maximum Likelihood Estimate

- Relative frequency
- Makes training data a probable as possible
- Overfits

$$p(w_2|w_1) = \frac{C(w_1, w_2)}{C(w_1)}$$

Limitations of the Maximum Likelihood Estimator

- Problem: often infinitely surprised when unseen word appears, P(*unseen*) = 0
  - Problem: this happens commonly
  - Probabilities of zero-count words are too low
  - Probabilities of nonzero-count words are too high
  - Estimates for high count words are fairly accurate
  - Estimates for low count words are unstable
  - We need "smoothing"

Sparsity

- How often does an every day word like “kick” occur in a million words of text?
  - “kick”: about 10 [depends vastly on genre, of course]
  - “wrist”: about 5
- Normally we want to know about something bigger than a single word, like how often you “kick a ball”, or how often the dative alternation “he kicked the baby a toy” occurs.
- How often can we expect that to occur in 1 million words?
- Almost never.
- “There’s no data like more data”
  - Must be of the right domain

Severity of the sparse data problem

count	2-grams	3-grams
1	8,045,024	53,737.350
2	2,065,469	9,229,958
3	970,434	3,654,791
>4	3,413,290	8,728,789
>0	14,494,217	75,349,888
possible	6.8×10^{10}	1.7×10^{16}

Vocab size 260,741 words, 365M words training

The Zero Problem

- Necessarily some zeros
  - trigram model: 1.7×10^{16} parameters
  - but only 2.6×10^{6} words of training data
- How should we distribute some probability mass over all possibilities in the model
  - optimal situation: even the least frequent trigram would occur several times, in order to distinguish its probability versus other trigrams
  - optimal situation cannot happen, unfortunately (how much data would we need?)
- Two kinds of zeros: $p(w|h)=0$, or even $p(h)=0$

Laplace smoothing

$$p(w_2|w_1) = \frac{C(w_1, w_2) + 1}{C(w_1) + V}$$

- *V* is the vocabulary size (assume fixed, closed vocabulary)
- This is the Bayesian *maximum a posteriori* estimator you get by assuming a uniform prior on multinomials (a Dirichlet prior)

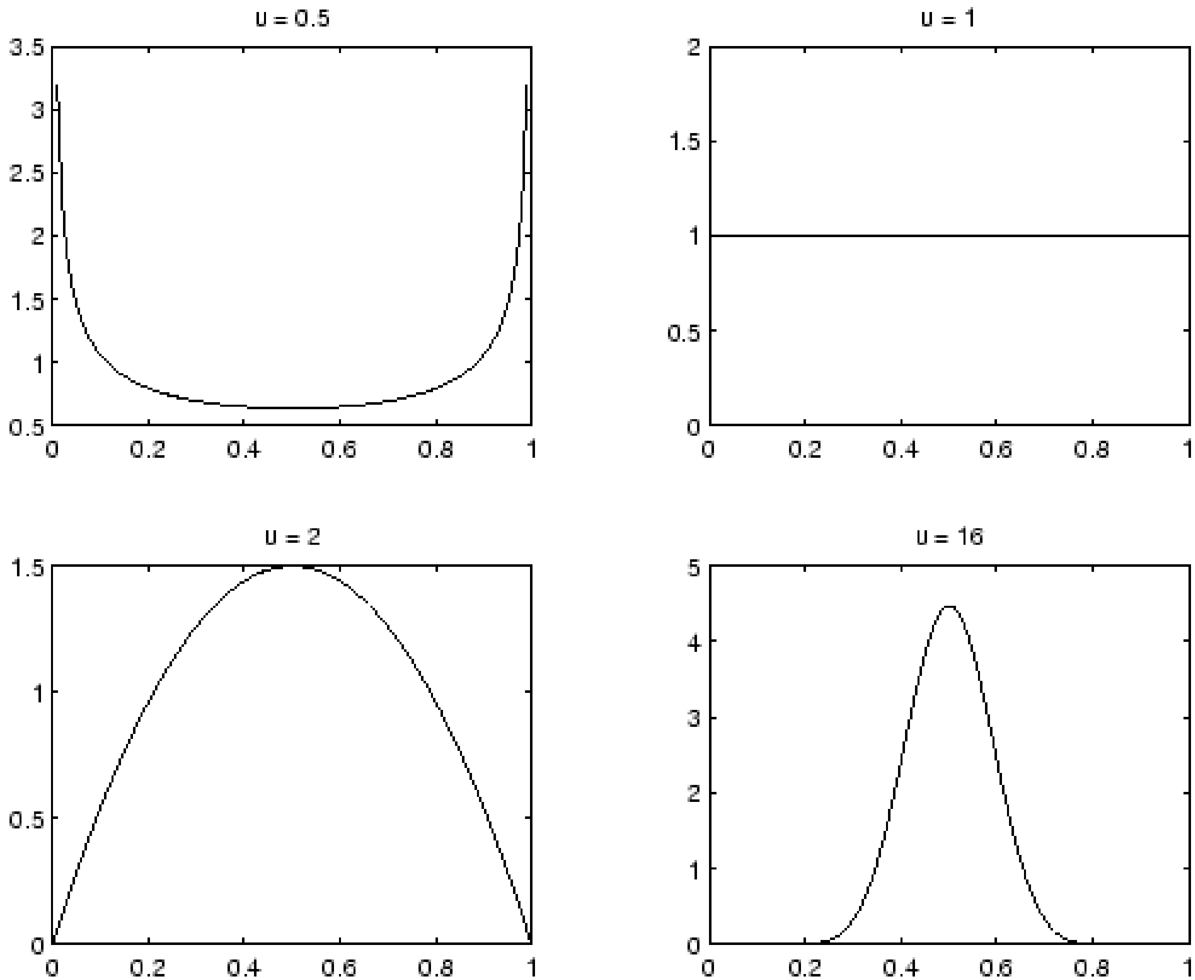
Dirichlet Distribution

- "Multinomial" is a die: a distribution over a finite alphabet of outcomes
- "Dirichlet" is a dice generator: a distribution over multinomials!

$$p(q) \sim \mathrm{Dirichlet}(\alpha_1, \alpha_2, ...\alpha_K) = \frac{\Gamma(\sum_k \alpha_k)}{\prod_k \Gamma(\alpha_k)} \prod_k q_k^{\alpha_k - 1}$$

- It is a conjugate prior for multinomials

Dirichlet Examples



For $\alpha_1 = \sigma_2 = \alpha$

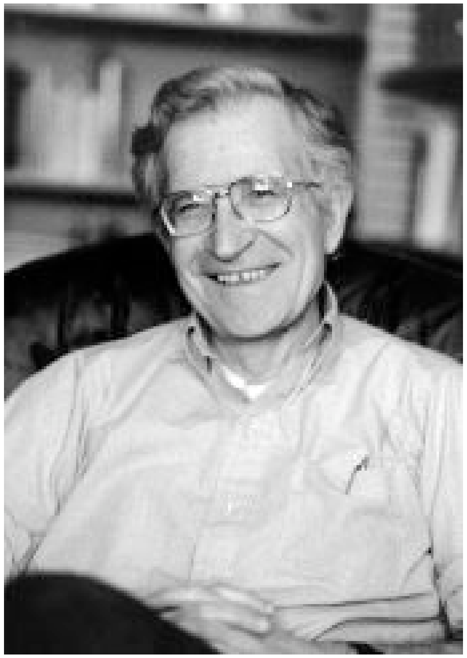
Laplace Smoothing

- Problem: gives too much probability mass to unseens
- Not good for large vocabulary, comparatively little data (NLP!)
- e.g. 10,000 word vocab, 1,000,000 words of training data, but "comes across" occurs 10 times. Of those, 8 times next word is "as"
  - P_{MLE}(as|comes across) = 0.8
  - $P_{Laplace}$(as|comes across) = (8+1)/(10+10000)=0.0009
- Quick fix: Lidstone's law (Mitchell's 1997 "*m*-estimate"):

$$p(w_2|w_1) = \frac{C(w_1, w_2) + \lambda}{C(w_1) + \lambda V}$$

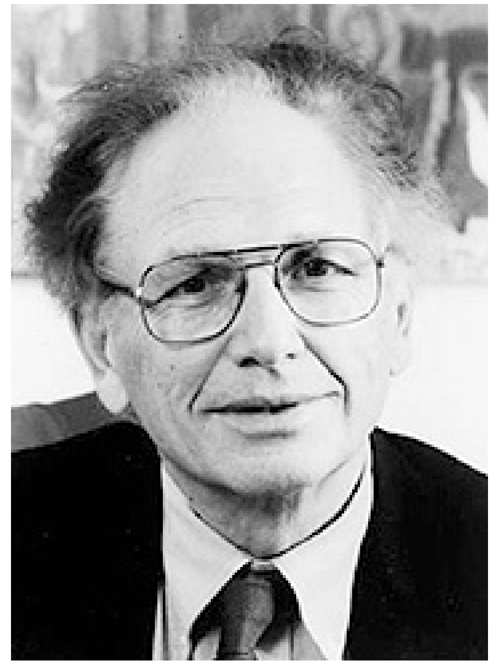
$$\text{for} \lambda < 1, \ \text{e.g. } 1/2 \text{ or } 0.05$$

Statistical Language Modeling



Noam Chomsky

But it must be recognized that the notion of "probability of a sentence" is an entirely useless one, under any known interpretation of the term. (1969)



Fred Jelinek

Anytime a linguist leaves the group, the [speech] recognition rate goes up. (while at IBM speech group, 1988).

- Progress in the field is often dominated, not by the need to create fancier more complex models,
- but by the need to do a good job of estimating parameters for the simpler models we already have.

- Real benefit comes from targeted enhancements, and sharp tool set of excellent estimation techniques

Distinctiveness of NLP as an ML problem

- Most structure is hidden
- Relational, constraint satisfaction nature
- Long pipelines, with cascading errors
- Large and strange, sparse discrete distributions
- Large scale
- Feature-driven; performance driven

HW#4

As, usual, your choice:

- Naive Bayes Classifier
  - Spam vs Ham
  - English vs French vs Spanish vs Klingon
  - “Sliding window” Part-of-Speech” tagger
- N-gram language model
  - Train and generate language
  - Use for spelling correction (there vs their)

HW#4 Help
Accuracy Evaluation

Result of running classifier on a test set:

```
filename trueclass predclass p(predclass|doc)
filename trueclass predclass p(predclass|doc)
filename trueclass predclass p(predclass|doc)
...
```

	true spam	**true ham**
pred spam	TP	FP
pred ham	FN	TN

Accuracy = (TP+TN) / (TP+TN+FP+FN)
Precision = TP / (TP+FP)
Recall = TP / (TP+FN)
F1 = harmonic mean of Precision & Recall

HW#4 Help
Precision-Recall Curve

Result of running classifier on a test set:

```
filename trueclass predclass p(predclass|doc)
filename trueclass predclass p(predclass|doc)
filename trueclass predclass p(predclass|doc)
...
```

	true spam	**true ham**
pred spam	TP	FP
pred ham	FN	TN

Accuracy = (TP+TN) / (TP+TN+FP+FN)
Precision = TP / (TP+FP)
Recall = TP / (TP+FN)
F1 = harmonic mean of Precision & Recall

HW#4 Help
Accuracy-Coverage Curve

Result of running classifier on a test set:

```
filename trueclass predclass p(predclass|doc)
filename trueclass predclass p(predclass|doc)
filename trueclass predclass p(predclass|doc)
...
```

	true spam	**true ham**
pred spam	TP	FP
pred ham	FN	TN

Accuracy = (TP+TN) / (TP+TN+FP+FN)
Precision = TP / (TP+FP)
Recall = TP / (TP+FN)
F1 = harmonic mean of Precision & Recall

HW#4 Help
Working with log-probabilities

$$p(c|d) \propto p(c) \prod_i p(w_i|c)$$

$$\log(p(c|d)) \propto \log(p(c)) + \sum_i \log(p(w_i|c))$$

- Getting back to p(c|d)
  - Subtract a constant to make all non-positive
  - exp()