Introduction to Natural Language Processing I [Statistické metody zpracování přirozených jazyků I] (NPFL067)

http://ufal.mff.cuni.cz/courses/npfl067

prof. RNDr. Jan Hajič, Dr. / doc. RNDr. Pavel Pecina, Ph.D.
ÚFAL MFF UK
{hajic,pecina}@ufal.mff.cuni.cz
http://ufal.mff.cuni.cz/jan-hajic
http://ufal.mff.cuni.cz/~pecina/index.html

Intro to NLP

- Instructors: Jan Hajič / Pavel Pecina
  - ÚFAL MFF UK, office: 420 / 422 MS
  - Hours: J. Hajic: Mon 10:00-11:00
  - preferred contact: **{hajic,pecina}@ufal.mff.cuni.cz**
- Room & time:
  - lecture: room S1, Tue 12:20-13:50
  - seminar [cvičení] room S1, Tue 14:00-15:30
  - Oct 2, 2018 – Jan 8, 2019
  - Final written exam (probable) date: Jan 15, 2019

Textbooks you need

- Manning, C. D., Schütze, H.:
  - *Foundations of Statistical Natural Language Processing*. The MIT Press. 1999. ISBN 0-262-13360-1. **[required]**
- Jurafsky, D., Martin, J.H.:
  - *Speech and Language Processing.* Prentice-Hall. 2000. ISBN 0-13-095069-6 and later editions. **[recommended].**

Other reading

- Charniak, E:
  - *Statistical Language Learning.* The MIT Press. 1996. ISBN 0-262-53141-0.
- Cover, T. M., Thomas, J. A.:
  - *Elements of Information Theory.* Wiley. 1991. ISBN 0-471-06259-6.
- Jelinek, F.:
  - *Statistical Methods for Speech Recognition*. The MIT Press. 1998. ISBN 0-262-10066-5
- Proceedings of major conferences:
  - ACL (Assoc. of Computational Linguistics)
  - EACL/NAACL/IJCNLP (European/American/Asian Chapter of ACL)
  - EMNLP (Empirical Methods in NLP)
  - COLING (Intl. Committee of Computational Linguistics)

Course requirements

- Grade components: requirements & weights:
  - Homeworks (1): 50%
  - Final Exam: 50%
- Exam:
  - approx. 4 questions:
    - mostly explanatory answers (1/4 page or so),
    - algorithms
    - only a few multiple choice questions

Homeworks

- Homework:
  - Entropy, Language Modeling
- Organization
    - (little) paper-and-pencil exercises, lot of programming
    - turning-in mechanism: see the web
    - no plagiarism!
- Deadline
  - Jan. 31, 2018
  - Late penalty: 5% of grade (0-100) per day (max. 50%)

Course segments

- Intro & Probability & Information Theory
  - The very basics: definitions, formulas, examples.
- Language Modeling
  - n-gram models, parameter estimation
  - smoothing (EM algorithm)
- Words and the Lexicon
  - word classes, mutual information, bit of lexicography
- Hidden Markov Models
  - background, algorithms, parameter estimation

NLP: The Main Issues

- Why is NLP difficult?
  - many "words", many "phenomena" --> many "rules"
    - **OED: 400k words; Finnish lexicon (of forms): ~2 . 10^7**
    - **sentences, clauses, phrases, constituents, coordination, negation, imperatives/questions, inflections, parts of speech, pronunciation, topic/focus, and much more!**
  - irregularity (exceptions, exceptions to the exceptions, ...)
    - **potato -> potatoes (tomato, hero,...); photo -> photos,** and even: both **mango -> mangos** or **-> mangoes**
    - Adjective / Noun order: **new book, electrical engineering, general regulations, flower garden, garden flower, ...:** but **Governor General**

Difficulties in NLP (cont.)

- ambiguity
  - **books: NOUN** or **VERB?**
    - you **need** many <u>books</u> *vs.* she <u>**books**</u> her flights online
  - **No left turn weekdays 4-6 pm / except transit vehicles (***Charles Street at Cold Spring***)**
    - when may transit vehicles turn: <u>Always</u>? <u>Never</u>?
  - **Thank you for not smoking, drinking, eating or playing radios without earphones.** *(MTA bus)*
    - Thank you for not eating without earphones??
    - or even: Thank you for ~~not~~ drinking without earphones!?
  - **My neighbor's hat was taken by wind. He tried to catch it.**
    - ...catch the <u>wind</u> *or* ...catch the <u>hat</u> ?

(Categorical) Rules or Statistics?

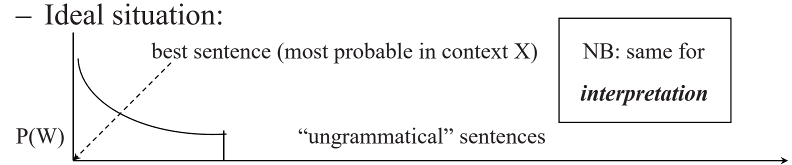
- Preferences:
  - clear cases: context clues: she books --> books is a verb
    - rule: if an ambiguous word (verb/nonverb) is preceded by a matching personal pronoun -> word is a verb
  - less clear cases: pronoun reference
    - she/he/it refers to the most recent noun or pronoun (?) (but maybe we can specify exceptions)
  - selectional:
    - catching hat >> catching wind (but why not?)
  - semantic:
    - never thank for drinking in a bus! (but what about the earphones?)

Solutions

- Don't guess if you know:
  - **morphology (inflections)**
  - **lexicons (lists of words)**
  - **unambiguous names**
  - **perhaps some (really) fixed phrases**
  - **syntactic rules?**
- Use statistics (based on real-world data) for preferences (only?)
  - **No doubt: but this is the big question!**

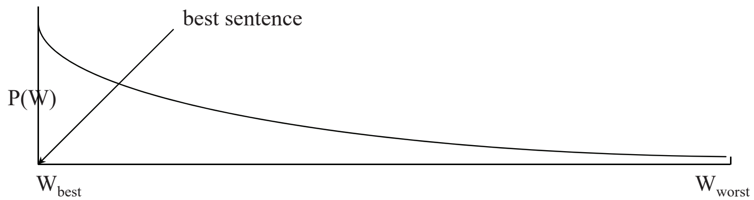
Statistical NLP

- Imagine:
  - Each sentence W = { $w_1, w_2, ..., w_n$ } gets a probability P(W|X) in a context X (think of it in the intuitive sense for now)
  - For every possible context X, sort all the imaginable sentences W according to P(W|X):
  - Ideal situation:



Real World Situation

- Unable to specify set of grammatical sentences today using fixed “categorical” rules (maybe never, cf. arguments in MS)
- Use statistical “model” based on **REAL WORLD DATA** and care about the best sentence only (disregarding the “grammaticality” issue)



Probability

Experiments & Sample Spaces

- Experiment, process, test, ...
- Set of possible basic outcomes: sample space Ω
  - coin toss (Ω = {head,tail}), die (Ω = {1..6})
  - yes/no opinion poll, quality test (bad/good) (Ω = {0,1})
  - lottery ($|\Omega| \cong 10^7 .. 10^{12}$)
  - # of traffic accidents somewhere per year (Ω = N)
  - spelling errors ($\Omega = Z^*$), where Z is an alphabet, and Z^* is a set of possible strings over such and alphabet
  - missing word ($|\Omega| \cong$ vocabulary size)

Events

- Event A is a set of basic outcomes
- Usually $A \subset \Omega$, and all $A \in 2^\Omega$ (the event space)
  - Ω is then the certain event, $\emptyset$ is the impossible event
- Example:
  - experiment: three times coin toss
    - **Ω = {HHH, HHT, HTH, HTT, THH, THT, TTH, TTT}**
  - count cases with exactly two tails: then
    - **A = {HTT, THT, TTH}**
  - all heads:
    - **A = {HHH}**

Probability

- Repeat experiment many times, record how many times a given event A occurred (“count” c_1).
- Do this whole series many times; remember all c_is.
- Observation: if repeated really many times, the ratios of $\mathbf{c_i/T_i}$ (where T_i is the number of experiments run in the *i-th* series) are close to some (unknown but) **constant** value.
- Call this constant a ***probability of A***. Notation: **p(A)**

Estimating probability

- Remember: ... close to an *unknown* constant.
- We can only estimate it:
  - from a single series (typical case, as mostly the outcome of a series is given to us and we cannot repeat the experiment), set

    $$p(A) = c_1/T_1.$$

  - otherwise, take the weighted average of all c_i/T_i (or, if the data allows, simply look at the set of series as if it is a single long series).
- This is the **best** estimate.

Example

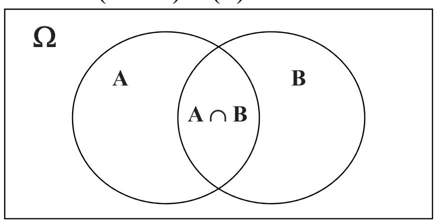
- Recall our example:
  - experiment: three times coin toss
    - **Ω = {HHH, HHT, HTH, HTT, THH, THT, TTH, TTT}**
  - count cases with exactly two tails: A **= {HTT, THT, TTH}**
- Run an experiment 1000 times (i.e. 3000 tosses)
- Counted: 386 cases with two tails (**HTT**, **THT**, or **TTH**)
- estimate: p(A) = 386 / 1000 = .386
- Run again: 373, 399, 382, 355, 372, 406, 359
  - p(A) = .379 (weighted average) or simply 3032 / 8000
- *Uniform* distribution assumption: p(A) = 3/8 = .375

Basic Properties

- Basic properties:
  - $p: 2^{\Omega} \rightarrow [0,1]$
  - $p(\Omega) = 1$
  - Disjoint events: $p(\cup A_i) = \sum_i p(A_i)$
- [NB: *axiomatic definition* of probability: take the above three conditions as axioms]
- Immediate consequences:
  - $p(\emptyset) = 0$, $p(\bar{A}) = 1 - p(A)$, $A \subseteq B \Rightarrow p(A) \leq p(B)$
  - $\sum_{a \in \Omega} p(a) = 1$

Joint and Conditional Probability

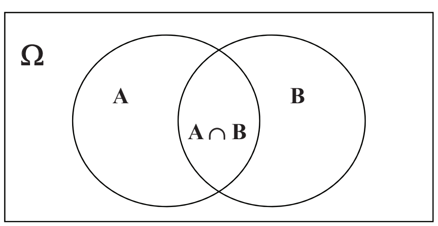
- p(A,B) = p(A $\cap$ B)
- p(A|B) = p(A,B) / p(B)
  - Estimating form counts:
    - **p(A|B) = p(A,B) / p(B) = (c(A $\cap$ B) / T) / (c(B) / T) = = c(A $\cap$ B) / c(B)**



Bayes Rule

- p(A,B) = p(B,A) since p(A $\cap$ B) = p(B $\cap$ A)
  - therefore: p(A|B) p(B) = p(B|A) p(A), and therefore

$$p(A|B) = p(B|A)\ p(A) / p(B) \quad !$$



Independence

- Can we compute p(A,B) from p(A) and p(B)?
- Recall from previous foil:

p(A|B) = p(B|A) p(A) / p(B)

p(A|B) p(B) = p(B|A) p(A)

p(A,B) = p(B|A) p(A)

... we're almost there: how p(B|A) relates to p(B)?

- p(B|A) = P(B) iff A and B are **independent**

- Example: two coin tosses, weather today and weather on March 4th 1789;
- Any two events for which p(B|A) = P(B)!

Chain Rule

$$p(A_1, A_2, A_3, A_4, ..., A_n) = \quad !$$

$$p(A_1|A_2,A_3,A_4,...,A_n) \times p(A_2|A_3,A_4,...,A_n) \times$$
$$\times\ p(A_3|A_4,...,A_n) \times\ ...\ p(A_{n-1}|A_n) \times p(A_n)$$

- this is a direct consequence of the Bayes rule.

The Golden Rule
(of Classic Statistical NLP)

- Interested in an event A given B (when it is not easy or practical or desirable to estimate p(A|B)):
- take Bayes rule, max over all As:
- $\mathrm{argmax}_A\, p(A|B) = \mathrm{argmax}_A\, p(B|A)\, .\, p(A)$ / ~~p(B)~~ =

$$\mathrm{argmax}_A\, p(B|A)\, p(A) \;!$$

- ... as p(B) is constant when changing As

Random Variable

- is a function $X: \Omega \rightarrow Q$
  - in general: $Q = R^n$, typically R
  - easier to handle real numbers than real-world events
- random variable is *discrete* if Q is <u>countable</u> (i.e. also if <u>finite</u>)
- Example: *die*: natural "numbering" [1,6], *coin*: {0,1}
- Probability distribution:
  - $p_X(x) = p(X=x) =_{df} p(A_x)$ where $A_x = \{a \in \Omega : X(a) = x\}$
  - often just p(x) if it is clear from context what X is

Expectation
Joint and Conditional Distributions

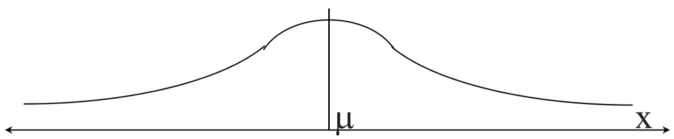
- is a mean of a random variable (weighted average)
  - $E(X) = \sum_{x \in X(\Omega)} x\, .\, p_X(x)$
- Example: one six-sided die: 3.5, two dice (sum) 7
- Joint and Conditional distribution rules:
  - analogous to probability of events
- Bayes: $p_{X|Y}(x,y) =_{notation} p_{XY}(x|y) =_{even\ simpler\ notation}$ **p(x|y) = p(y|x) . p(x) / p(y)**
- Chain rule: **p(w,x,y,z) = p(z).p(y|z).p(x|y,z).p(w|x,y,z)**

Standard distributions

- Binomial (discrete)
  - outcome: 0 or 1 (thus: *bi*nomial)
  - make *n* trials
  - interested in the (probability of) number of successes *r*
- Must be careful: it's not uniform!
- $p_b(r|n) = \binom{n}{r} / 2^n$ (for equally likely outcome)
- $\binom{n}{r}$ counts how many possibilities there are for choosing r objects out of n; = n! / ((n-r)! r!)

Continuous Distributions

- The normal distribution ("Gaussian")
- $p_{norm}(x|\mu,\sigma) = e^{-(x-\mu)^2/(2\sigma^2)}/\sigma\sqrt{2\pi}$
- where:
  - μ is the mean (x-coordinate of the peak) (0)
  - σ is the standard deviation (1)



- other: hyperbolic, t

Essential Information Theory

The Notion of Entropy

- Entropy ~ "chaos", fuzziness, opposite of order, ...
  - you know it:
    - **it is much easier to create "mess" than to tidy things up...**
- Comes from physics:
  - Entropy does not go down unless energy is applied
- Measure of ***uncertainty:***
  - if low... low uncertainty; the higher the entropy, the higher uncertainty, but the higher "surprise" (information) we can get out of an experiment

The Formula

- Let $p_X(x)$ be a distribution of random variable X
- Basic outcomes (alphabet) Ω

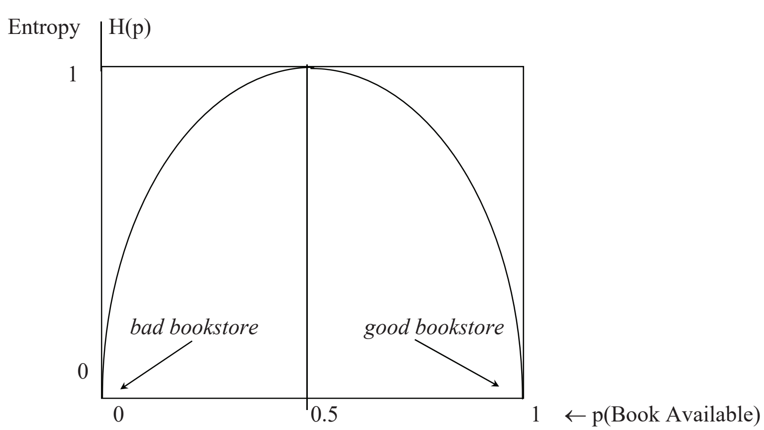
$$H(X) = - \sum_{x \in \Omega} p(x) \log_2 p(x) \quad !$$

- Unit: bits ($\log_{10}$: nats)
- Notation: $H(X) = H_p(X) = H(p) = H_X(p) = H(p_X)$

Using the Formula: Example

- Toss a fair coin: $\Omega = \{head,tail\}$
  - p(head) = .5, p(tail) = .5
  - **H(p)** = - 0.5 $\log_2(0.5)$ + (- 0.5 $\log_2(0.5)$) = $2 \times ((-0.5) \times (-1)) = 2 \times 0.5 =$ **1**
- Take fair, 32-sided die: p(x) = 1 / 32 for every side x
  - **H(p)** = $-\sum_{i = 1..32} p(x_i) \log_2 p(x_i) = - 32 (p(x_1) \log_2 p(x_1))$ (since for all *i* $p(x_i) = p(x_1) = 1/32$) $= -32 \times ((1/32) \times (-5)) =$ **5** *(now you see why it's called* ***bits****?)*
- Unfair coin:
  - p(head) = .2 ... **H(p)** = **.722**; p(head) = .01 ... **H(p)** = **.081**

Example: Book Availability



The Limits

- When H(p) = 0?
  - if a result of an experiment is ***known*** ahead of time:
  - necessarily:

    $$\exists x \in \Omega;\ p(x) = 1\ \&\ \forall y \in \Omega;\ y \neq x \Rightarrow p(y) = 0$$
- Upper bound?
  - none in general
  - for $|\Omega| = n$: $H(p) \leq \log_2 n$
    - **nothing can be more uncertain than the uniform distribution**

Entropy and Expectation

- Recall:
  - $E(X) = \sum_{x \in X(\Omega)} p_X(x) \times x$
- Then:

$$E(\log_2(1/p_X(x))) = \sum_{x \in X(\Omega)} p_X(x) \log_2(1/p_X(x)) =$$

$$= - \sum_{x \in X(\Omega)} p_X(x) \log_2 p_X(x) =$$

$$= H(p_X) =_{notation} H(p)$$

Perplexity: motivation

- Recall:
  - 2 equiprobable outcomes: H(p) = 1 bit
  - 32 equiprobable outcomes: H(p) = 5 bits
  - 4.3 billion equiprobable outcomes: H(p) ~= 32 bits
- What if the outcomes are not equiprobable?
  - 32 outcomes, 2 equiprobable at .5, rest impossible:
    - **H(p) = 1 bit**
  - Any measure for comparing the entropy (i.e. uncertainty/difficulty of prediction) (also) for random variables with *different number of outcomes*?

Perplexity

- Perplexity:
  - $G(p) = 2^{H(p)}$
- ... so we are back at 32 (for 32 eqp. outcomes), 2 for fair coins, etc.
- it is easier to imagine:
  - NLP example: vocabulary size of a vocabulary with uniform distribution, which is equally hard to predict
- the "wilder" (biased) distribution, the better:
  - lower entropy, lower perplexity

Joint Entropy and Conditional Entropy

- Two random variables: X (space Ω),Y (Ψ)
- Joint entropy:
  - no big deal: ((X,Y) considered a single event):

$$H(X,Y) = - \sum_{x \in \Omega} \sum_{y \in \Psi} p(x,y) \log_2 p(x,y)$$

- Conditional entropy:

$$H(Y|X) = - \sum_{x \in \Omega} \sum_{y \in \Psi} \underline{p(x,y)} \log_2 p(y|x)$$

recall that $H(X) = E(\log_2(1/p_X(x)))$

(weighted "average", and <u>weights are not conditional</u>)

Conditional Entropy (Using the Calculus)

- other definition:

$$H(Y|X) = \sum_{x \in \Omega} p(x)\, H(Y|X=x) =$$

for H(Y|X=x), we can use the single-variable definition (x ~ constant)

$$= \sum_{x \in \Omega} p(x) \left(- \sum_{y \in \Psi} p(y|x) \log_2 p(y|x) \right) =$$

$$= - \sum_{x \in \Omega} \sum_{y \in \Psi} p(y|x)\, p(x) \log_2 p(y|x) =$$

$$= - \sum_{x \in \Omega} \sum_{y \in \Psi} p(x,y) \log_2 p(y|x)$$

Properties of Entropy I

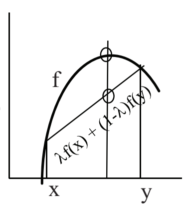
- Entropy is non-negative:
  - $H(X) \geq 0$
  - proof: (recall: $H(X) = - \sum_{x \in \Omega} p(x) \log_2 p(x)$)
    - **log(p(x)) is negative or zero for x ≤ 1,**
    - **p(x) is non-negative; their product p(x)log(p(x) is thus negative;**
    - **sum of negative numbers is negative;**
    - **and -*f* is positive for negative *f***
- Chain rule:
  - H(X,Y) = H(Y|X) + H(X), as well as
  - H(X,Y) = H(X|Y) + H(Y) (since H(Y,X) = H(X,Y))

Properties of Entropy II

- Conditional Entropy is better (than unconditional):
  - $H(Y|X) \leq H(Y)$ (proof on Monday)
- $H(X,Y) \leq H(X) + H(Y)$ (follows from the previous (in)equalities)
  - **equality iff X,Y independent**
  - **[recall: X,Y independent iff p(X,Y) = p(X)p(Y)]**
- H(p) is concave (remember the book availability graph?)
  - concave function f over an interval (a,b):

    $\forall x,y \in (a,b), \forall \lambda \in [0,1]$:

    $f(\lambda x + (1-\lambda)y) \geq \lambda f(x) + (1-\lambda)f(y)$
  - **function f is convex if -f is concave**



- [for proofs and generalizations, see Cover/Thomas]

"Coding" Interpretation of Entropy

- The least (average) number of bits needed to encode a message (string, sequence, series,...) (each element having being a result of a random process with some distribution p): = H(p)
- Remember various compressing algorithms?
  - they do well on data with repeating (= easily predictable = low entropy) patterns
  - their results though have high entropy ⇒ compressing compressed data does nothing

Coding: Example

- How many bits do we need for ISO Latin 1?
  - ⇒ the trivial answer: 8
- Experience: some chars are more common, some (very) rare:
  - **...so what if we use more bits for the rare, and less bits for the frequent? [be careful: want to decode (easily)!]**
  - **suppose: p('a') = 0.3, p('b') = 0.3, p('c') = 0.3, the rest: p(x)≅ .0004**
  - **code: 'a' ~ 00, 'b' ~ 01, 'c' ~ 10, rest: $11b_1b_2b_3b_4b_5b_6b_7b_8$**
  - **code acbbécbaac:** 0010010111000011111001000010

    a c b b é c b a a c
  - **number of bits used: 28 (vs. 80 using "naive" coding)**
- code length ~ 1 / probability; conditional prob OK!

Entropy of a Language

- Imagine that we produce the next letter using

$$p(l_{n+1}|l_1,...,l_n),$$

where $l_1,...,l_n$ is the sequence of ***all*** the letters which had been uttered so far (i.e. ***n*** is really big!); let's call $l_1,...,l_n$ the ***history*** h (h_{n+1}), and all histories H:
- Then compute its entropy:
  - $- \sum_{h \in H} \sum_{l \in A} p(l,h) \log_2 p(l|h)$
- Not very practical, isn't it?

Kullback-Leibler Distance (Relative Entropy)

- Remember:
  - long series of experiments... c_i/T_i oscillates around some number... we can only estimate it... to get a distribution q.
- So we get a distribution q; (sample space Ω, r.v. X)

  the true distribution is, however, p. (same Ω, X)

  ⇒ how big error are we making?
- D(p||q) (the Kullback-Leibler distance):

$$D(p||q) = \sum_{x \in \Omega} p(x) \log_2 (p(x)/q(x)) = E_p \log_2 (p(x)/q(x))$$

Comments on Relative Entropy

- Conventions:
  - $0 \log 0 = 0$
  - $p \log (p/0) = \infty$ (for $p > 0$)
- Distance? (less "misleading": Divergence)
  - not quite:
    - **not symmetric: D(p||q) ≠ D(q||p)**
    - **does not satisfy the triangle inequality**
  - but useful to look at it that way
- H(p) + D(p||q): bits needed for encoding p if q is used

Mutual Information (MI)

in terms of relative entropy

- Random variables X, Y; $p_{X \cap Y}(x,y)$, $p_X(x)$, $p_Y(y)$
- Mutual information (between two random variables X,Y):

$$I(X,Y) = D(p(x,y) \,||\, p(x)p(y))$$

- I(X,Y) measures how much (our knowledge of) Y contributes (on average) to easing the prediction of X
- or, how p(x,y) deviates from (independent) p(x)p(y)

Mutual Information: the Formula

- Rewrite the definition: [recall: $D(r\|s) = \sum_{v \in \Omega} r(v) \log_2 (r(v)/s(v))$; substitute $r(v) = p(x,y)$, $s(v) = p(x)p(y)$; $\langle v \rangle \sim \langle x,y \rangle$]

$$I(X,Y) = D(p(x,y) \,\|\, p(x)p(y)) =$$
$$= \sum_{x \in \Omega} \sum_{y \in \Psi} p(x,y) \log_2 (p(x,y)/p(x)p(y)) \quad !$$

- Measured in bits (what else? :-)

From Mutual Information to Entropy

- by how many bits the knowledge of Y ***lowers*** the entropy H(X):

$$I(X,Y) = \sum_{x \in \Omega} \sum_{y \in \Psi} p(x,y) \log_2 (p(x,y)/p(y)p(x)) =$$

...use $p(x,y)/p(y) = p(x|y)$

$$= \sum_{x \in \Omega} \sum_{y \in \Psi} p(x,y) \log_2 (p(x|y)/p(x)) =$$

...use $\log(a/b) = \log a - \log b$ ($a \sim p(x|y)$, $b \sim p(x)$), distribute sums

$$= \sum_{x \in \Omega} \sum_{y \in \Psi} p(x,y) \log_2 p(x|y) - \sum_{x \in \Omega} \sum_{y \in \Psi} p(x,y) \log_2 p(x) =$$

...use def. of $H(X|Y)$ (left term)*, and $\sum_{y \in \Psi} p(x,y) = p(x)$* (right term)

$$= - H(X|Y) + (- \sum_{x \in \Omega} p(x) \log_2 p(x)) =$$

...use def. of $H(X)$ (right term)*, swap terms*

$$= H(X) - H(X|Y)$$ ...by symmetry, $= H(Y) - H(Y|X)$

Properties of MI vs. Entropy

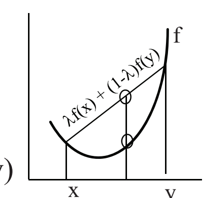
- $I(X,Y) = H(X) - H(X|Y)$ = number of bits the knowledge of Y lowers the entropy of X

  $= H(Y) - H(Y|X)$ (prev. foil, symmetry)

Recall: $H(X,Y) = H(X|Y) + H(Y) \Rightarrow -H(X|Y) = H(Y) - H(X,Y) \Rightarrow$

- $I(X,Y) = H(X) + H(Y) - H(X,Y)$
- $I(X,X) = H(X)$ (since $H(X|X) = 0$)
- $I(X,Y) = I(Y,X)$ (just for completeness)
- $I(X,Y) \geq 0$ ... let's prove that now (as promised).

Jensen's Inequality



- Recall: f is convex on interval (a,b) iff
  $\forall x,y \in (a,b), \forall \lambda \in [0,1]$:
  $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$
- J.I.: for distribution p(x), r.v. X on Ω, and convex f,
  $$f\left(\sum_{x \in \Omega} p(x)\, x\right) \leq \sum_{x \in \Omega} p(x)\, f(x)$$
- Proof (idea): by induction on the number of basic outcomes;
- start with $|\Omega| = 2$ by:
  - **$p(x_1)f(x_1) + p(x_2)f(x_2) \geq f(p(x_1)x_1 + p(x_2)x_2)$ ($\Leftarrow$ def. of convexity)**
  - **for the induction step ($|\Omega| = k \rightarrow k+1$), just use the induction hypothesis and def. of convexity (again).**

Information Inequality

$D(p||q) \geq 0$ **!**

- Proof:

$0 = -\log 1 = -\log \sum_{x\in\Omega} q(x) = -\log \sum_{x\in\Omega} (q(x)/p(x))p(x) \leq$

...apply Jensen's inequality here (- log is convex)...

$\leq \sum_{x\in\Omega} p(x)\,(-\log(q(x)/p(x))) = \sum_{x\in\Omega} p(x)\log(p(x)/q(x)) =$

$= D(p||q)$

Other (In)Equalities and Facts

- Log sum inequality: for $r_i, s_i \geq 0$

  $\sum_{i=1..n} (r_i \log(r_i/s_i)) \leq (\sum_{i=1..n} r_i) \log(\sum_{i=1..n} r_i / \sum_{i=1..n} s_i))$
- $D(p||q)$ is convex [in p,q] ($\Leftarrow$ log sum inequality)
- $H(p_X) \leq \log_2|\Omega|$, where Ω is the sample space of p_X

  Proof: uniform u(x), same sample space Ω: $\sum p(x) \log u(x) = -\log_2|\Omega|$;

  $\log_2|\Omega| - H(X) = -\sum p(x) \log u(x) + \sum p(x) \log p(x) = D(p||u) \geq 0$
- H(p) is concave [in p]:

  Proof: from $H(X) = \log_2|\Omega| - D(p||u)$, $D(p||u)$ convex $\Rightarrow H(x)$ concave

Cross-Entropy

- Typical case: we've got series of observations

  $T = \{t_1, t_2, t_3, t_4, ..., t_n\}$ (numbers, words, ...; $t_i \in \Omega$);

  estimate (simple):

  $\forall y \in \Omega$: $\tilde{p}(y) = c(y) / |T|$, def. $c(y) = |\{t \in T; t = y\}|$
- ...but the true p is unknown; every sample is too small!
- Natural question: how well do we do using $\tilde{p}$ [instead of p]?
- Idea: simulate actual p by using a different T'

  (or rather: by using different observation we simulate the insufficiency of T vs. some other data ("random" difference))

Cross Entropy: The Formula

- $H_{p'}(\tilde{p}) = H(p') + D(p'||\tilde{p})$

$H_{p'}(\tilde{p}) = -\sum_{x\in\Omega} p'(x) \log_2 \tilde{p}(x)$ **!**

- p' is certainly not the true p, but we can consider it the "real world" distribution against which we test $\tilde{p}$
- note on notation (confusing...): $p/p' \leftrightarrow \tilde{p}$, also $H_{T'}(p)$
- (Cross)Perplexity: $G_{p'}(p) = G_{T'}(p) = 2^{H_{p'}(\tilde{p})}$

Conditional Cross Entropy

- So far: "unconditional" distribution(s) p(x), p'(x)...
- In practice: virtually always conditioning on context
- Interested in: sample space Ψ, r.v. Y, $y \in \Psi$;

  context: sample space Ω, r.v. X, $x \in \Omega$;:

  "our" distribution p(y|x), test against p'(y,x),

  which is taken from some independent data:

  $H_{p'}(p) = - \sum_{y \in \Psi, x \in \Omega} p'(y,x) \log_2 p(y|x)$

Sample Space vs. Data

- In practice, it is often inconvenient to sum over the sample space(s) Ψ, Ω (especially for cross entropy!)
- Use the following formula:

$$H_{p'}(p) = - \sum_{y \in \Psi, x \in \Omega} p'(y,x) \log_2 p(y|x) = - 1/|T'| \sum_{i = 1..|T'|} \log_2 p(y_i|x_i) \quad !$$

- This is in fact the normalized log probability of the "test" data:

$$H_{p'}(p) = - 1/|T'| \log_2 \prod_{i = 1..|T'|} p(y_i|x_i)$$

Computation Example

- $\Omega = \{a,b,..,z\}$, prob. distribution (assumed/estimated from data):

  $p(a) = .25$, $p(b) = .5$, $p(\alpha) = 1/64$ for $\alpha \in \{c..r\}$, $= 0$ for the rest: s,t,u,v,w,x,y,z
- Data (test): <u>barb</u> $p'(a) = p'(r) = .25$, $p'(b) = .5$
- Sum over Ω:

α	a b c d e f g ... p q r s t ... z
$-p'(\alpha)\log_2 p(\alpha)$	.5+.5+0+0+0+0+0+0+0+0+0+1.5+0+0+0+0+0 = <u>2.5</u>

- Sum over data:

i / s_i	1/b		2/a		3/r		4/b			1/|T'|
$-\log_2 p(s_i)$	1	+	2	+	6	+	1	= 10		(1/4) × 10 = <u>2.5</u>

Cross Entropy: Some Observations

- H(p) ?? $<, =, >$?? $H_{p'}(p)$: ALL!
- Previous example:

  [$p(a) = .25$, $p(b) = .5$, $p(\alpha) = 1/64$ for $\alpha \in \{c..r\}$, $= 0$ for the rest: s,t,u,v,w,x,y,z]

  H(p) = 2.5 bits = H(p') (<u>barb</u>)
- Other data: <u>probable</u>: (1/8)(6+6+6+1+2+1+6+6)= 4.25

  H(p) < 4.25 bits = H(p') (<u>probable</u>)
- And finally: <u>abba</u>: (1/4)(2+1+1+2)= 1.5

  H(p) > 1.5 bits = H(p') (<u>abba</u>)
- But what about: <u>baby</u> $-p'(\text{'y'})\log_2 p(\text{'y'}) = -.25 \log_2 0 = \infty$ (??)

Cross Entropy: Usage

- Comparing data??
  - *NO!* (we believe that we test on ***real*** data!)
- Rather: comparing distributions (***vs.*** real data)
- Have (got) 2 distributions: p and q (on some Ω, X)
  - which is better?
  - better: has lower cross-entropy (perplexity) on real data S
- "Real" data: S
- $H_S(p) = - 1/|S| \sum_{i = 1..|S|} \log_2 p(y_i|x_i)$?? $H_S(q) = - 1/|S| \sum_{i = 1..|S|} \log_2 q(y_i|x_i)$

Comparing Distributions

Test data S: probable

- p(.) from prev. example: $H_S(p) = 4.25$

  p(a) = .25, p(b) = .5, p(α) = 1/64 for α ∈{c..r}, = 0 for the rest: s,t,u,v,w,x,y,z
- q(.|.) (conditional; defined by a table):

q(.|.)→ ↓	a	b	e	l	o	p	r	other
a	0	.5	0	0	0	.125	0	0
b	1	0	0	0	1	.125	0	0
e	0	0	0	1	0	.125	0	0
l	0	.5	0	0	0	.125	0	0
o	0	0	0	0	0	.125	1	0
p	0	0	0	0	0	.125	0	1
r	0	0	0	0	0	.125	0	0
other	0	0	1	0	0	.125	0	0

ex.: q(o|r) = 1

q(r|p) = .125

(1/8) (log(p|oth.)+log(r|p)+log(o|r)+log(b|o)+log(a|b)+log(b|a)+log(l|b)+log(e|l))

(1/8) (0 + 3 + 0 + 0 + 1 + 0 + 1 + 0)

$H_S(q) = .625$

Language Modeling (and the Noisy Channel)

The Noisy Channel

- Prototypical case:

Input		Output (noisy)
0,1,1,1,0,1,0,1,... →	The channel (adds noise)	→ 0,1,1,<u>0</u>,0,1,<u>1</u>,<u>0</u>,...

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

  known: the noisy output; want to know: the input (***decoding***)

Noisy Channel Applications

- OCR
  - straightforward: text → print (adds noise), scan → 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 → text

Noisy Channel: The Golden Rule of ...

OCR, ASR, HR, MT, ...

- Recall:

  $p(A|B) = p(B|A)\ p(A) / p(B)$ (Bayes formula)

  $A_{best} = \text{argmax}_A\ p(B|A)\ p(A)$ (The Golden Rule)

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

The Perfect Language Model

- Sequence of word forms [forget about tagging for the moment]
- Notation: $A \sim W = (w_1,w_2,w_3,...,w_d)$
- The big (modeling) question:

  $$p(W) = ?$$

- Well, we know (Bayes/chain rule →):

  $$p(W) = p(w_1,w_2,w_3,...,w_d) =$$

  $$= p(w_1) \times p(w_2|w_1) \times p(w_3|w_1,w_2) \times ... \times p(w_d|w_1,w_2,...,w_{d-1})$$

- Not practical (even short W → too many parameters)

Markov Chain

- Unlimited memory (cf. previous foil):
  - for w_i, we know <u>all</u> its predecessors $w_1,w_2,w_3,...,w_{i-1}$
- Limited memory:
  - we disregard “too old” predecessors
  - remember only *k* previous words: $w_{i-k},w_{i-k+1},...,w_{i-1}$
  - called “k^{th} order Markov approximation”
- + stationary character (no change over time):

  $$p(W) \cong \Pi_{i=1..d}\, p(w_i|w_{i-k},w_{i-k+1},...,w_{i-1}),\ d = |W|$$

n-gram Language Models

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

prediction   history

$$p(W) =_{df} \Pi_{i=1..d}\, p(w_i|w_{i-n+1},w_{i-n+2},\ldots,w_{i-1})$$!

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

LM: Observations

- How large *n*?
  - nothing is enough (theoretically)
  - but anyway: as much as possible (→ close to "perfect" model)
  - empirically: ***3***
    - **parameter estimation? (reliability, data availability, storage space, ...)**
    - **4 is too much: $|V|=60k \rightarrow 1.296\times10^{19}$ parameters**
    - **but: 6-7 would be (almost) ideal (having enough data): in fact, one can recover the original text ssequence from 7-grams!**
- Reliability ~ (1 / Detail) (→ need compromise)
- For now, keep word forms (no "linguistic" processing)

The Length Issue

- $\forall n;\ \Sigma_{w\in\Omega^n}\, p(w) = 1 \Rightarrow \Sigma_{n=1..\infty}\Sigma_{w\in\Omega^n}\, p(w) >> 1\ (\rightarrow\infty)$
- We want to model <u>all</u> sequences of words
  - for "fixed" length tasks: no problem - n fixed, sum is 1
    - **tagging, OCR/handwriting (if words identified ahead of time)**
  - for "variable" length tasks: have to account for
    - **discount shorter sentences**
- General model: for each sequence of words of length n, define $p'(w) = \lambda_n p(w)$ such that $\Sigma_{n=1..\infty}\lambda_n = 1 \Rightarrow$

$$\Sigma_{n=1..\infty}\Sigma_{w\in\Omega^n}\, p'(w)=1$$

e.g., estimate λ_n from data; or use normal or other distribution

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 $c_2(y,z) = \Sigma_w c_3(y,z,w)$**
    - **or count differently at the beginning (& end) of data!**

$$\mathbf{p(w_i|w_{i-2},w_{i-1}) =_{est.} c_3(w_{i-2},w_{i-1},w_i) / c_2(w_{i-2},w_{i-1})} \quad !$$

Character Language Model

- Use individual characters instead of words:

$$p(W) =_{df} \Pi_{i=1..d} p(c_i|c_{i-n+1},c_{i-n+2},...,c_{i-1})$$

- 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: $p_2(He|<s>) = 1$, $p_2(can|He) = 1$, $p_2(buy|can) = .5$,
    $p_2(of|can) = .5$, $p_2(the|buy) = 1$,...
  - Trigram: $p_3(He|<s>,<s>) = 1$, $p_3(can|<s>,He) = 1$,
    $p_3(buy|He,can) = 1$, $p_3(of|the,can) = 1$, ..., $p_3(.|of,soda) = 1$.
  - Entropy: $H(p_1) = 2.75$, $H(p_2) = .25$, $H(p_3) = 0$ ← Great?!

LM: an Example (The Problem)

- Cross-entropy:
- S = <s> <s> It was the greatest buy of all.
- Even $H_S(p_1)$ fails ($= H_S(p_2) = H_S(p_3) = \infty$), because:
  - all unigrams but $p_1(the)$, $p_1(buy)$, $p_1(of)$ and $p_1(.)$ are 0.
  - all bigram probabilities are 0.
  - all trigram probabilities are 0.
- We want: to make all (theoretically possible*) probabilities non-zero.

*in fact, all: remember our graph from day 1?

LM Smoothing
(And the EM Algorithm)

The Zero Problem

- "Raw" n-gram language model estimate:
  - necessarily, some zeros
    - **!many: trigram model → 2.16×10^{14} parameters, data ~ 10^9 words**
  - which are true 0?
    - **optimal situation: even the least frequent trigram would be seen several times, in order to distinguish it's probability vs. other trigrams**
    - **optimal situation cannot happen, unfortunately (open question: how many data would we need?)**
  - → we don't know
  - we must eliminate the zeros
- Two kinds of zeros: $p(w|h) = 0$, or even $p(h) = 0$!

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

    $H(p) = \infty$: prevents comparing data with > 0 "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 Ω): almost p(w) but no zeros
- Discount w for (some) $p(w) > 0$: new $p'(w) < p(w)$

  $$\Sigma_{w\in \text{discounted}} (p(w) - p'(w)) = D$$
- 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 $\Sigma_{w\in\Omega} p'(w) = 1$
- There are many ways of ***smoothing***

Smoothing by Adding 1

- Simplest but not really usable:
  - Predicting words w from a vocabulary V, training data T:

    $p'(w|h) = (c(h,w) + 1) \ / \ (c(h) + |V|)$
    - **for non-conditional distributions: $p'(w) = (c(w) + 1) / (|T| + |V|)$**
  - 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 = { what, is, it, small, ?, <s>, flying, birds, are, a, bird, . }, $|V| = 12$**
  - **p(it)=.125, p(what)=.25, p(.)=0 p(what is it?) = $.25^2 \times .125^2 \cong$ .001**

    p(it is flying.) = $.125 \times .25 \times 0^2 = 0$
  - **p'(it) =.1, p'(what) =.15, p'(.)=.05 p'(what is it?) = $.15^2 \times .1^2 \cong$ .0002**

    p'(it is flying.) = $.1 \times .15 \times .05^2 \cong .00004$

Adding *less than* 1

- Equally simple:
  - Predicting words w from a vocabulary V, training data T:

    $p'(w|h) = (c(h,w) + \lambda) \ / \ (c(h) + \lambda|V|), \lambda < 1$
    - **for non-conditional distributions: $p'(w) = (c(w) + \lambda) / (|T| + \lambda|V|)$**
- Example: Training data: <s> what is it what is small ? $|T| = 8$
  - **V = { what, is, it, small, ?, <s>, flying, birds, are, a, bird, . }, $|V| = 12$**
  - **p(it)=.125, p(what)=.25, p(.)=0 p(what is it?) = $.25^2 \times .125^2 \cong$ .001**

    p(it is flying.) = $.125 \times .25 \times 0^2 = 0$
  - **Use $\lambda = .1$:**
  - **p'(it)$\cong$ .12, p'(what)$\cong$ .23, p'(.)$\cong$ .01 p'(what is it?) = $.23^2 \times .12^2 \cong$ .0007**

    p'(it is flying.) = $.12 \times .23 \times .01^2 \cong$ .000003

Good - Turing

- Suitable for estimation from large data
  - similar idea: discount/boost the relative frequency estimate:

    $p_r(w) = (c(w) + 1) \times N(c(w) + 1) / (|T| \times N(c(w)))$,

    where N(c) is the count of words with count c (count-of-counts)

    specifically, for c(w) = 0 (unseen words), $p_r(w) = N(1) / (|T| \times N(0))$
  - good for small counts (< 5-10, where N(c) is high)
  - variants (see MS)
  - normalization! (so that we have $\Sigma_w \, p'(w) = 1$)

Good-Turing: An Example

- Example: remember: $p_r(w) = (c(w) + 1) \times N(c(w) + 1) / (|T| \times N(c(w)))$

  Training data: <s> what is it what is small ? $|T| = 8$
  - **V = { what, is, it, small, ?, <s>, flying, birds, are, a, bird, . }, $|V| = 12$**

    p(it)=.125, p(what)=.25, p(.)=0 p(what is it?) = $.25^2 \times .125^2 \cong$ .001

    p(it is flying.) = $.125 \times .25 \times 0^2 = 0$
  - **Raw reestimation (N(0) = 6, N(1) = 4, N(2) = 2, N(i) = 0 for i > 2):**

    $p_r(it) = (1+1) \times N(1+1)/(8 \times N(1)) = 2 \times 2/(8 \times 4) = .125$

    $p_r(what) = (2+1) \times N(2+1)/(8 \times N(2)) = 3 \times 0/(8 \times 2) = 0$: keep orig. p(what)

    $p_r(.) = (0+1) \times N(0+1)/(8 \times N(0)) = 1 \times 4/(8 \times 6) \cong .083$
  - **Normalize (divide by 1.5 = $\Sigma_{w \in |V|} p_r(w)$) and compute:**

    p'(it)$\cong$ .08, p'(what)$\cong$ .17, p'(.)$\cong$ .06 p'(what is it?) = $.17^2 \times .08^2 \cong$ .0002

    p'(it is flying.) = $.08 \times .17 \times .06^2 \cong$ .00004

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**

→ reliability

← detail

- Simplest possible combination:
  - sum of probabilities, normalize:
    - **p(0|0) = .8, p(1|0) = .2, p(0|1) = 1, p(1|1) = 0, p(0) = .4, p(1) = .6:**
    - **p'(0|0) = .6, p'(1|0) = .4, p'(0|1) = .7, p'(1|1) = .3**

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:

  $\lambda_i > 0$, $\Sigma_{i=0..n}\ \lambda_i = 1$ is sufficient ($\lambda_0 = 1 - \Sigma_{i=1..n}\ \lambda_i$) (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): $-(1/|D|)\Sigma_{i=1..|D|}\log_2(p'_\lambda(w_i|h_i))$

Held-out Data

- What data to use?
  - try the training data T: but we will always get $\lambda_3 = 1$
    - **why? (let p_{iT} be an i-gram distribution estimated using r.f. from T)**
    - **minimizing $H_T(p'_\lambda)$ over a vector λ, $p'_\lambda = \lambda_3 p_{3T}+\lambda_2 p_{2T}+\lambda_1 p_{1T}+\lambda_0/|V|$**
      - remember: $H_T(p'_\lambda) = H(p_{3T})+D(p_{3T}||p'_\lambda)$;
        - (p_{3T} fixed → $H(p_{3T})$ fixed, best)
      - which p'_λ minimizes $H_T(p'_\lambda)$? ... a p'_λ 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 λ!
    - **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!**

The Formulas

- Repeat: minimizing $-(1/|H|)\Sigma_{i=1..|H|}\log_2(p'_\lambda(w_i|h_i))$ over λ

$$p'_\lambda(w_i|\ h_i) = p'_\lambda(w_i|\ w_{i-2},w_{i-1}) = \lambda_3\ p_3(w_i|\ w_{i-2},w_{i-1}) + \lambda_2\ p_2(w_i|\ w_{i-1}) + \lambda_1\ p_1(w_i) + \lambda_0\ /|V|$$!

- "Expected Counts (of lambdas)": j = 0..3

$$c(\lambda_j) = \Sigma_{i=1..|H|}\ (\lambda_j p_j(w_i|h_i)\ /\ p'_\lambda(w_i|h_i))$$!

- "Next λ": j = 0..3

$$\lambda_{j,next} = c(\lambda_j)\ /\ \Sigma_{k=0..3}\ (c(\lambda_k))$$!

The (Smoothing) EM Algorithm

1. Start with some λ, such that $\lambda_j > 0$ for all $j \in 0..3$.
2. Compute "Expected Counts" for each λ_j.
3. Compute new set of λ_j, using the "Next λ" formula.
4. Start over at step 2, unless a termination condition is met.

- Termination condition: convergence of λ.
  - Simply set an ε, and finish if $|\lambda_j - \lambda_{j,next}| < \varepsilon$ for each j (step 3).
- Guaranteed to converge:

  follows from Jensen's inequality, plus a technical proof.

Remark on Linear Interpolation Smoothing

- "Bucketed" smoothing:
  - use several vectors of λ instead of one, based on (the frequency of) history: $\lambda(h)$
    - **e.g. for h = (micrograms,per) we will have**

      λ(h) = (.999,.0009,.00009,.00001)

      (because "cubic" is the only word to follow...)
  - actually: not a separate set for each history, but rather a set for "similar" histories ("bucket"):

    $\lambda(b(h))$, where b: $V^2 \rightarrow N$ (in the case of trigrams)

    b classifies histories according to their reliability (~ frequency)

Bucketed Smoothing: The Algorithm

- First, determine the bucketing function b (use heldout!):
  - decide in advance you want e.g. 1000 buckets
  - compute the total frequency of histories in 1 bucket ($f_{max}(b)$)
  - gradually fill your buckets from the most frequent bigrams so that the sum of frequencies does not exceed $f_{max}(b)$ (you might end up with slightly more than 1000 buckets)
- Divide your heldout data according to buckets
- Apply the previous algorithm to each bucket and its data

Simple Example

- Raw distribution (unigram only; smooth with uniform):

  p(a) = .25, p(b) = .5, $p(\alpha) = 1/64$ for $\alpha \in \{c..r\}$, = 0 for the rest: s,t,u,v,w,x,y,z
- Heldout data: baby; use one set of λ (λ_1: unigram, λ_0: uniform)
- Start with $\lambda_1 = .5$; $p'_\lambda(b) = .5 \times .5 + .5 / 26 = .27$

  $p'_\lambda(a) = .5 \times .25 + .5 / 26 = .14$

  $p'_\lambda(y) = .5 \times 0 + .5 / 26 = .02$

$c(\lambda_1) = .5x.5/.27 + .5x.25/.14 + .5x.5/.27 + .5x0/.02 = 2.72$

$c(\lambda_0) = .5x.04/.27 + .5x.04/.14 + .5x.04/.27 + .5x.04/.02 = 1.28$

Normalize: $\lambda_{1,next} = .68$, $\lambda_{0,next} = .32$.

Repeat from step 2 (recompute p'_λ first for efficient computation, then $c(\lambda_i)$, ...)

Finish when new lambdas almost equal to the old ones (say, < 0.01 difference).

Some More Technical Hints

- Set V = {all words from training data}.
  - **You may also consider $V = T \cup H$, but it does not make the coding in any way simpler (in fact, harder).**
  - **But: you must *never* use the test data for you vocabulary!**
- Prepend two "words" in front of all data:
  - **avoids beginning-of-data problems**
  - **call these index -1 and 0: then the formulas hold exactly**
- When $c_n(w,h) = 0$:
  - **Assign 0 probability to $p_n(w|h)$ where $c_{n-1}(h) > 0$, but a uniform probability (1/|V|) to those $p_n(w|h)$ where $c_{n-1}(h) = 0$ [this must be done both when working on the heldout data during EM, as well as when computing cross-entropy on the test data!]**

Words and the Company They Keep

Motivation

- Environment:
  - mostly "not a full analysis (sentence/text parsing)"
- Tasks where "words & company" are important:
  - word sense disambiguation (MT, IR, TD, IE)
  - lexical entries: subdivision & definitions (lexicography)
  - language modeling (generalization, [kind of] smoothing)
  - word/phrase/term translation (MT, Multilingual IR)
  - NL generation ("natural" phrases) (Generation, MT)
  - parsing (lexically-based selectional preferences)

Collocations

- Collocation
  - Firth: "word is characterized by the company it keeps"; collocations of a given word are statements of the habitual or customary places of that word.
  - non-compositionality of meaning
    - **cannot be derived directly from its parts (heavy rain)**
  - non-substitutability in context
    - **for parts (red light)**
  - non-modifiability (& non-transformability)
    - **kick the ~~yellow~~ bucket; take exception~~s~~ to**

Association and Co-occurence; Terms

- Does not fall under "collocation", but:
- Interesting just because it does often [rarely] appear together or in the same (or similar) context:
    - **(doctors, nurses)**
    - **(hardware,software)**
    - **(gas, fuel)**
    - **(hammer, nail)**
    - **(communism, free speech)**
- Terms:
  - need not be > 1 word (notebook, washer)

Collocations of Special Interest

- Idioms: really fixed phrases
    - **kick the bucket, birds-of-a-feather, run for office**
- Proper names: difficult to recognize even with lists
    - **Tuesday (person's name), May, Winston Churchill, IBM, Inc.**
- Numerical expressions
  - containing "ordinary" words
    - **Monday Oct 04 1999, two thousand seven hundred fifty**
- Phrasal verbs
  - Separable parts:
    - **look up, take off**

Further Notions

- Synonymy: different form/word, same meaning:
    - **notebook / laptop**
- Antonymy: opposite meaning:
    - **new/old, black/white, start/stop**
- Homonymy: same form/word, different meaning:
    - **"true" (random, unrelated): can (aux. verb / can of Coke)**
    - **related: polysemy; notebook, shift, grade, ...**
- Other:
    - **Hyperonymy/Hyponymy: general vs. special: vehicle/car**
    - **Meronymy/Holonymy: whole vs. part: body/leg**

How to Find Collocations?

- Frequency
  - plain
  - filtered
- Hypothesis testing
  - *t* test
  - χ^2 test
- Pointwise ("poor man's") Mutual Information
- (Average) Mutual Information

Frequency

- Simple
  - Count n-grams; high frequency n-grams are candidates:
    - **mostly function words**
    - **frequent names**
- Filtered
  - Stop list: words/forms which (we think) cannot be a part of a collocation
    - **a, the, and, or, but, not, ...**
  - Part of Speech (possible collocation patterns)
    - **A+N, N+N, N+of+N, ...**

Hypothesis Testing

- Hypothesis
  - something we test (against)
- Most often:
  - compare possibly interesting thing vs. "random" chance
  - "Null hypothesis":
    - **something occurs by chance (that's what we suppose).**
    - **Assuming this, prove that the probabilty of the "real world" is then too low (typically < 0.05, also 0.005, 0.001)... therefore reject the null hypothesis (thus confirming "interesting" things are happening!)**
    - **Otherwise, it's possibile there is nothing interesting.**

t test (Student's *t* test)

- Significance of difference
  - compute "magic" number against normal distribution (mean μ)
  - using real-world data: (x' real data mean, s^2 variance, N size):
    - $t = (x' - \mu) / \sqrt{s^2 / N}$
  - find in tables (see MS, p. 609):
    - **d.f. = degrees of freedom (parameters which are not determined by other parameters)**
    - **percentile level p = 0.05 (or better)**
  - the bigger t:
    - **the better chances that there is the interesting feature we hope for (i.e. we can reject the null hypothesis)**
    - **t: at least the value from the table(s)**

t test on words

- null hypothesis: independence
  - **mean μ: $p(w_1)\,p(w_2)$**
- data estimates:
  - **x' = MLE of joint probability from data**
  - **s^2 is p(1-p), i.e. almost p for small p; N is the data size**
- Example: (d.f. ~ sample size)
  - **'general term' (homework corpus): c(general) = 108, c(term) = 40**
  - **c(general,term) = 2; expected p(general)p(term) = 8.8E-8**
  - **$t = (9.0E\text{-}6 - 8.8E\text{-}8) / (9.0E\text{-}6 / 221097)^{1/2} = 1.40$ (not > 2.576) thus 'general term' is <u>not</u> a collocation with confidence 0.005**
  - **'true species': (84/1779/9): $t = 2.774 > 2.576$!!**

Pearson's Chi-square test

- χ^2 test (general formula): $\Sigma_{i,j} (O_{ij}-E_{ij})^2 / E_{ij}$
  - where O_{ij}/E_{ij} is the observed/expected count of events i, j
- for two-outcomes-only events:

$w_{right} \setminus w_{left}$	= true	≠ true
= species	9	1,770
≠ species	75	219,243

$\chi^2 =$ 221097(219243x9-75x1770)^2/1779x84x221013x219318 = 103.39 > 7.88
(at .005 thus we can reject the independence assumption)

Mutual Information and Word Classes

Pointwise Mutual Information

- This is ***NOT*** the MI as defined in Information Theory
  - (IT: average of the following; not of ***values***)
- ...but might be useful:

  $I'(a,b) = \log_2 (p(a,b) / p(a)p(b)) = \log_2 (p(a|b) / p(a))$
- Example (same):

  $I'(true,species) = \log_2 (4.1e\text{-}5 / 3.8e\text{-}4 \times 8.0e\text{-}3) = 3.74$

  $I'(general,term) = \log_2 (9.0e\text{-}6 / 1.8e\text{-}4 \times 4.9e\text{-}4) = 6.68$
  - **measured in bits but it is difficult to give it an interpretation**
  - **used for ranking (≁ the null hypothesis tests)**

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 assigment
  - **What happens to our (bigram) LM?**
    - p(homework | simple) = high probability
    - p(assigment | simple) = low probability (smoothed with p(assigment))
  - They should be much closer!

Word Classes

- Observation: similar words behave in a similar way
  - trigram LM:
  - trigram LM, conditioning:
    - a ... homework (any atribute 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 assigment**
  - 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..n]
    - **$p_k(w_i|h_i) = c(h_i,w_i) / c(h_i)$ [history: (k-1) words]**
  - Introduce classes:

$$\mathbf{p_k(w_i|h_i) = p(w_i|c_i)\ p_k(c_i|h_i)}\ !$$

    - **history: 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 → 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:
  - $p(w_i|c_i) = p(w_i|r(w_i)) = c(w_i) / c(r(w_i)) = c(w_i) / c(c_i)$
    - **!!! $c(w_i,c_i) = c(w_i)$ [since c_i determined by w_i]**
  - $p_k(c_i|h_i)$:
    - **$p_3(c_i|h_i) = p_3(c_i|c_{i-2},c_{i-1}) = c(c_{i-2},c_{i-1},c_i) / c(c_{i-2},c_{i-1})$**
    - **$p_2(c_i|h_i) = p_2(c_i|c_{i-1}) = c(c_{i-1},c_i) / c(c_{i-1})$**
    - **$p_1(c_i|h_i) = p_1(c_i) = c(c_i) / |T|$**
- Then smooth as usual
  - not the $p(w_i|c_i)$ nor $p_k(c_i|h_i)$ individually, but the $\mathbf{p_k(w_i|h_i)}$

Classes: How To Get Them

- We supposed the classes are given
- Maybe there are in [human] dictionaries, but...
  - dictionaries are incomplete
  - dictionaries are unreliable
  - do not define classes as equivalence relation (overlap)
  - do not define classes suitable for LM
    - **small, short... maybe; small and difficult?**
- $\rightarrow$ we have to construct them <u>from data</u> (again...)

Creating the Word-to-Class Map

- We will talk about <u>bigrams</u> from now
- Bigram estimate:
  - **$p_2(c_i|h_i) = p_2(c_i|c_{i-1}) = c(c_{i-1},c_i) / c(c_{i-1}) = c(r(w_{i-1}),r(w_i)) / c(r(w_{i-1}))$**
- Form of the model:
  - just raw bigram for now:
    - **$P(T) = \prod_{i=1..|T|} p(w_i|r(w_i))\, p_2(r(w_i)|r(w_{i-1}))$ $\quad (p_2(c_1|c_0) =_{df} p(c_1))$**
- Maximize over r (given r $\rightarrow$ fixed p, p_2):
  - define objective $L(r) = 1/|T| \Sigma_{i=1..|T|} \log(p(w_i|r(w_i))\, p_2(r(w_i)|r(w_{i-1})))$
  - $r_{best} = \text{argmax}_r L(r)$ (L(r) = norm. logprob of training data... as usual)

Simplifying the Objective Function

- Start from $L(r) = 1/|T| \Sigma_{i=1..|T|} \log(p(w_i|r(w_i))\, p_2(r(w_i)|r(w_{i-1})))$:

  $1/|T| \Sigma_{i=1..|T|} \log(p(w_i|r(w_i))\, \underline{p(r(w_i))}\, p_2(r(w_i)|r(w_{i-1})) / \underline{p(r(w_i))}) =$

  $1/|T| \Sigma_{i=1..|T|} \log(\underline{p(w_i,r(w_i))}\, p_2(r(w_i)|r(w_{i-1})) / p(r(w_i))) =$

  $1/|T| \Sigma_{i=1..|T|} \log(\underline{p(w_i)}) + 1/|T| \Sigma_{i=1..|T|} \log(p_2(r(w_i)|r(w_{i-1})) / p(r(w_i))) =$

  $-H(W) + 1/|T| \Sigma_{i=1..|T|} \log(p_2(r(w_i)|r(w_{i-1}))\, \underline{p(r(w_{i-1})) / (p(r(w_{i-1}))}\, p(r(w_i)))) =$

  $-H(W) + 1/|T| \Sigma_{i=1..|T|} \log(\underline{p(r(w_i),r(w_{i-1}))} / (p(r(w_{i-1}))\, p(r(w_i)))) =$

  $-H(W) + \Sigma_{d,e \in C}\, p(d,e) \log(\, p(d,e) / (p(d)\, p(e))\,) =$

  $-H(W) + I(D,E)$

  (event E picks class adjacent (to the right) to the one picked by D)
- Since W does not depend on r, we ended up with I(D,E).

  the need to maximize

Maximizing Mutual Information

(dependent on the mapping r)

- Result from previous foil:
  - Maximizing the probability of data amounts to maximizing I(D,E), the mutual information of the adjacent classes.
- Good:
  - We know what a MI is, and we know how to maximize.
- Bad:
  - There is no way how to maximize over so many possible partitionings: $|V|^{|V|}$ - no way to test them all.

Training or Heldout?

- Training:
  - best I(D,E): all words in a class of its own
    - → **will not give us anything new.**
- Heldout: ok, but:
  - must smooth to test any possible partitioning (unfeasible):
    - → **using raw model: 0 probability of heldout (almost) guaranteed**
      - → will not be able to compare anything
  - some smoothing estimates? (to be explored...)
- Solution:
  - use training anyway, but only keep I(D,E) as large as possible

The Greedy Algorithm

- Define merging operation on the mapping r: $V \to C$:
  - merge: $R \times C \times C \to R' \times C^{-1}$: $(r,k,l) \to r',C'$ such that
  - $C^{-1} = \{C - \{k,l\} \cup \{m\}\}$ (throw out k and l, add new $m \notin C$)
  - $r'(w) =$ ..... m for $w \in r_{INV}(\{k,l\})$,

    ..... r(w) otherwise.
- 1. Start with each word in its own class (C = V), r = id.
- 2. Merge two classes k,l into one, m, such that

  $$(k,l) = \text{argmax}_{k,l}\, I_{merge(r,k,l)}(D,E).$$
- 3. Set new (r,C) = merge(r,k,l).
- 4. Repeat 2 and 3 until |C| reaches predetermined size.

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]

Word Classes: Programming Tips & Tricks

The Algorithm (review)

- Define merge(r,k,l) = (r',C') such that
  - **C' = C - {k,l} ∪ {m (a new class)}**
  - **r'(w) = r(w) except for k,l member words for which it is m.**
- 1. Start with each word in its own class (C = V), r = id.
- 2. Merge two classes k,l into one, m, such that

$$(k,l) = \text{argmax}_{k,,l}\ I_{merge(r,k,l)}(D,E).$$

- 3. Set new (r,C) = merge(r,k,l).
- 4. Repeat 2 and 3 until |C| reaches a predetermined size.

Complexity Issues

- Still too complex:
  - |V| iterations of the steps 2 and 3.
  - $|V|^2$ steps to maximize $\text{argmax}_{k,l}$ (selecting k,l freely from |C|, which is in the order of $|V|^2$)
  - $|V|^2$ steps to compute I(D,E) (sum within sum, all classes, also: includes log)
  - $\Rightarrow$ total: $|V|^5$
  - i.e., for |V| = 100, about 10^{10} steps; ~ several hours!
  - but |V| ~ 50,000 or more

Trick #1: Recomputing The MI the Smart Way: Subtracting...

- Bigram count table:

l \ r	c_1	c_2	c_3	c_4
c_1	10	2	0	1
c_2	0	0	5	2
c_3	0	2	0	3
c_4	2	3	0	0

- Test-merging c_2 and c_4: recompute only rows/cols 2 & 4:
  - subtract column/row (2 & 4) from the MI sum (intersect.!)
  - add sums of merged counts (row & column)

...and Adding

- Add the merged counts:

l \ r	c_1	c_2’	c_3
c_1	10	3	0
c_2’	2	5	5
c_3	0	5	0

- Be careful at intersections:
  - (don’t forget to add this:)

	c_2	c_3	c_4
c_2	0	5	2
c_3	2	0	3
c_4	3	0	0

Trick #2: Precompute the Counts-to-be-Subtracted

- Summing loop goes through i,j
- ...but the single row/column sums do not depend on the (resulting sums after the) merge
- ⇒ can be precomputed
  - **only 2k logs to compute at each algorithm iteration, instead of k^2**
- Then for each “merge-to-be” compute only add-on sums, plus “intersection adjustment”

Formulas for Tricks #1 and #2

- Let’s have <u>k</u> classes at a certain iteration. Define:

$$q_k(l,r) = p_k(l,r) \log(p_k(l,r) / (p_{kl}(l)\ p_{kr}(r)))$$

now the same, but using counts:

$$q_k(l,r) = c_k(l,r)/N \log(N\ c_k(l,r)/(c_{kl}(l)\ c_{kr}(r)))$$

- Define further (row+column <u>i</u> sum):

precomputed; intersection adjustment

$$s_k(a) = \Sigma_{l=1..k} q_k(l,a) + \Sigma_{r=1..k} q_k(a,r) - q_k(a,a)$$

- Then, the subtraction part of Trick #1 amounts to

$$sub_k(a,b) = s_k(a) + s_k(b) - q_k(a,b) - q_k(b,a)$$

remaining intersect. adj.

Formulas - cont.

- After-merge add-on:

$$add_k(a,b) = \Sigma_{l=1..k, l\neq a,b} q_k(l,a+b) + \Sigma_{r=1..k, r\neq a,b} q_k(a+b,r) + q_k(a+b,a+b)$$

- What is it <u>a+b</u>? Answer: the <u>new (merged) class</u>.
- Hint: use the definition of q_k as a “macro”, and then

$p_k(a+b,r) = p_k(a,r) + p_k(b,r)$ (same for other sums, equivalent)

- The above sums cannot be precomputed
- After-merge Mutual Information (I_k is the “old” MI, kept from previous iteration of the algorithm):

$I_k(a,b)$ (MI after merge of cl. a,b) $= I_k - sub_k(a,b) + add_k(a,b)$

Trick #3: Ignore Zero Counts

- Many bigrams are 0
  - (see the paper: Canadian Hansards, < .1 % of bigrams are non-zero)
- Create linked lists of non-zero counts in columns and rows (similar effect: use perl's hashes)
- Update links after merge (after step 3)

Trick #4: Use Updated Loss of MI

- We are now down to $|V|^4$: $|V|$ merges, each merge takes $|V|^2$ "test-merges", each test-merge involves order-of-$|V|$ operations ($add_k(i,j)$ term, foil #8)
- *Observation*: many numbers (s_k, q_k) needed to compute the mutual information loss due to a merge of i+j ***do not change***: namely, those which are not in the vicinity of neither i nor j.
- *Idea*: keep the MI loss matrix for all pairs of classes, and (after a merge) update only those cells which have been influenced by the merge.

Formulas for Trick #4 (s_{k-1},L_{k-1})

- Keep a matrix of "losses" $L_k(d,e)$.[1]
- Init: $L_k(d,e) = sub_k(d,e) - add_k(d,e)$ [then $I_k(d,e) = I_k - L_k(d,e)$]
- Suppose a,b are now the two classes merged into a:
- Update (k-1: index used for the *next* iteration; $i,j \neq a,b$):
  - $s_{k-1}(i) = s_k(i) - q_k(i,a) - q_k(a,i) - q_k(i,b) - q_k(b,i) + q_{k-1}(a,i) + q_{k-1}(i,a)$
  - [2]$L_{k-1}(i,j) = L_k(i,j) - s_k(i) + s_{k-1}(i) - s_k(j) + s_{k-1}(j) +$
    $+ q_k(i+j,a) + q_k(a,i+j) + q_k(i+j,b) + q_k(b,i+j) -$
    $- q_{k-1}(i+j,a) - q_{k-1}(a,i+j)$ *[NB: may substitute even for s_k , s_{k-1}]*

NB [1] L_k is symmetrical $L_k(d,e) = L_k(e,d)$ (q_k is something different!)

[2]The update formula $L_{k-1}(l,m)$ is wrong in the Brown et. al paper

Completing Trick #4

- $s_{k-1}(a)$ must be computed using the "Init" sum.
- $L_{k-1}(a,i) = L_{k-1}(i,a)$ must be computed in a similar way, for all $i \neq a,b$.
- $s_{k-1}(b)$, $L_{k-1}(b,i)$, $L_{k-1}(i,b)$ are not needed anymore (keep track of such data, i.e. mark every class already merged into some other class and do not use it anymore).
- Keep track of the minimal loss during the $L_k(i,j)$ update process (so that the next merge to be taken is obvious immediately after finishing the update step).

Efficient Implementation

- Data Structures: (N - # of bigrams in data [fixed])
  - Hist(k) history of merges
    - **Hist(k) = (a,b) merged when the remaining number of classes was k**
  - $c_k(i,j)$ bigram class counts [updated]
  - $c_{kl}(i)$, $c_{kr}(i)$ unigram (marginal) counts [updated]
  - $L_k(a,b)$ table of losses; upper-right trianlge [updated]
  - $s_k(a)$ "subtraction" subterms [optionally updated]
  - $q_k(i,j)$ subterms involving a log [opt. updated]
    - **The optionally updated data structures will give linear improvement only in the subsequent steps, but at least $s_k(i)$ is necessary in the initialization phase (1st iteration)**

Implementation: the Initialization Phase

- 1 Read data in, init counts $c_k(l,r)$; then $\forall l,r,a,b;\ a < b$:
- 2 Init unigram counts:

  $c_{kl}(l) = \Sigma_{r=1..k} c_k(l,r),\qquad c_{kr}(r) = \Sigma_{l=1..k} c_k(l,r)$

  - complicated? remember, must take care of start & end of data!
- 3 Init $q_k(l,r)$: use the 2nd formula (count-based) on foil 7,

  $q_k(l,r) = c_k(l,r)/N \log(N\, c_k(l,r)/(c_{kl}(l)\, c_{kr}(r)))$
- 4 Init $s_k(a) = \Sigma_{l=1..k} q_k(l,a) + \Sigma_{r=1..k} q_k(a,r) - q_k(a,a)$
- 5 Init $L_k(a,b) = s_k(a)+s_k(b)-q_k(a,b)-q_k(b,a)-q_k(a+b,a+b)+$

  $- \Sigma_{l=1..k,l\neq a,b} q_k(l,a+b) - \Sigma_{r=1..k,r\neq a,b} q_k(a+b,r)$

Implementation: Select & Update

- 6 Select the best pair (a,b) to merge into a (watch the candidates when computing $L_k(a,b)$); save to Hist(k)
- 7 Optionally, update $q_k(i,j)$ for all $i,j \neq b$, get $q_{k-1}(i,j)$
  - remember those $q_k(i,j)$ values needed for the updates below
- 8 Optionally, update $s_k(i)$ for all $i \neq b$, to get $s_{k-1}(i)$
  - again, remember the $s_k(i)$ values for the "loss table" update
- 9 Update the loss table, $L_k(i,j)$, to $L_{k-1}(i,j)$, using the tabulated q_k, q_{k-1}, s_k and s_{k-1} values, or compute the needed $q_k(i,j)$ and $q_{k-1}(i,j)$ values dynamically from the counts: $c_k(i+j,b) = c_k(i,b) + c_k(j,b)$; $c_{k-1}(a,i) = c_k(a+b,i)$

Towards the Next Iteration

- 10 During the $L_k(i,j)$ update, keep track of the minimal loss of MI, and the two classes which caused it.
- 11 Remember such best merge in Hist(k).
- 12 Get rid of all s_k, q_k, L_k values.
- 13 Set $k = k - 1$; stop if $k == 1$.
- 14 Start the next iteration
  - either by the optional updates (steps 7 and 8), or
  - directly updating $L_k(i,j)$ again (step 9).

Moving Words Around

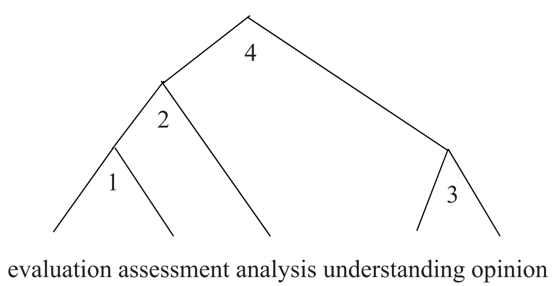
- Improving Mutual Information
  - take a word from one class, move it to another (i.e., two classes change: the moved-from and the moved-to), compute $I_{new}(D,E)$; keep change permanent if

$$I_{new}(D,E) > I(D,E)$$

  - keep moving words until no move improves I(D,E)
- Do it at every iteration, or at every <u>m</u> iterations
- Use similar “smart” methods as for merging

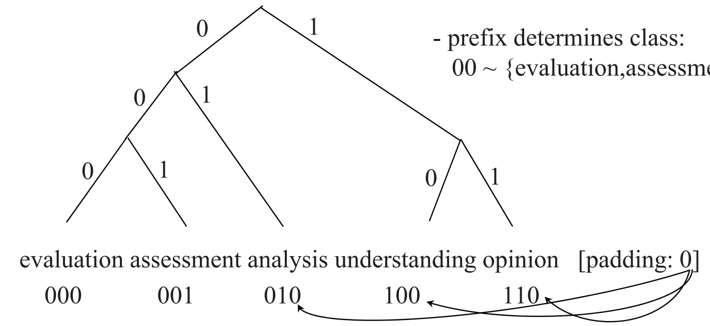
Using the Hierarchy

- Natural Form of Classes
  - follows from the sequence of merges:



Numbering the Classes (within the Hierarchy)

- Binary branching
- Assign 0/1 to the left/right branch at every node:



- prefix determines class:
  00 ~ {evaluation,assessment}

Markov Models

Review: Markov Process

- Bayes formula (chain rule):

$$P(W) = P(w_1,w_2,\ldots,w_T) = \prod_{i=1..T} p(w_i|w_1,w_2,\ldots,w_{i-n+1},\ldots,w_{i-1})$$

(the terms $w_1,w_2,\ldots$ are crossed out; "approximation")

- n-gram language models:
  - Markov process (chain) of the order n-1:

$$P(W) = P(w_1,w_2,\ldots,w_T) = \prod_{i=1..T} p(w_i|w_{i-n+1},w_{i-n+2},\ldots,w_{i-1})$$

Using just one distribution (Ex.: trigram model: $p(w_i|w_{i-2},w_{i-1})$):

Positions: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Words: My car **broke down** , and within hours Bob 's car **broke down** , too .

$p(,|\textbf{broke down}) = p(w_5|w_3,w_4) = p(w_{14}|w_{12},w_{13})$

Markov Properties

- Generalize to any process (not just words/LM):
  - Sequence of random variables: $X = (X_1,X_2,\ldots,X_T)$
  - Sample space S (*states*), size N: $S = \{s_0,s_1,s_2,\ldots,s_N\}$

1. Limited History (Context, Horizon):

$\forall i \in 1..T;\ P(X_i|X_1,\ldots,X_{i-1}) = P(X_i|X_{i-1})$

1 7 3 7 9 0 6 7 [3] 4 5... 1 7 3 7 9 0 6 7 [3] 4 5...

2. Time invariance (M.C. is stationary, homogeneous)

$\forall i \in 1..T, \forall y,x \in S;\ P(X_i=y|X_{i-1}=x) = p(y|x)$

1 [7] [3] [7] [9] 0 6 [7] [3] 4 5...

? ok...same ***distribution***

Long History Possible

- What if we want trigrams:

1 7 3 7 9 0 [6 7] [3] 4 5...

- Formally, use transformation:

Define new variables Q_i, such that $X_i = \{Q_{i-1},Q_i\}$:

Then

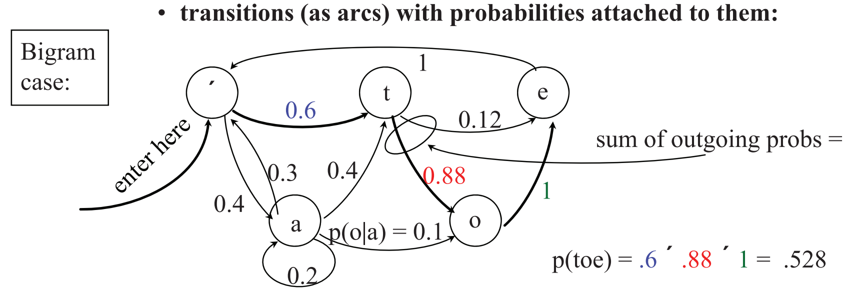
$$P(X_i|X_{i-1}) = P(Q_{i-1},Q_i|Q_{i-2},Q_{i-1}) = P(Q_i|Q_{i-2},Q_{i-1})$$

Predicting (X_i): 1 7 3 7 [9 0] 6 7 3 4 5...

History ($X_{i-1} = \{Q_{i-2},Q_{i-1}\}$): [× ×] [1 ×] [7 1] [3 7] .... [0 9] [6 0] [7 6] [3 7] [4 3]

Graph Representation: State Diagram

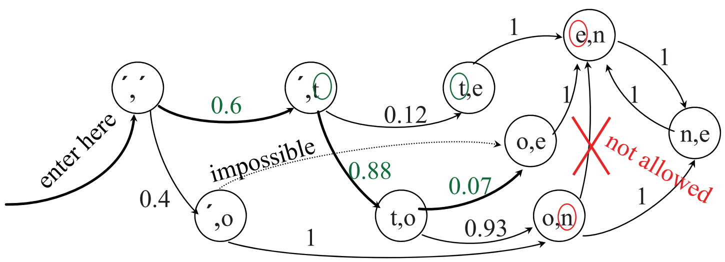
- $S = \{s_0,s_1,s_2,\ldots,s_N\}$: states
- Distribution $P(X_i|X_{i-1})$:
  - **transitions (as arcs) with probabilities attached to them:**



$p(toe) = .6 \times .88 \times 1 = .528$

The Trigram Case

- $S = \{s_0, s_1, s_2, \ldots, s_N\}$: states: pairs $s_i = (x,y)$
- Distribution $P(X_i|X_{i-1})$: (r.v. X: generates pairs s_i)



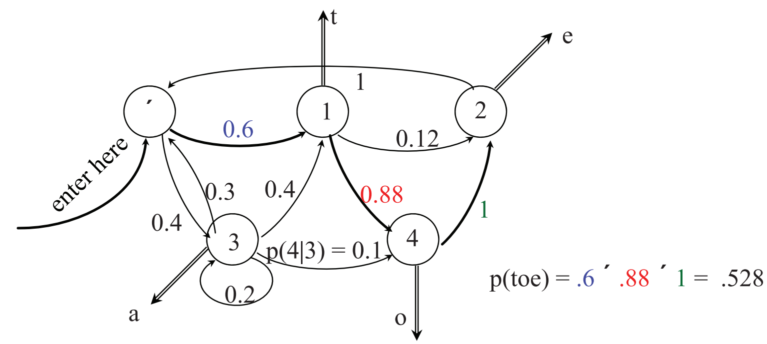
p(toe) = .6 ´ .88 ´ .07 ≅ .037 p(one) = ?

Finite State Automaton

- States ~ symbols of the [input/output] alphabet
  - pairs (or more): last element of the n-tuple
- Arcs ~ transitions (sequence of states)
- [Classical FSA: alphabet symbols on arcs:
  - transformation: arcs ↔ nodes]
- Possible thanks to the "limited history" M'ov Property
- So far: ***Visible*** Markov Models (VMM)

Hidden Markov Models

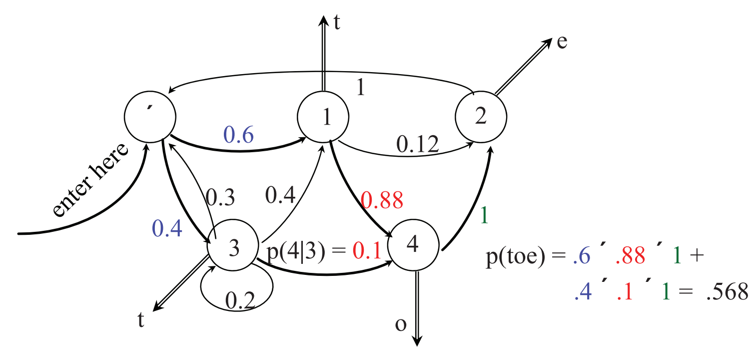
- The simplest HMM: states generate [observable] output (using the "data" alphabet) but remain "invisible":



p(toe) = .6 ´ .88 ´ 1 = .528

Added Flexibility

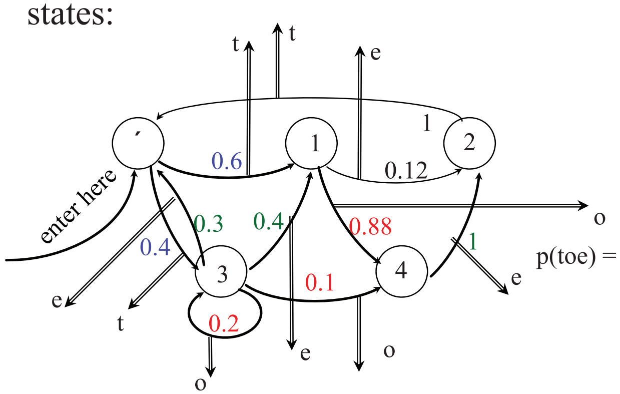
- So far, no change; but different states may generate the same output (why not?):



p(toe) = .6 ´ .88 ´ 1 +
.4 ´ .1 ´ 1 = .568

Output from Arcs...

- Added flexibility: Generate output from arcs, not states:

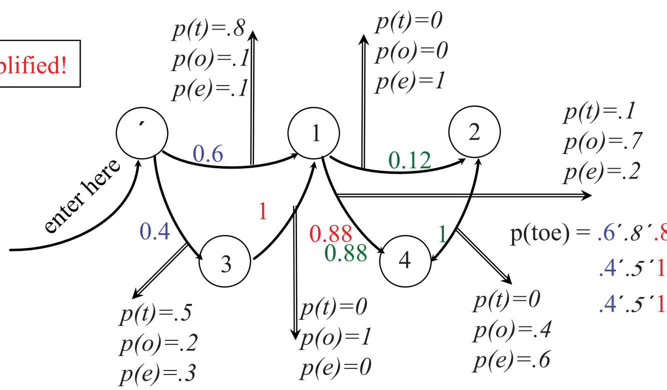


$p(toe) = .6 \times .88 \times 1 + .4 \times .1 \times 1 + .4 \times .2 \times .3 + .4 \times .2 \times .4 = .624$

... and Finally, Add Output Probabilities

- Maximum flexibility: [Unigram] distribution (sample space: output alphabet) at each output arc:

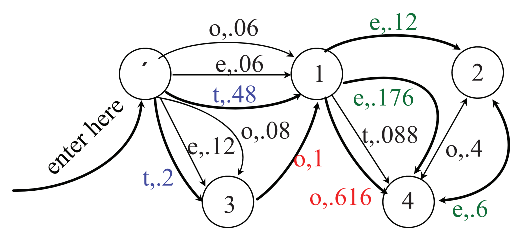
!simplified!



$p(toe) = .6 \times .8 \times .88 \times .7 \times 1 \times .6 + .4 \times .5 \times 1 \times 1 \times .88 \times .2 + .4 \times .5 \times 1 \times 1 \times .12 \times 1 \cong .237$

Slightly Different View

- Allow for multiple arcs from $s_i \rightarrow s_j$, mark them by output symbols, get rid of output distributions:



$p(toe) = .48 \times .616 \times .6 + .2 \times 1 \times .176 + .2 \times 1 \times .12 \cong .237$

In the future, we will use the view more convenient for the problem at hand.

Formalization

- HMM (the most general case):
  - five-tuple (S, s_0, Y, P_S, P_Y), where:
    - **$S = \{s_0, s_1, s_2, \ldots, s_T\}$ is the set of states, s_0 is the initial state,**
    - **$Y = \{y_1, y_2, \ldots, y_V\}$ is the output alphabet,**
    - **$P_S(s_j|s_i)$ is the set of prob. distributions of transitions,**
      - size of P_S: $|S|^2$.
    - **$P_Y(y_k|s_i,s_j)$ is the set of output (emission) probability distributions.**
      - size of P_Y: $|S|^2 \times |Y|$
- Example:
  - $S = \{x, 1, 2, 3, 4\}$, $s_0 = x$
  - $Y = \{ t, o, e \}$

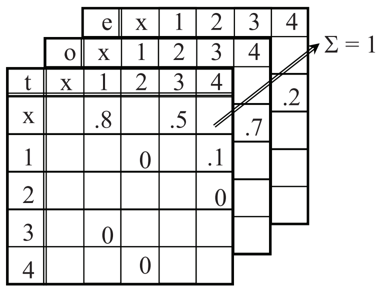
Formalization - Example

- Example (for graph, see foils 11,12):
  - $S = \{x, 1, 2, 3, 4\}$, $s_0 = x$
  - $Y = \{ e, o, t \}$
  - P_S:

	x	1	2	3	4
x	0	.6	0	.4	0
1	0	0	.12	0	.88
2	0	0	0	0	1
3	0	1	0	0	0
4	0	0	1	0	0

$\Sigma = 1$

P_Y:



Using the HMM

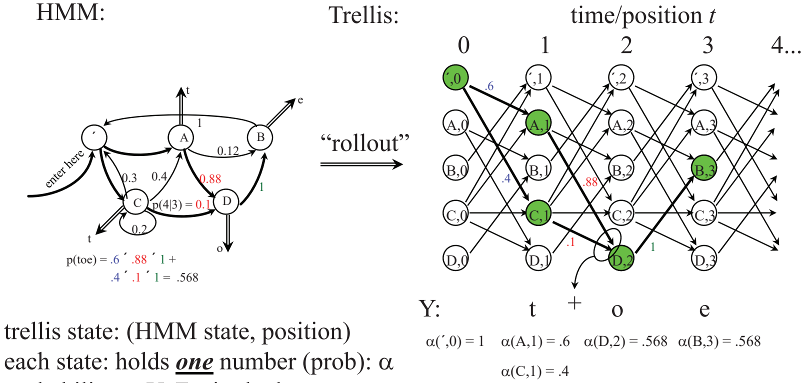
- The generation algorithm (of limited value :-)):
  1. Start in $s = s_0$.
  2. Move from s to s' with probability $P_S(s'|s)$.
  3. Output (emit) symbol y_k with probability $P_S(y_k|s,s')$.
  4. Repeat from step 2 (until somebody says enough).
- More interesting usage:
  - Given an output sequence $Y = \{y_1,y_2,...,y_k\}$, compute its probability.
  - Given an output sequence $Y = \{y_1,y_2,...,y_k\}$, compute the most likely sequence of states which has generated it.
  - ...plus variations: e.g., n best state sequences

HMM Algorithms: Trellis and Viterbi

HMM: The Two Tasks

- HMM (the general case):
  - five-tuple (S, S_0, Y, P_S, P_Y), where:
    - **$S = \{s_1,s_2,...,s_T\}$ is the set of states, S_0 is the initial state,**
    - **$Y = \{y_1,y_2,...,y_V\}$ is the output alphabet,**
    - **$P_S(s_j|s_i)$ is the set of prob. distributions of transitions,**
    - **$P_Y(y_k|s_i,s_j)$ is the set of output (emission) probability distributions.**
- Given an HMM & an output sequence $Y = \{y_1,y_2,...,y_k\}$:
  (Task 1) compute the probability of Y;
  (Task 2) compute the most likely sequence of states which has generated Y.

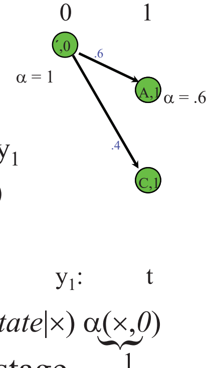
Trellis - Deterministic Output



- trellis state: (HMM state, position)
- each state: holds ***one*** number (prob): α
- probability or Y: Σα in the last state

Creating the Trellis: The Start

- Start in the start state (×),
  - set its α(×,*0*) to 1.
- Create the first stage:
  - get the first "output" symbol y_1
  - create the first stage (column)
  - but only those trellis states which generate y_1
  - set their α(*state*,*1*) to the P_S(*state*|×) α(×,*0*) (where α(×,*0*) = 1)
- ...and forget about the *0*-th stage

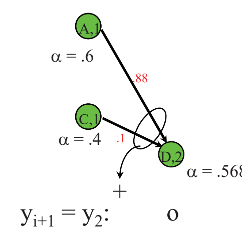


Trellis: The Next Step

- Suppose we are in stage *i*
- Creating the next stage:
  - create all trellis states in the next stage which generate y_{i+1}, but only those reachable from any of the stage-*i* states
  - set their α(*state*,*i+1*) to:

    P_S(*state*|*prev.state*) × α(*prev.state*, *i*)

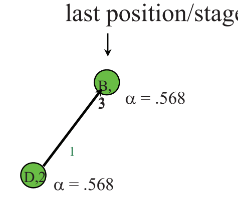
    (add up all such numbers on arcs going to a common trellis state)
  - ...and forget about stage *i*



Trellis: The Last Step

- Continue until "output" exhausted
  - |Y| = 3: until stage 3
- Add together all the α(*state*,|*Y*|)
- That's the <u>P(Y)</u>.
- Observation (pleasant):
  - memory usage max: 2|S|
  - multiplications max: $|S|^2|Y|$

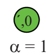
last position/stage

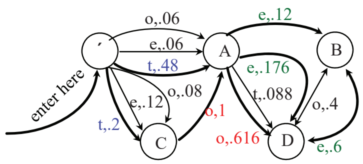


P(Y) = .568

Trellis: The General Case (still, bigrams)

- Start as usual:
  - start state (´), set its α(´,*0*) to 1.

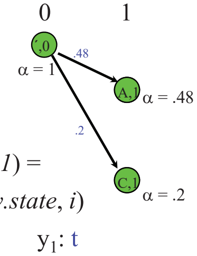


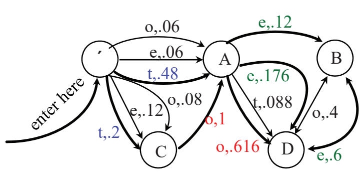


p(toe) = .48´.616´.6+
.2´1´.176 +
.2´1´.12 ≅ .237

General Trellis: The Next Step

- We are in stage *i* :
  - Generate the next stage *i+1* as before (except now arcs generate output, thus use only those arcs marked by the output symbol y_{i+1})
  - For each generated *state*, compute α(*state*,*i+1*) =
  $= \Sigma_{\text{incoming arcs}} P_Y(y_{i+1}|state, prev.state) \times \alpha(prev.state, i)$

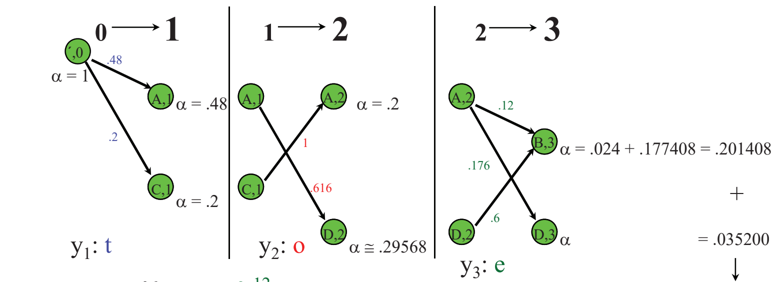


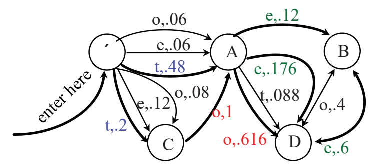


...and forget about stage *i* as usual.

Trellis: The Complete Example

Stage:

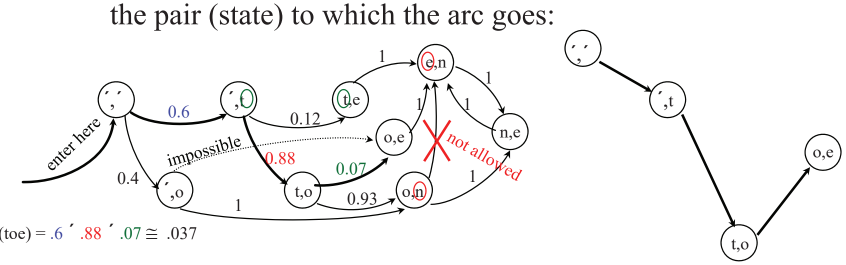




P(Y) = P(toe) = .236608

The Case of Trigrams

- Like before, but:
  - states correspond to bigrams,
  - output function always emits the second output symbol of the pair (state) to which the arc goes:



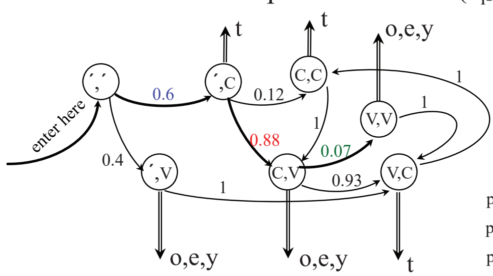
p(toe) = .6 ´ .88 ´ .07 ≅ .037

Multiple paths not possible → trellis not really needed

Trigrams with Classes

- More interesting:
  - n-gram class LM: $p(w_i|w_{i-2},w_{i-1}) = p(w_i|c_i)\ p(c_i|c_{i-2},c_{i-1})$

    $\rightarrow$ states are pairs of classes (c_{i-1},c_i), and emit "words": (letters in our example)



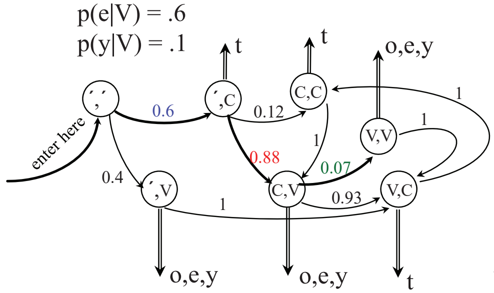
p(t|C) = 1 usual,
p(o|V) = .3 non-
p(e|V) = .6 overlapping
p(y|V) = .1 classes

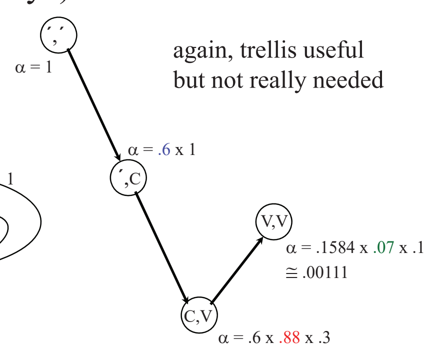
p(toe) = .6 × 1 × .88 × .3 × .07 × .6 ≅ .00665
p(teo) = .6 × 1 × .88 × .6 × .07 × .3 ≅ .00665
p(toy) = .6 × 1 × .88 × .3 × .07 × .1 ≅ .00111
p(tty) = .6 × 1 × .12 × 1 × 1 × .1 ≅ .0072

Class Trigrams: the Trellis

- Trellis generation (Y = "toy"):

p(t|C) = 1
p(o|V) = .3
p(e|V) = .6
p(y|V) = .1

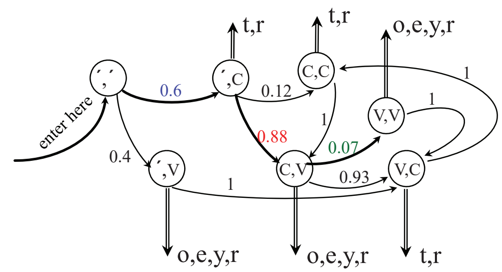




again, trellis useful
but not really needed

Overlapping Classes

- Imagine that classes may overlap
  - e.g. ‘r’ is sometimes vowel sometimes consonant, belongs to V as well as C:

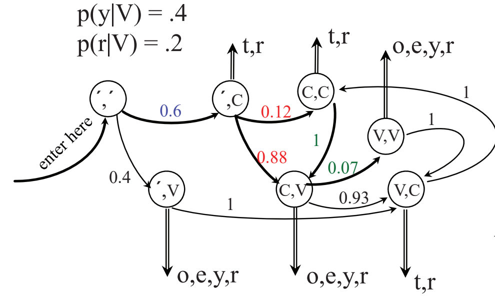


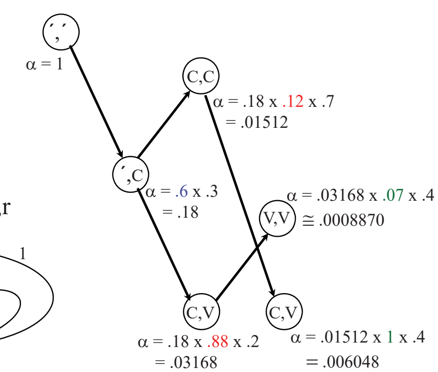
p(t|C) = .3
p(r|C) = .7
p(o|V) = .1
p(e|V) = .3
p(y|V) = .4
p(r|V) = .2

p(try) = ?

Overlapping Classes: Trellis Example

p(t|C) = .3
p(r|C) = .7
p(o|V) = .1
p(e|V) = .3
p(y|V) = .4
p(r|V) = .2





Y: t r y p(Y) = .006935

Trellis: Remarks

- So far, we went left to right (computing α)
- Same result: going right to left (computing β)
  - supposed we know where to start (finite data)
- In fact, we might start in the middle going left and right
- Important for parameter estimation

  (Forward-Backward Algortihm alias Baum-Welch)
- Implementation issues:
  - scaling/normalizing probabilities, to avoid too small numbers & addition problems with many transitions

The Viterbi Algorithm

- Solving the task of finding the most likely sequence of states which generated the observed data
- i.e., finding

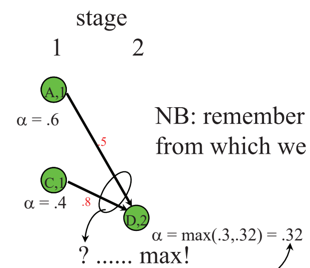
$$S_{best} = \text{argmax}_S P(S|Y)$$

which is equal to (Y is constant and thus P(Y) is fixed):

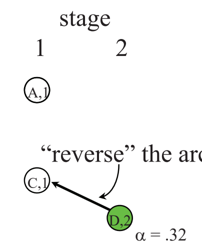
$$S_{best} = \text{argmax}_S P(S,Y) =$$
$$= \text{argmax}_S P(s_0,s_1,s_2,\ldots,s_k,y_1,y_2,\ldots,y_k) =$$
$$= \text{argmax}_S \Pi_{i=1..k}\, p(y_i|s_i,s_{i-1})p(s_i|s_{i-1})$$

The Crucial Observation

- Imagine the trellis build as before (but do not compute the αs yet; assume they are o.k.); stage *i*:



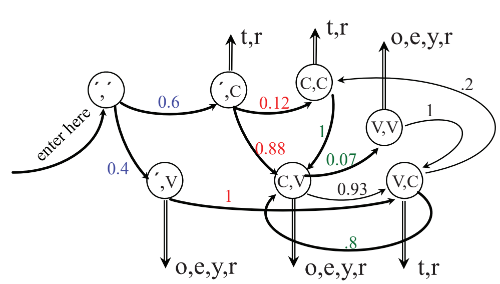
NB: remember previous state from which we got the maximum:



this is certainly the "backwards" maximum to (D,2)... but it cannot change even whenever we go forward (M. Property: Limited History)

Viterbi Example

- 'r' classification (C or V?, sequence?):



p(t|C) = .3
p(r|C) = .7
p(o|V) = .1
p(e|V) = .3
p(y|V) = .4
p(r|V) = .2

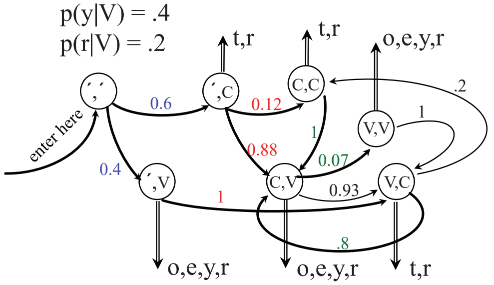
$\text{argmax}_{XYZ}\, p(rry|XYZ) = ?$

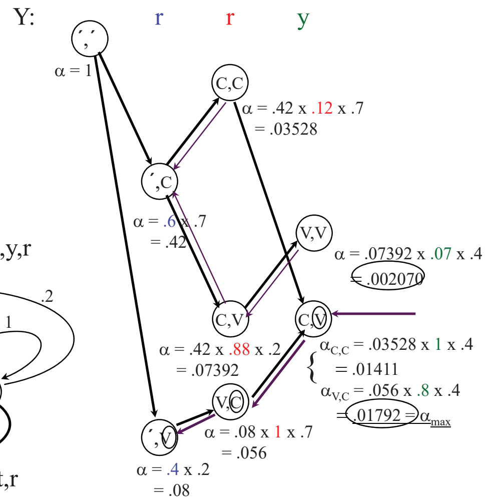
Possible state seq.: (′,v)(v,c)(c,v)[VCV], (′,c)(c,c)(c,v)[CCV], (′,c)(c,v)(v,v) [CVV]

Viterbi Computation

$p(t|C) = .3$
$p(r|C) = .7$
$p(o|V) = .1$
$p(e|V) = .3$
$p(y|V) = .4$
$p(r|V) = .2$

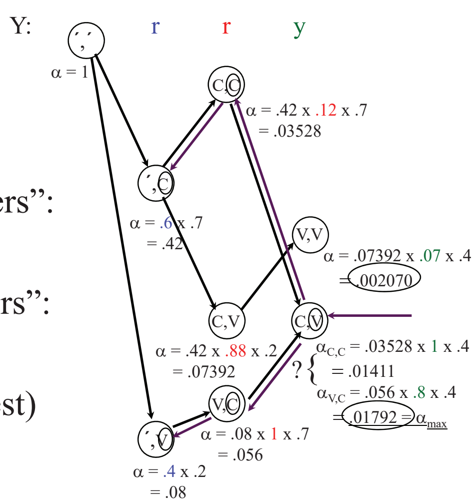
α in trellis state: best prob from start to here





n-best State Sequences

- Keep track of n best "back pointers":
- Ex.: n= 2:
  Two "winners":
  VCV (best)
  CCV (2nd best)

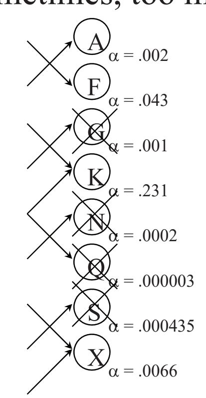


Tracking Back the n-best paths

- Backtracking-style algorithm:
  - **Start at the end, in the best of the n states (s_{best})**
  - **Put the other n-1 best nodes/back pointer pairs on stack, except those leading from s_{best} to the same best-back state.**
- Follow the back "beam" towards the start of the data, spitting out nodes on the way (backwards of course) using always only the best back pointer.
- At every beam split, push the diverging node/back pointer pairs onto the stack (node/beam width is sufficient!).
- When you reach the start of data, close the path, and pop the top-most node/back pointer(width) pair from the stack.
- Repeat until the stack is empty; expand the result tree if necessary.

Pruning

- Sometimes, too many trellis states in a stage:



criteria: (a) α < threshold
(b) Σπ < threshold
(c) # of states > threshold
(get rid of smallest α)

HMM Parameter Estimation: the Baum-Welch Algorithm

HMM: The Tasks

- HMM (the general case):
  - five-tuple (S, S_0, Y, P_S, P_Y), where:
    - **$S = \{s_1,s_2,...,s_T\}$ is the set of states, S_0 is the initial state,**
    - **$Y = \{y_1,y_2,...,y_V\}$ is the output alphabet,**
    - **$P_S(s_j|s_i)$ is the set of prob. distributions of transitions,**
    - **$P_Y(y_k|s_i,s_j)$ is the set of output (emission) probability distributions.**
- Given an HMM & an output sequence $Y = \{y_1,y_2,...,y_k\}$:

  ✓(Task 1) compute the probability of Y;

  ✓(Task 2) compute the most likely sequence of states which has generated Y.

  (Task 3) Estimating the parameters (transition/output distributions)

A Variant of EM

- Idea (~ EM, for another variant see LM smoothing):
  - Start with (possibly random) estimates of P_S and P_Y.
  - Compute (fractional) "counts" of state transitions/emissions taken, from P_S and P_Y, given data Y.
  - Adjust the estimates of P_S and P_Y from these "counts" (using the MLE, i.e. relative frequency as the estimate).
- Remarks:
  - many more parameters than the simple four-way smoothing
  - no proofs here; see Jelinek, Chapter 9

Setting

- HMM (without P_S, P_Y) (S, S_0, Y), and data $T = \{y^i \in Y\}_{i=1..|T|}$
    - **will use T ~ |T|**
  - HMM structure is given: (S, S_0)
  - P_S:Typically, one wants to allow "fully connected" graph
    - **(i.e. no transitions forbidden ~ no transitions set to hard 0)**
    - **why? → we better leave it on the learning phase, based on the data!**
    - **sometimes possible to remove some transitions ahead of time**
  - P_Y: should be restricted (if not, we will not get anywhere!)
    - **restricted ~ hard 0 probabilities of p(y|s,s')**
    - **"Dictionary": states ↔ words, "m:n" mapping on S × Y (in general)**

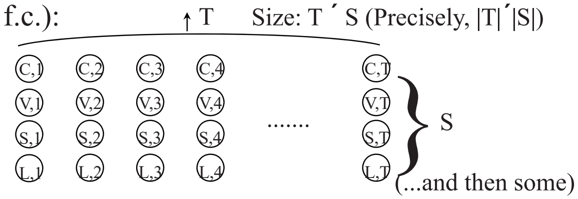
Initialization

- For computing the initial expected "counts"
- Important part
  - EM guaranteed to find a *local* maximum only (albeit a good one in most cases)
- P_Y initialization more important
  - fortunately, often easy to determine
    - **together with dictionary ↔ vocabulary mapping, get counts, then MLE**
- P_S initialization less important
  - e.g. uniform distribution for each p(.|s)

Data Structures

- Will need storage for:
  - The predetermined structure of the HMM
    (unless fully connected → need not to keep it!)
  - The parameters to be estimated (P_S, P_Y)
  - The expected counts (same size as P_S, P_Y)
  - The training data $T = \{y^i \in Y\}_{i=1..T}$
  - The trellis (if f.c.): ↑T Size: T ´ S (Precisely, |T|´|S|)

Each trellis state: ***two*** [float] numbers (forward/backward)



The Algorithm Part I

1. Initialize P_S, P_Y
2. Compute "forward" probabilities:
   - **follow the procedure for trellis (summing), compute α(s,i)**
   - **use the current values of P_S, P_Y (p(s'|s), p(y|s,s')):**

     $$\alpha(s',i) = \sum_{s\to s'} \alpha(s,i-1) \times p(s'|s) \times p(y_i|s,s')$$
   - **NB: do not throw away the previous stage!**
3. Compute "backward" probabilities
   - **start at all nodes of the last stage, proceed backwards, β(s,i)**
   - **i.e., probability of the "tail" of data from stage *i* to the end of data**

     $$\beta(s',i) = \sum_{s\leftarrow s'} \beta(s,i+1) \times p(s|s') \times p(y_{i+1}|s',s)$$
   - **also, keep the β(s,i) at all trellis states**

The Algorithm Part II

4. Collect counts:
   - for each output/transition pair compute

     $$c(y,s,s') = \sum_{i=0..k-1, y=y_{i+1}} \alpha(s,i)\ p(s'|s)\ p(y_{i+1}|s,s')\ \beta(s',i+1)$$

     one pass through data, only stop at (output) y — prefix prob. — this transition prob ´ output prob — tail prob

     $c(s,s') = \sum_{y\in Y} c(y,s,s')$ (assuming all observed y_i in Y)

     $c(s) = \sum_{s'\in S} c(s,s')$
5. Reestimate: $p'(s'|s) = c(s,s')/c(s)$ $p'(y|s,s') = c(y,s,s')/c(s,s')$
6. Repeat 2-5 until desired convergence limit is reached.

Baum-Welch: Tips & Tricks

- Normalization badly needed
  - long training data → extremely small probabilities
- Normalize α,β using the same norm. factor:

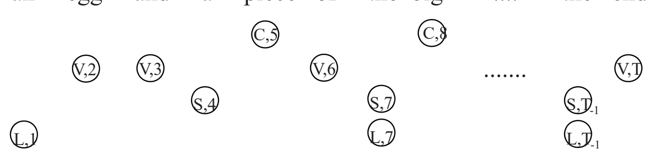
  $N(i) = \Sigma_{s\in S}\ \alpha(s,i)$

  as follows:
  - **compute α(s,i) as usual (Step 2 of the algorithm), computing the sum N(i) at the given stage *i* as you go.**
  - **at the end of each stage, recompute all αs (for each state s):**
  - $\boldsymbol{\alpha^*(s,i) = \alpha(s,i) / N(i)}$
  - **use the same N(i) for βs at the end of each backward (Step 3) stage:**
  - $\boldsymbol{\beta^*(s,i) = \beta(s,i) / N(i)}$

Example

- Task: pronunciation of "the"
- Solution: build HMM, fully connected, 4 states:
  - **S - short article, L - long article, C,V - starting w/consonant, vowel**
  - **thus, only "the" is ambiguous (a, an, the - not members of C,V)**
- Output from states only (p(w|s,s') = p(w|s'))
- Data Y: an egg and a piece of the big .... the end

Trellis:



Example: Initialization

- Output probabilities:

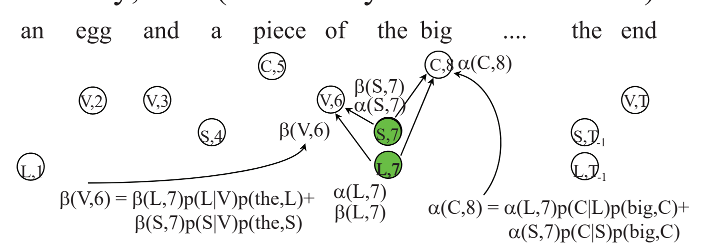
  $p_{init}(w|c) = c(c,w) / c(c)$; where c(S,the) = c(L,the) = c(the)/2

  (other than that, everything is deterministic)
- Transition probabilities:
  - $p_{init}(c'|c) = 1/4$ (uniform)
- Don't forget:
  - about the space needed
  - initialize α(X,0) = 1 (X : the never-occurring front buffer st.)
  - initialize β(s,T) = 1 for all s (except for s = X)

Fill in alpha, beta

- Left to right, alpha:

  $\boldsymbol{\alpha(s',i) = \Sigma_{s\rightarrow s'}\ \alpha(s,i-1) \times p(s'|s) \times p(w_i|s')}$ ← output from states
- Remember normalization (N(i)).
- Similarly, beta (on the way back from the end).

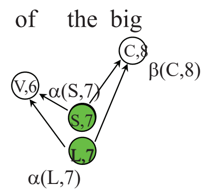


Counts & Reestimation

- One pass through data
- At each position *i*, go through all pairs (s_i,s_{i+1})
- Increment appropriate counters by frac. counts (Step 4):
  - **$inc(y_{i+1},s_i,s_{i+1}) = a(s_i,i)\ p(s_{i+1}|s_i)\ p(y_{i+1}|s_{i+1})\ b(s_{i+1},i+1)$**
  - **$c(y,s_i,s_{i+1})$ += inc (for y at pos i+1)**
  - **$c(s_i,s_{i+1})$ += inc (always)**
  - **$c(s_i)$ += inc (always)**

inc(big,L,C) = $\alpha(L,7)p(C|L)p(big,C)\beta(C,8)$

inc(big,S,C) = $\alpha(S,7)p(C|S)p(big,C)\beta(C,8)$



- Reestimate p(s'|s), p(y|s)
  - **and hope for increase in p(C|S) and p(V|L)...!!**

HMM: Final Remarks

- Parameter "tying":
  - keep certain parameters same (~ just one "counter" for all of them)
  - any combination in principle possible
  - ex.: smoothing (just one set of lambdas)
- Real Numbers Output
  - Y of infinite size (R, R^n):
    - **parametric (typically: few) distribution needed (e.g., "Gaussian")**
- "Empty" transitions: do not generate output
  - **~ vertical arcs in trellis; do not use in "counting"**