Language Modeling (and the Noisy Channel)
The Noisy Channel

- Prototypical case:
  
  \[
  \begin{array}{c}
  \text{Input} \\
  0,1,1,1,0,1,0,1,...
  \end{array}
  \]
  \[
  \begin{array}{c}
  \text{Output (noisy)} \\
  0,1,1,0,0,1,1,0,...
  \end{array}
  \]
  
  (adds noise)

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

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

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

Recall:

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

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

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

- \(p(A)\): the language model

OCR, ASR, HR, MT, …
Probabilistic Language Models

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

Why?
Probabilistic Language Modeling

- **Goal:** compute the probability of a sentence or sequence of words:
  - $P(W) = P(w_1, w_2, w_3, w_4, w_5...w_n)$

- Related task: probability of an upcoming word:
  - $P(w_5 | w_1, w_2, w_3, w_4)$

- A model that computes either of these:
  - $P(W)$ or $P(w_n | w_1, w_2...w_{n-1})$ is called a language model.

- Better: the grammar But language model or LM is standard
The Perfect Language Model

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

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

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

- + stationary character (no change over time):

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

- $(n-1)^{th}$ order Markov approximation → n-gram LM:
  \[
  p(W) = \prod_{i=1}^{d} p(w_i | w_{i-n+1}, w_{i-n+2}, \ldots, w_{i-1})
  \]

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

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

- Reliability ~ (1 / Detail) (→ need compromise) (detail=many gram)

- For now, keep word forms (no “linguistic” processing)
Parameter Estimation

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

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

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

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

- Use individual characters instead of words:

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

- Same formulas etc.
- Might consider 4-grams, 5-grams or even more
- Good only for language comparison
- Transform cross-entropy between letter- and word-based models:

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

- Training data:
  
  `<s> <s> He can buy the can of soda.`

- Unigram: 
  
  \[
  p_1(\text{He}) = p_1(\text{buy}) = p_1(\text{the}) = p_1(\text{of}) = p_1(\text{soda}) = p_1(\cdot) = .125 \\
  p_1(\text{can}) = .25
  \]

- Bigram: 
  
  \[
  p_2(\text{He} | <s>) = 1, p_2(\text{can} | \text{He}) = 1, p_2(\text{buy} | \text{can}) = .5, \\
  p_2(\text{of} | \text{can}) = .5, p_2(\text{the} | \text{buy}) = 1,...
  \]

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

- (normalized for all n-grams) Entropy: 
  
  \[
  H(p_1) = 2.75, \ H(p_2) = .25, \ H(p_3) = 0 \leftarrow \text{Great}?!\]
Language Modeling Toolkits

- SRILM
Google N-Gram Release, August 2006

All Our N-gram are Belong to You

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

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

...?

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

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

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

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

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

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

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

Claude Shannon

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

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

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

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

Chain rule:

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

For bigrams:

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

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

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

\[
PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
= \left( \frac{1}{10} \right)^{-\frac{1}{N}} \\
= \frac{1}{10^{-1}} \\
= \frac{1}{\frac{1}{10}} \\
= 10
\]
Lower perplexity = better model

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

<table>
<thead>
<tr>
<th>N-gram Order</th>
<th>Unigram</th>
<th>Bigram</th>
<th>Trigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perplexity</td>
<td>962</td>
<td>170</td>
<td>109</td>
</tr>
</tbody>
</table>
Unigram
Months the my and issue of year foreign new exchange’s september were recession ex-
change new endorsed a acquire to six executives

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

Trigram
They also point to ninety nine point six billion dollars from two hundred four oh six three
percent of the rates of interest stores as Mexico and Brazil on market conditions
LM: an Example

- Training data:
  
  \[
  \text{He can buy the can of soda.}
  \]

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

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

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

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

- Cross-entropy:
- $S = \langle s \rangle \langle s \rangle$ It was the greatest buy of all. (test data)
- Even $H_S(p_1)$ fails ($= H_S(p_2) = H_S(p_3) = \infty$), because:
  - all unigrams but $p_1$(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 probabilities non-zero. \(\Rightarrow\) data sparseness handling
Why do we need Nonzero Probs?

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

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

- Get new \( p'(w) \) (same \( \Omega \)): almost \( p(w) \) but no zeros
- Discount \( w \) for (some) \( p(w) > 0 \): new \( p'(w) < p(w) \)
  \[
  \sum_{w \in \text{discounted}} (p(w) - p'(w)) = D
  \]
- Distribute \( D \) to all \( w \); \( p(w) = 0 \): new \( p'(w) > p(w) \)
  - possibly also to other \( w \) with low \( p(w) \)
- For some \( w \) (possibly): \( p'(w) = p(w) \)
- Make sure \( \sum_{w \in \Omega} p'(w) = 1 \)
- There are many ways of smoothing
Smoothing by Adding 1 (Laplace)

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

- Example:  
  Training data:  
  \(<s>\) what is it what is small ?  
  \( |T| = 8 \)
  
  \( V = \{ \text{what, is, it, small, ?, <s>, flying, birds, are, a, bird, .} \}, \ |V| = 12 \)
  
  \( p(\text{it})=.125, p(\text{what})=.25, p(.)=0 \quad p(\text{what is it?}) = .25^2 \times .125^2 \approx .001 \)
  
  \( p(\text{it is flying.}) = .125 \times .25 \times 0^2 = 0 \)
  
  \( p'(\text{it}) = .1, p'(\text{what}) = .15, p'(.)=.05 \quad p'(\text{what is it?}) = .15^2 \times .1^2 \approx .0002 \)
  
  \( p'(\text{it is flying.}) = .1 \times .15 \times .05^2 \approx .00004 \)

  (assume word independence!)
Adding less than 1

Equally simple:

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

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

- V = { what, is, it, small, ?, <s>, flying, birds, are, a, bird, . }, |V| = 12
- \[ p(it) = .125, \ p(what) = .25, \ p(.) = 0 \]  \[ p(what \ is \ it?) = .25^2 \times .125^2 \approx .001 \]
  
  \[ p(it \ is \ flying.) = .125 \times .25 \times .02 = 0 \]

- Use \( \lambda = .1 \):

- \[ p'(it) \approx .12, \ p'(what) \approx .23, \ p'(.) \approx .01 \]  \[ p'(what \ is \ it?) = .23^2 \times .12^2 \approx .0007 \]
  
  \[ p'(it \ is \ flying.) = .12 \times .23 \times .01^2 \approx .000003 \]
Backoff and Interpolation

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

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

- Interpolation:
  - Mix unigram, bigram, trigram

- Interpolation works better
Smoothing by Combination: Linear Interpolation

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

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

- Simplest possible combination:
  - sum of probabilities, normalize:
    - 
      \[ p(0|0) = 0.8, \quad p(1|0) = 0.2, \quad p(0|1) = 1, \quad p(1|1) = 0, \quad p(0) = 0.4, \quad p(1) = 0.6 \]
    - 
      \[ p'(0|0) = 0.6, \quad p'(1|0) = 0.4, \quad p'(0|1) = 0.7, \quad p'(1|1) = 0.3 \]
    - 
      \[ p'(0|0) = 0.5p(0|0) + 0.5p(0) \]
Typical n-gram LM Smoothing

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

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

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

What data to use? (to estimate $\lambda$)

- (bad) try the training data $T$: but we will always get $\lambda_3 = 1$
  - why? (let $p_{iT}$ be an i-gram distribution estimated using relative freq. from $T$)
  - minimizing $H_T(p'_{i\lambda})$ over a vector $\lambda$, $p'_{i\lambda} = \lambda_3 p_{3T} + \lambda_2 p_{2T} + \lambda_1 p_{1T} + \lambda_0 / |V|$
    - remember: $H_T(p'_{i\lambda}) = H(p_{3T}) + D(p_{3T} || p'_{i\lambda})$; ($p_{3T}$ fixed $\rightarrow$ $H(p_{3T})$ fixed, best)
    - which $p'_{i\lambda}$ minimizes $H_T(p'_{i\lambda})$? Obviously, a $p'_{i\lambda}$ for which $D(p_{3T} || p'_{i\lambda}) = 0$
    - ...and that’s $p_{3T}$ (because $D(p || p) = 0$, as we know).
    - ...and certainly $p'_{i\lambda} = p_{3T}$ if $\lambda_3 = 1$ (maybe in some other cases, too).
  - ($p'_{i\lambda} = 1 \times p_{3T} + 0 \times p_{2T} + 0 \times p_{1T} + 0 / |V|$)

thus: do not use the training data for estimation of $\lambda$!

- must hold out part of the training data (heldout data, $H$):
  - …call the remaining data the (true/raw) training data, $T$
  - the test data $S$ (e.g., for comparison purposes): still different data!
Markov Models
Review: Markov Process

- Bayes formula (chain rule):

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

- 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.

\[ p(\cdot | \text{broke down}) = p(w_5 | w_3, w_4)) = p(w_{14} | w_{12}, w_{13}) \text{ [stationary]} \]
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...

   1 7 3 7 9 0 6 7

   7

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
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:
  - $S = \{x, 1, 2, 3, 4\}$, $s_0 = x$
  - $Y = \{e, o, t\}$
  - $P_S$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>.6</td>
<td>0</td>
<td>.4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>.12</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- $P_Y$

<table>
<thead>
<tr>
<th></th>
<th>e</th>
<th>x</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>x</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>.8</td>
<td>.5</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>.2</td>
</tr>
</tbody>
</table>

$\Sigma = 1$
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 \(Y = \{y_1,y_2,\ldots,y_k\}\), compute its probability.
  - Given an output sequence \(Y = \{y_1,y_2,\ldots,y_k\}\), 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:
    - \(S = \{s_1, s_2, \ldots, s_T\}\) 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,
    - \(P_Y(y_k | s_i, s_j)\) is the set of output (emission) probability distributions.

- Given an HMM & an output sequence \(Y = \{y_1, y_2, \ldots, y_k\}\):
  - (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): $\alpha$
- probability of $Y$: $\Sigma \alpha$ in the last state
Creating the Trellis: The Start

- Start in the start state \( (x) \),
  - set its \( \alpha(x,0) \) 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 \( \alpha(state,1) \) to the \( P_s(state|x) \ \alpha(x,0) \)
- ...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 $\alpha(state, i+1)$ to:
    $$\sum P_s(state | prev.state) \times \alpha(prev.state, i)$$
    (add up all such numbers on arcs going to a common trellis state)
  - ...and forget about stage $i$
Continue until “output” exhausted
- \[ |Y| = 3: \text{until stage 3} \]
- Add together all the \( \alpha(\text{state}, |Y|) \)
- That’s the \( P(Y) \).
- Observation (pleasant):
  - memory usage max: \( 2|S| \)
  - multiplications max: \( |S|^2|Y| \)
Trellis: The Complete Example

Stage:

\[ \alpha = 1 \]

\[ \alpha = .48 \]

\[ \alpha = .2 \]

\[ \alpha \approx .29568 \]

\[ \alpha = .024 + .177408 = .201408 \]

\[ \alpha \approx .035200 \]

\[ P(Y) = P(toe) = .236608 \]
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:

\[
p(toe) = 0.6 \times 0.88 \times 0.07 \cong 0.037
\]

Multiple paths not possible $\rightarrow$ trellis not really needed
Trigrams with Classes

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

→ 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-overlapping
- \( p(e|V) = .6 \) overlapping
- \( p(y|V) = .1 \) classes

\[
\begin{align*}
p(\text{toe}) &= .6 \times 1 \times .88 \times .3 \times .07 \times .6 \approx .00665 \\
p(\text{teo}) &= .6 \times 1 \times .88 \times .6 \times .07 \times .3 \approx .00665 \\
p(\text{toy}) &= .6 \times 1 \times .88 \times .3 \times .07 \times .1 \approx .00111 \\
p(\text{tty}) &= .6 \times 1 \times .2 \times 1 \times 1 \times .1 \approx .0072
\end{align*}
\]
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
Imagine that classes may overlap

- e.g. ‘r’ is sometimes vowel sometimes consonant, belongs to V as well as C:

\[
\begin{align*}
p(t|C) &= .3 \\
p(r|C) &= .7 \\
p(o|V) &= .1 \\
p(e|V) &= .3 \\
p(y|V) &= .4 \\
p(r|V) &= .2 \\
p(\text{try}) &= ?
\end{align*}
\]
Overlapping Classes: Trellis Example

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

\[ \alpha = 1 \]
\[ \alpha = .18 \times .12 \times .7 \]
\[ = .01512 \]

\[ \alpha = .01512 \times 1 \times .4 \]
\[ = .006048 \]

\[ \alpha = .6 \times .3 \]
\[ = .18 \]

\[ \alpha = .03168 \times .07 \times .4 \]
\[ \approx .0008870 \]

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

\[ \alpha = .18 \times .12 \times .7 \]
\[ = .01512 \]

\[ \alpha = .03168 \times .07 \times .4 \]
\[ \approx .0008870 \]

\[ Y: \quad t \quad r \quad y \quad p(Y) = .006935 \]
Trellis: Remarks

- So far, we went left to right (computing $\alpha$)
- Same result: going right to left (computing $\beta$)
  - 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 Algorithm 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

\[ S_{\text{best}} = \arg\max_S P(S | Y) \]

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

\[ S_{\text{best}} = \arg\max_S P(S, Y) = \]

\[ = \arg\max_S P(s_0, s_1, s_2, \ldots, s_k, y_1, y_2, \ldots, y_k) = \]

\[ = \arg\max_S \prod_{i=1..k} p(y_i | s_i, s_{i-1})p(s_i | s_{i-1}) \]
The Crucial Observation

Imagine the trellis built as before (but use max instead of sum); stage $i$:

- $\alpha = .6$
- $\alpha = .4$
- $\alpha = \max(.3,.32) = .32$

Stage 1: $\alpha = .6$
Stage 2: $\alpha = .32$

NB: remember previous state from which we got the maximum: for every $\alpha$

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?):

Possible state seq.: (x,v)(v,c)(c,v)[VCV], (x,c)(c,c)(c,v)[CCV], (x,c)(c,v)(v,v) [CVV]
Viterbi Computation

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

\[ \alpha = 1 \]
\[ \alpha = .6 \times .7 \]
\[ = .42 \]
\[ \alpha = .42 \times .88 \times .2 \]
\[ = .07392 \]
\[ \alpha = .4 \times .2 \]
\[ = .08 \]

\[ \alpha = .08 \times 1 \times .7 \]
\[ = .056 \]
\[ \alpha_{c,c} = .03528 \times 1 \times .4 \]
\[ = .1411 \]
\[ \alpha_{v,c} = .056 \times .8 \times .4 \]
\[ = .01792 \]

\[ \alpha_{\text{max}} \]

\[ \alpha_{c,c} = .03528 \times 1 \times .4 \]
\[ = .00207 \]

\[ \alpha_{v,c} = .056 \times .8 \times .4 \]
\[ = .01792 \]

\[ \alpha_{\text{max}} \]
**n-best State Sequences**

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

- Sometimes, too many trellis states in a stage:

![Diagram of trellis states with alpha values]

- criteria:  
  (a) $\alpha < \text{threshold}$  
  (b) num of states $> \text{threshold}$  
  (get rid of smallest $\alpha$)
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:
    - \(S = \{s_1, s_2, \ldots, s_T\}\) 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,
    - \(P_Y(y_k|s_i, s_j)\) is the set of output (emission) probability distributions.

- Given an HMM & an output sequence \(Y = \{y_1, y_2, \ldots, y_k\}\):
  - (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: Expectation-Maximization):
  - 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).
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 (e.g. POS tag) ↔ words, “m:n” mapping on \( S \times Y \) (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 $\leftrightarrow$ 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.):

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

  \[
  \begin{array}{cccc}
  C,1 & C,2 & C,3 & C,4 \\
  V,1 & V,2 & V,3 & V,4 \\
  S,1 & S,2 & S,3 & S,4 \\
  L,1 & L,2 & L,3 & L,4 \\
  \end{array}
  \]

  \[\uparrow T \quad \text{Size: } T \times S \text{ (Precisely, } |T| \times |S|)\]

  \[
  \begin{array}{ccc}
  C,T & V,T & S,T \\
  \end{array}
  \]

  \[
  \begin{array}{cccc}
  \text{...} & \text{...} & \text{...} & \text{...} \\
  \end{array}
  \]

  \[\{ \text{S} \}
  \]

  \[\text{...and then some}\]
The Algorithm Part I

1. Initialize $P_S$, $P_Y$

2. Compute “forward” probabilities:
   - Follow the procedure for trellis (summing), compute $\alpha(s,i)$ everywhere
   - Use the current values of $P_S$, $P_Y$ ($p(s'|s)$, $p(y_i|s,s')$):
     \[
     \alpha(s',i) = \sum_{s \rightarrow s'} \alpha(s,i-1) \times p(s'|s) \times p(y_i|s,s')
     \]
   - NB: do not throw away the previous stage!

3. Compute “backward” probabilities
   - Start at all nodes of the last stage, proceed backwards, $\beta(s,i)$
   - 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 $\beta(s,i)$ at all trellis states
The Algorithm Part II

4. Collect counts: (E-step)
   - for each output/transition pair compute
     \[ c(y,s,s') = \sum_{i=0..k-1, y=y} \alpha(s,i) \left[ \frac{p(s' | s) \ p(y_{i+1} | s, s')}}{\beta(s',i+1)} \right] \]

5. Reestimate: (M-step)
   \[ p'(s' | s) = \frac{c(s,s')}{c(s)} \quad p'(y | s, s') = \frac{c(y,s,s')}{c(s,s')} \]

6. Repeat 2-5 until desired convergence limit is reached.
Normalization badly needed
- long training data → extremely small probabilities

Normalize $\alpha, \beta$ using the same norm. factor:

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

as follows:
- compute $\alpha(s, i)$ 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 $\alpha$s (for each state $s$):
  $$\alpha^*(s, i) = \alpha(s, i) / N(i)$$
- use the same $N(i)$ for $\beta$s at the end of each backward (Step 3) stage:
  $$\beta^*(s, i) = \beta(s, i) / N(i)$$
Example

- Task: predict pronunciation of “the”
- Solution: build HMM, fully connected, 4 states:
  - S - short article, L - long article, C,V - word 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:**
  \[ p_{\text{init}}(w|s) = \frac{c(s,w)}{c(s)}; \text{ where } c(S,\text{the}) = c(L,\text{the}) = \frac{c(\text{the})}{2} \]
  (other than that, everything is deterministic)

- **Transition probabilities:**
  - \[ p_{\text{init}}(s'|s) = \frac{1}{4} \text{ (uniform)} \]

- **Don’t forget:**
  - about the space needed
  - initialize \( \alpha(X,0) = 1 \) (\( X \) : the never-occurring front buffer st.)
  - initialize \( \beta(s,T) = 1 \) for all \( s \) (except for \( s = X \))
Fill in alpha, beta

- Left to right, alpha:
  \[
  \alpha(s', i) = \sum_{s ightarrow s'} \alpha(s, i-1) \times p(s' | s) \times p(w_i | s')
  \]

- Remember normalization (N(i)).

- Similarly, beta (on the way back from the end).

\[
\begin{align*}
\beta(V, 6) &= \beta(L, 7)p(L|V)p(\text{the}|L) + \beta(S, 7)p(S|V)p(\text{the}|S) \\
\alpha(V, 8) &= \alpha(L, 7)p(C|L)p(\text{big}|C) + \alpha(S, 7)p(C|S)p(\text{big}|C)
\end{align*}
\]

- an egg and a piece of the big .... the end
Counts & Reestimation

- One pass through data
- At each position $i$, go through all pairs $(s_i, s_{i+1})$
- (E-step) Increment appropriate counters by frac. counts (Step 4):
  - $\text{inc}(y_{i+1}, s_i, s_{i+1}) = a(s_i, i) \ p(s_{i+1} | s_i) \ p(y_{i+1} | s_{i+1}) \ b(s_{i+1}, i+1)$
  - $\text{c}(y, s_i, s_{i+1}) += \text{inc}$ (for $y$ at pos $i+1$)
  - $\text{c}(s_i, s_{i+1}) += \text{inc}$ (always)
  - $\text{c}(s_i) += \text{inc}$ (always)
- (M-step) Reestimate $p(s' | s), p(y | s)$
  - and hope for increase in $p(C | L)$ and $p(\text{the} | L)$...!! (e.g. the coke, the pant)
HMM: Final Remarks

- Parameter “tying”:
  - keep certain parameters same (~ just one “counter” for all of them) – data sparseness
  - 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”
Part-of-speech tagging

A simple but useful form of linguistic analysis
Perhaps starting with Aristotle in the West (384–322 BCE), there was the idea of having parts of speech

- a.k.a lexical categories, word classes, “tags”, POS

It comes from Dionysius Thrax of Alexandria (c. 100 BCE) the idea that is still with us that there are 8 parts of speech

- But actually his 8 aren’t exactly the ones we are taught today

  - Thrax: noun, verb, article, adverb, preposition, conjunction, participle, pronoun

  - School grammar: noun, verb, adjective, adverb, preposition, conjunction, pronoun, interjection
Open class (lexical) words

**Nouns**
- **Proper**
  - IBM
  - Italy
- **Common**
  - cat / cats
  - snow

**Verbs**
- **Main**
  - see
  - registered
- **Modals**
  - can
  - had

**Adjectives**
- old
- older
- oldest

**Adverbs**
- slowly

**Numbers**
- 122,312
- one

**Determiners**
- the
- some

**Conjunctions**
- and
- or

**Pronouns**
- he
- its

Closed class (functional)

**Prepositions**
- to
- with

**Particles**
- off
- up

**Interjections**
- Ow
- Eh
POS Tagging

- Words often have more than one POS: *back*
  - The *back* door = JJ
  - On my *back* = NN
  - Win the voters *back* = RB
  - Promised to *back* the bill = VB

- The POS tagging problem is to determine the POS tag for a particular instance of a word.
POS Tagging

- **Input:** Plays well with others
- **Ambiguity:** NNS/VBZ UH/JJ/NN/RB IN NNS
- **Output:** Plays/VBZ well/RB with/IN others/NNS
- **Uses:**
  - Text-to-speech (how do we pronounce “lead”?)
  - Can write regexps like (Det) Adj* N+ over the output for phrases, etc.
  - As input to or to speed up a full parser
  - If you know the tag, you can back off to it in other tasks
Penn Treebank POS tags - 48 Tags

- **CC** Coordinating conj.  **TO** infinitival to  **CD** Cardinal number  **UH** Interjection  **DT** Determiner  **VB** Verb, base form  **EX** Existential there  
  **VBD** Verb, past tense  **FW** Foreign word  **VBG** Verb, gerund/present  
  **IN** Preposition  **VBN** Verb, past participle  **JJ** Adjective  
  **JJR** Adjective, comparative  **VBP** Verb, non-3rd ps. sg. present  
  **JJS** Adjective, superlative  **WDT** Wh-determiner  **LS** List item marker  
  **WP** Wh-pronoun  **MD** Modal  **WPS** Possessive wh-pronoun  
  **NN** Noun, singular or mass  **WRB** Wh-adverb  **NNS** Noun, plural  
  **NNP** Proper noun, singular  **$** Dollar sign  **NNPS** Proper noun, plural  
  **.** Sentence-final punctuation  **PDT** Predeterminer  **,** Comma  
  **POS** Possessive ending  **:** Colon, semi-colon  **PRP** Personal pronoun  
  **(** Left bracket character  **PPS** Possessive pronoun  **)** Right bracket character  
  **RB** Adverb  ``` Straight double quote  **RBR** Adverb, comparative  
  ``` Left open single quote  **RBS** Adverb, superlative  ``` Left open double quote  
  **RP** Particle  ’ Right close single quote  **SYM** Symbol  ” Right close double quote
How many tags are correct? (Tag accuracy)

- About 97% currently
- But baseline is already 90%
  - Baseline is performance of stupidest possible method
    - Tag every word with its most frequent tag
    - Tag unknown words as nouns
- Partly easy because
  - Many words are unambiguous
  - You get points for them (the, a, etc.) and for punctuation marks!
Deciding on the correct part of speech can be difficult even for people

- Mrs/Shaefer/Shaefer never/never got/got around/around to/to joining/joining

- All/All we/Gotta gotta do/do is/is go/go around/around the/the corner-corner

- Chateau/Petrus/Petrus costs/costs around/around 250/250
How difficult is POS tagging?

- About 11% of the word types in the Brown corpus are ambiguous with regard to part of speech.
- But they tend to be very common words. E.g., *that*
  - I know *that* he is honest = IN
  - Yes, *that* play was nice = DT
  - You can’t go *that* far = RB
- 40% of the word tokens are ambiguous.
HMM Tagging
Recall:

- tagging ~ morphological disambiguation
- tagset $V_T \subseteq (C_1, C_2, \ldots, C_n)$
  - $C_i$ - morphological categories, such as POS, NUMBER, CASE, PERSON, TENSE, GENDER, ...
- mapping $w \rightarrow \{t \in V_T\}$ exists [just word tagging!]
  - restriction of Morphological Analysis: $A^+ \rightarrow 2^{(L, C_1, C_2, \ldots, C_n)}$
    - where $A$ is the language alphabet, $L$ is the set of lemmas
- extension to punctuation, sentence boundaries (treated as words)
The Setting

- Noisy Channel setting:
  - Input (tags)  
  - Output (words)

  The channel
  (adds “noise”)

- Goal (as usual): discover “input” to the channel (\(T\), the tag seq.) given the “output” (\(W\), the word sequence)
  - \(p(T|W) = p(W|T) \cdot p(T) / p(W)\)
  - \(p(W)\) fixed (\(W\) given)...
    \[\arg\max_T p(T|W) = \arg\max_T p(W|T) \cdot p(T)\]
The Model

- Two models \((d = |W| = |T|\) word sequence length):
  - \(p(W|T) = \prod_{i=1..d} p(w_i|w_1,...,w_{i-1},t_1,...,t_d)\)
  - \(p(T) = \prod_{i=1..d} p(t_i|t_1,...,t_{i-1})\)
- Too much parameters (as always)
- Approximation using the following assumptions:
  - tag depends on limited history: \(p(t_i|t_1,...,t_{i-1}) \approx p(t_i|t_{i-n+1},...,t_{i-1})\)
    - n-gram tag “language” model
  - word depends on tag only: \(p(w_i|w_1,...,w_{i-1},t_1,...,t_d) \approx p(w_i|t_i)\)
The HMM Model Definition

- (Almost) the general HMM:
  - output (words) emitted by states (not arcs)
  - states: (n-1)-tuples of tags if n-gram tag model used
  - 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 (the words),
    - \(P_S(s_j | s_i)\) is the set of prob. distributions of transitions
      \[ P_S(s_j | s_i) = p(t_i | t_{i-n+1}, \ldots, t_{i-1}); s_j = (t_{i-n+2}, \ldots, t_i), s_i = (t_{i-n+1}, \ldots, t_{i-1}) \]
    - \(P_Y(y_k | s_i)\) is the set of output (emission) probability distributions
      \[ P_Y(y_k | s_i) = P_Y(y_k | s_j) \text{ if } s_i \text{ and } s_j \text{ contain the same tag as the rightmost element: } P_Y(y_k | s_i) = p(w_i | t_i) \]
Supervised Learning - Training
(Manually Annotated Data Available)

- Use MLE
  - \( p(w_i \mid t_i) = \frac{c_{wt}(t_i, w_i)}{c_t(t_i)} \)
  - \( p(t_i \mid t_{i-n+1}, \ldots, t_{i-1}) = \frac{c_{tn}(t_{i-n+1}, \ldots, t_{i-1}, t_i)}{c_{t(n-1)}(t_{i-n+1}, \ldots, t_{i-1})} \)

- Smooth (both!)
  - \( p(w_i \mid t_i) \): “Add 1” for all possible tag-word pairs using a predefined dictionary (thus some 0 kept!)
  - \( p(t_i \mid t_{i-n+1}, \ldots, t_{i-1}) \): linear interpolation:
    - e.g. for trigram model:
      \[
      p'_\lambda(t_i \mid t_{i-2}, t_{i-1}) = \lambda_3 p(t_i \mid t_{i-2}, t_{i-1}) + \lambda_2 p(t_i \mid t_{i-1}) + \lambda_1 p(t_i) + \lambda_0 / |V_T|
      \]
Unsupervised Learning - Training

- Completely unsupervised learning impossible
  - at least if we have the tagset given - how would we associate words with tags?
- Assumed (minimal) setting:
  - tagset known
  - dictionary/morph. analysis available (providing possible tags for any word)
- Use: Baum-Welch algorithm
  - “tying”: output (state-emitting only, same dist. from two states with same “final” tag)
Comments on Unsupervised Learning

- **Initialization of Baum-Welch**
  - is some annotated data available, use them
  - keep 0 for impossible output probabilities

- **Beware of:**
  - degradation of accuracy (Baum-Welch criterion: entropy, not accuracy!)
  - use heldout data for cross-checking

- **Supervised almost always better**
**Unknown Words**

- “OOV” words (out-of-vocabulary)
  - we do not have list of possible tags for them
  - and we certainly have no output probabilities

- Solutions:
  - try all tags (uniform distribution)
  - try open-class tags (uniform, unigram distribution)
  - try to “guess” possible tags (based on suffix/ending) - use different output distribution based on the ending (and/or other factors, such as capitalization)
Running the Tagger - Inferencing

- Use Viterbi decoding (HMM 2nd tasks)
  - remember to handle unknown words
  - single-best, n-best possible

- Another option:
  - assign always the best tag at each word, but consider all possibilities for previous tags (no back pointers nor a path-backpass)
  - introduces random errors, implausible sequences, but might get higher accuracy (less secondary errors)
(Tagger) Evaluation

- **A must**: Test data (S), previously unseen (in training)
  - change test data often if at all possible! (“feedback cheating”)
  - Error-rate based

- Formally:
  - Out(w) = set of output “items” for an input “item” w
  - True(w) = single correct output (annotation) for w
  - Errors(S) = $\sum_{i=1..|S|} \delta(\text{Out}(w_i) \neq \text{True}(w_i))$
  - Correct(S) = $\sum_{i=1..|S|} \delta(\text{True}(w_i) \in \text{Out}(w_i))$
  - Generated(S) = $\sum_{i=1..|S|} |\text{Out}(w_i)|$
Evaluation Metrics

- **Accuracy**: Single output (tagging: each word gets a single tag)
  - Error rate: $\text{Err}(S) = \frac{\text{Errors}(S)}{|S|}$
  - Accuracy: $\text{Acc}(S) = 1 - \frac{\text{Errors}(S)}{|S|} = 1 - \text{Err}(S)$

- **What if multiple (or no) output?**
  - Recall: $\text{R}(S) = \frac{\text{Correct}(S)}{|S|}$
  - Precision: $\text{P}(S) = \frac{\text{Correct}(S)}{\text{Generated}(S)}$
  - Combination: F measure: $F = \frac{1}{\frac{\alpha}{P} + \frac{(1-\alpha)}{R}}$
    - $\alpha$ is a weight given to precision vs. recall; for $\alpha=.5$, $F = \frac{2PR}{R+P}$