Introduction to
Natural Language Processing (600.465)

HMM Parameter Estimation: the Baum-Welch Algorithm

Dr. Jan Hajič
CS Dept., Johns Hopkins Univ.
hajic@cs.jhu.edu
www.cs.jhu.edu/~hajic

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, 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)

(...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 ($p(s'|s)$, $p(y|s,s')$):

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

$$\alpha^*(s,i) = \alpha(s,i) / N(i)$$

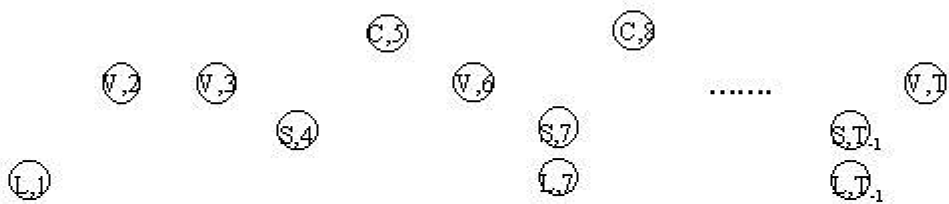
  - **use the same N(i) for βs at the end of each backward (Step 3) stage:**

$$\beta^*(s,i) = \beta(s,i) / N(i)$$

Example

- Task: pronunciation of "the"
- Solution: build HMM, fully connected, 4 states:
  - S - short article, L - long article, C,V - 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:

  $p_{init}(w|c) = c(c,w) / c(c)$; 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 $\alpha(X,0) = 1$ (X : the never-occurring front buffer st.)
  - initialize $\beta(s,T) = 1$ for all s (except for s = X)

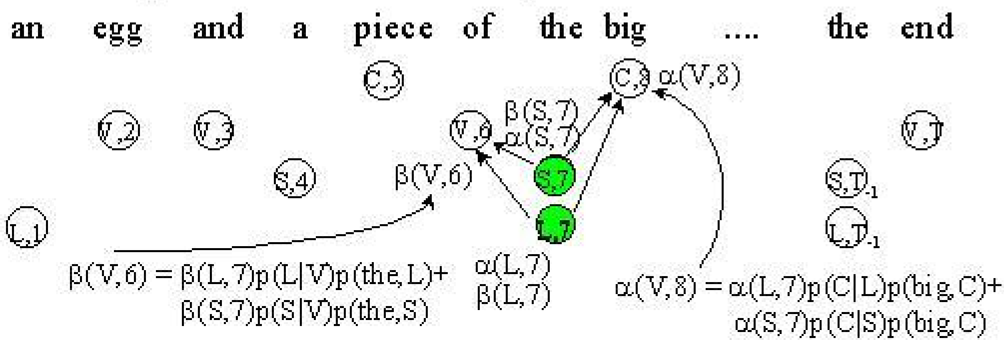
Fill in alpha, beta

- Left to right, alpha:

$$\alpha(s',i) = \sum_{s \to s'} \alpha(s,i\text{-}1) \times p(s'|s) \times p(w_i|s')$$

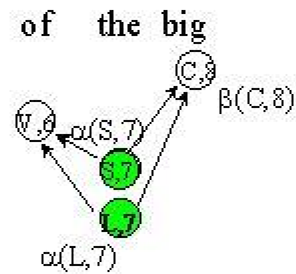
output from states

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

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

- 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"

Introduction to Natural Language Processing (600.465)

HMM Tagging

Dr. Jan Hajič
CS Dept., Johns Hopkins Univ.
hajic@cs.jhu.edu
www.cs.jhu.edu/~hajic

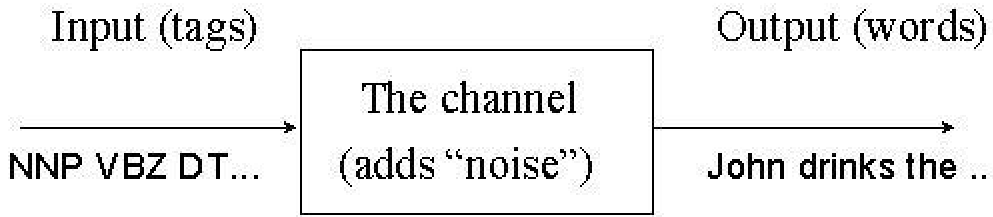
Review

- Recall:
  - tagging ~ morphological disambiguation
  - tagset $V_T \subset (C_1, C_2, \ldots C_n)$
    - C_i - morphological categories, such as POS, NUMBER, CASE, PERSON, TENSE, GENDER, ...
  - mapping $w \rightarrow \{t \in V_T\}$ exists
    - restriction of Morphological Analysis: $A^+ \rightarrow 2^{(L,C1,C2,\ldots,Cn)}$

      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:



- Goal (as usual): discover "input" to the channel (T, the tag seq.) given the "output" (W, the word sequence)
  - $p(T|W) = p(W|T)\, p(T) / p(W)$
  - $p(W)$ fixed (W given)...

$$\mathrm{argmax}_T\, p(T|W) = \mathrm{argmax}_T\, p(W|T)\, p(T)$$

The Model

- Two models (d=|W|=|T| word sequence length):
  - $p(W|T) = \Pi_{i=1..d} p(w_i|w_1,\ldots,w_{i-1},t_1,\ldots,t_d)$
  - $p(T) = \Pi_{i=1..d} p(t_i|t_1,\ldots,t_{i-1})$
- Too much parameters (as always)
- Approximation using the following assumptions:
  - **words do not depend on the context**
  - **tag depends on limited history: $p(t_i|t_1,\ldots,t_{i-1}) \cong p(t_i|t_{i-n+1},\ldots,t_{i-1})$**
    - n-gram tag "language" model
  - **word depends on tag only: $p(w_i|w_1,\ldots,w_{i-1},t_1,\ldots,t_d) \cong p(w_i|t_i)$**

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
      - another simplification: $P_Y(y_k|s_i) = P_Y(y_k|s_j)$ if s_i and s_j contain the same tag as the rightmost element: $P_Y(y_k|s_i) = p(w_i|t_i)$

Supervised Learning (Manually Annotated Data Available)

- Use MLE
  - $p(w_i|t_i) = c_{wt}(t_i,w_i) / c_t(t_i)$
  - $p(t_i|t_{i-n+1},\ldots,t_{i-1}) = c_{tn}(t_{i-n+1},\ldots,t_{i-1},t_i) / c_{t(n-1)}(t_{i-n+1},\ldots,t_{i-1})$
- Smooth (both!)
  - $p(w_i|t_i)$: "Add 1" for all possible tag,word pairs using a predefined dictionary (thus some 0 kept!)
  - $p(t_i|t_{i-n+1},\ldots,t_{i-1})$: linear interpolation:
    - **e.g. for trigram model:**

      $\mathbf{p'_\lambda(t_i|t_{i-2},t_{i-1}) = \lambda_3\, p(t_i|t_{i-2},t_{i-1}) + \lambda_2\, p(t_i|t_{i-1}) + \lambda_1\, p(t_i) + \lambda_0 / |V_T|}$

Unsupervised Learning

- 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 (see class 15, 10/13)
  - "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

- Use Viterbi
  - 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
  - $Errors(S) = \Sigma_{i=1..|S|}\delta(Out(w_i) \neq True(w_i))$
  - $Correct(S) = \Sigma_{i=1..|S|}\delta(True(w_i) \in Out(w_i))$
  - $Generated(S) = \Sigma_{i=1..|S|}|Out(w_i)|$

Evaluation Metrics

- Accuracy: Single output (tagging: each word gets a single tag)
  - Error rate: Err(S) = Errors(S) / |S|
  - Accuracy: Acc(S) = 1 - (Errors(S) / |S|) = 1 - Err(S)
- What if multiple (or no) output?
  - Recall: R(S) = Correct(S) / |S|
  - Precision: P(S) = Correct(S) / Generated(S)
  - Combination: F measure: $F = 1 / (\alpha/P + (1-\alpha)/R)$
    - α is a weight given to precision vs. recall; for α=.5, F = 2PR/(R+P)

Introduction to
Natural Language Processing (600.465)

Transformation-Based Tagging

Dr. Jan Hajič
CS Dept., Johns Hopkins Univ.
hajic@cs.jhu.edu
www.cs.jhu.edu/~hajic

The Task, Again

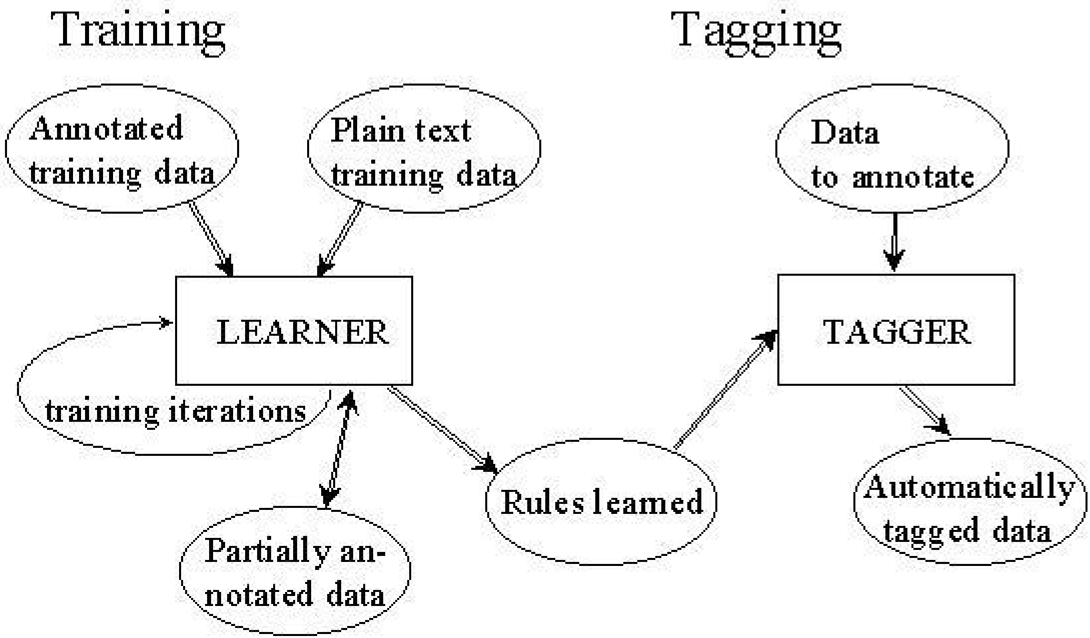
- Recall:
  - tagging ~ morphological disambiguation
  - tagset $V_T \subset (C_1, C_2, \ldots C_n)$
    - C_i - morphological categories, such as POS, NUMBER, CASE, PERSON, TENSE, GENDER, ...
  - mapping $w \rightarrow \{t \in V_T\}$ exists
    - restriction of Morphological Analysis: $A^+ \rightarrow 2^{(L,C1,C2,\ldots,Cn)}$

      where A is the language alphabet, L is the set of lemmas
  - extension to punctuation, sentence boundaries (treated as words)

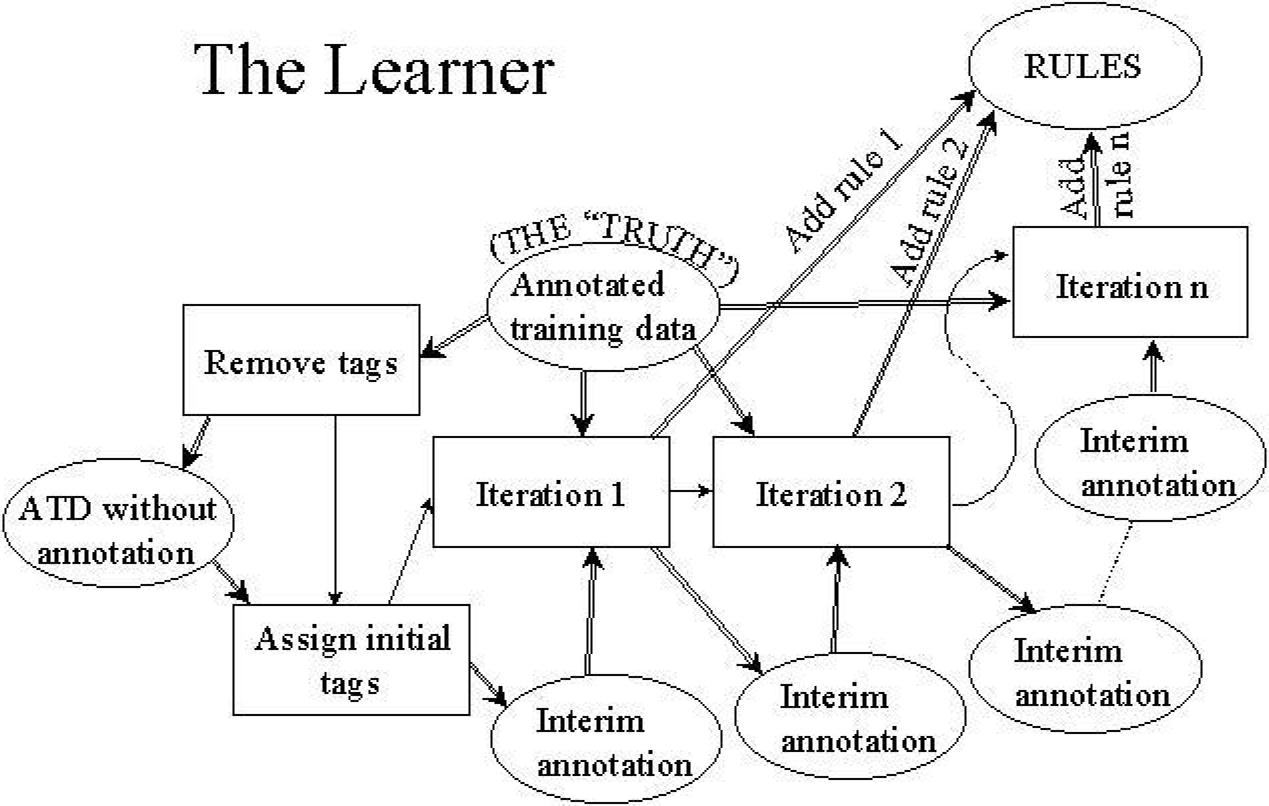
Setting

- ***Not*** a source channel view
- ***Not*** even a probabilistic model (no "numbers" used when tagging a text after a model is developed)
- Statistical, yes:
  - uses training data (combination of supervised [manually annotated data available] and unsupervised [plain text, large volume] training)
  - learning [rules]
  - criterion: accuracy (that's what we are interested in in the end, after all!)

The General Scheme



The Learner


RULES
Add rule 1
Add rule 2
Add rule n
(THE "TRUTH")
Annotated training data
Iteration n
Remove tags
ATD without annotation
Iteration 1
Iteration 2
Interim annotation
Assign initial tags
Interim annotation
Interim annotation
Interim annotation
Interim annotation

The I/O of an Iteration

- In (iteration i):
  - Intermediate data (initial or the result of previous iteration)
  - The TRUTH (the annotated training data)
  - [pool of possible rules]
- Out:
  - One rule $r_{selected(i)}$ to enhance the set of rules learned so far
  - Intermediate data (input data transformed by the rule learned in this iteration, $r_{selected(i)}$)

The Initial Assignment of Tags

- One possibility:
  - NN
- Another:
  - the most frequent tag for a given word form
- Even:
  - use an HMM tagger for the initial assignment
- Not particularly sensitive

The Criterion

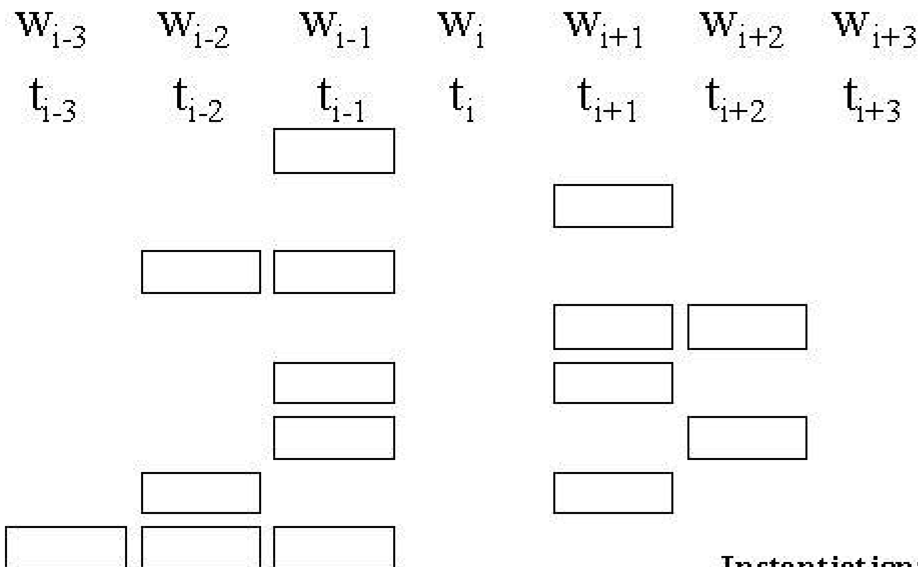
- Error rate (or Accuracy):
  - beginning of an iteration: some error rate E_{in}
  - each possible rule r, when applied at every data position:
    - **makes an improvement somewhere in the data ($c_{improved}(r)$)**
    - **makes it worse at some places ($c_{worsened}(r)$)**
    - **and, of course, does not touch the remaining data**
- Rule contribution to the improvement of the error rate:
    - **$contrib(r) = c_{improved}(r) - c_{worsened}(r)$**
- Rule selection at iteration i:
    - **$r_{selected(i)} = argmax_r\ contrib(r)$**
- New error rate: $E_{out} = E_{in} - contrib(r_{selected(i)})$

The Stopping Criterion

- Obvious:
  - no improvement can be made
    - **contrib(r) ≤ 0**
  - or improvement too small
    - **contrib(r) ≤** Threshold
- NB: prone to overtraining!
  - therefore, setting a reasonable threshold advisable
- Heldout?
  - maybe: remove rules which degrade performance on H

The Pool of Rules (Templates)

- Format: *change tag at position i from* *<u>a</u>* *to* *<u>b</u>* */* *<u>condition</u>*
- Context rules (condition definition - "template"):



Instantiation: w, t permitted

Lexical Rules

- Other type: lexical rules

w_{i-3} w_{i-2} w_{i-1} w_i w_{i+1} w_{i+2} w_{i+3}

t_{i-3} t_{i-2} t_{i-1} t_i t_{i+1} t_{i+2} t_{i+3}

"look inside the word"

- Example:
  - w_i has suffix -ied
  - w_i has prefix ge-

Rule Application

- Two possibilities:
  - immediate consequences (left-to-right):
    - data: DT NN VBP NN VBP NN...
    - rule: NN → NNS / preceded by NN VBP
    - apply rule at position 4:

      DT NN VBP NN VBP NN...

      DT NN VBP NNS VBP NN...
    - ...then rule cannot apply at position 6 (context not NN VBP).
  - delayed ("fixed input"):
    - use original input for context
    - the above rule then applies twice.

In Other Words...

- 1. Strip the tags off the truth, keep the original truth
- 2. Initialize the stripped data by some simple method
- 3. Start with an empty set of selected rules S.
- 4. Repeat until the stopping criterion applies:
  - compute the contribution of the rule r, for each r:

    $$\mathbf{contrib(r) = c_{improved}(r) - c_{worsened}(r)}$$

  - select r which has the biggest contribution contrib(r), add it to the final set of selected rules S.
- 5. Output the set S.

The Tagger

- Input:
  - untagged data
  - rules (S) learned by the learner
- Tagging:
  - use the same initialization as the learner did
  - for i = 1..n (n - the number of rules learnt)
    - apply the rule i to the whole intermediate data, changing (some) tags
  - the last intermediate data is the output.

N-best & Unsupervised Modifications

- N-best modification
  - allow adding tags by rules
  - criterion: optimal combination of accuracy and the number of tags per word (we want: close to ↓1)
- Unsupervised modification
  - use only unambiguous words for evaluation criterion
  - work extremely well for English
  - does not work for languages with few unambiguous words