6.891: Lecture 23 (December 3rd, 2003)

Optimality Theory

Overview

- An introduction to Optimality Theory (OT)
  linguistic examples, formal definitions
- Learning algorithm 1: Constraint Demotion
  [Tesar and Smolensky]
- Learning algorithm 2: [Boersma]
  An application to Finnish
- Learning algorithm 3: [Goldwater and Johnson]
- Open questions

Optimality Theory (OT)

- A "grammar" is a function that maps input forms $x \in \mathcal{X}$ to output forms $y \in \mathcal{Y}$

- OT provides a way of defining a function $F : \mathcal{X} \rightarrow \mathcal{Y}$,
  and a method for defining a "hypothesis space" $\mathcal{H}$ of possible grammars/functions

- Main application of OT is to *phonology*
  (although more recent work has investigated application to syntax)

- The framework was introduced by Prince and Smolensky

Syllable Structure

- Syllables consist of three portions: *onset*, *nucleus*, *coda*
- In general the nucleus has to be a vowel, onsets and codas have to be consonants
- Possible patterns:

$[_\sigma CVC]$
$[_\sigma CV]$
$[_\sigma V]$
$[_\sigma VC]$

An Example from Phonology

- An input form can map to several possible surface forms

$$/VCVC/ \quad \rightarrow \quad [_\sigma V][_\sigma CVC]$$

$$V[_\sigma CV]C$$

$$V[_\sigma CV][_\sigma C\acute{\square}]$$

$$[_\sigma \square V][_\sigma CV]C$$

- Notes:
  - $[_\sigma \ldots]$ is a syllable
  - In some cases, vowels or consonents in the **input** are not seen in the **output** ("underparsing") (e.g., $V[_\sigma CV]C$)
  - In some cases, vowels or consonants in the **output** are not seen in the **input** ("overparsing"): $\square$ and $\acute{\square}$ are "extra" elements inserted in onset or nucleus position

An Example from Phonology

- An example from Maori ([Prince and Smolensky]):

  $/inum/ \rightarrow [_\sigma i][_\sigma nu]m$ (pronounced "inu") is an instance of
  $/VCVC/ \rightarrow [_\sigma V][_\sigma CV]C$

- An example from Maori ([Prince and Smolensky]):

$/VCVC/$	$\rightarrow$	$[_\sigma V][_\sigma CVC]$	$[_\sigma i][_\sigma num]$	"inum"
		$V[_\sigma CV]C$	$i[_\sigma nu]m$	"nu"
		$V[_\sigma CV][_\sigma C\acute{\square}]$	$i[_\sigma nu][_\sigma ma]$	"numa"
		$[_\sigma \square V][_\sigma CV]C$	$[_\sigma mi][_\sigma nu]m$	"minu"

Constraints

- Say $\mathcal{X}$ is set of possible input forms, $\mathcal{Y}$ is set of possible output forms

- Then a **constraint** is a function $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{N}$
  (here $\mathbb{N}$ is the set of non-negative integers)

- $\phi(x, y)$ is the number of times that the input/output pair (x, y) violates the constraint ϕ

Constraint Examples

From [Tesar and Smolensky, 2000]:

$\phi_{\text{ONSET}}(x, y)$ = number of times y has a syllable which has no onset

$\phi_{\text{NOCODA}}(x, y)$ = number of times y has a syllable which has a coda

$\phi_{\text{FILL-NUC}}(x, y)$ = number of nucleus positions in y filled with a vowel that is not in x

$\phi_{\text{PARSE}}(x, y)$ = number of vowels/consonants in x which are not realised in y

$\phi_{\text{FILL-ONS}}(x, y)$ = number of onset positions in y filled with a consonent that is not in x

Candidates	Onset	NoCoda	Fill-Nuc	Parse	Fill-Ons
$/VCVC/ \rightarrow$					
$[_\sigma V][_\sigma CVC]$	1	1	0	0	0
$V[_\sigma CV]C$	0	0	0	2	0
$V[_\sigma CV][_\sigma C\acute{\square}]$	0	0	1	1	0
$[_\sigma \square V][_\sigma CV]C$	0	0	0	1	1

More Formalities

- We take $\mathbf{GEN}$ to be a function that maps an input form x to a set of candidates $\mathbf{GEN}(x) \subset \mathcal{Y}$
- Say we have N constraints, ϕ_i for $i = 1 \ldots N$
- The n-dimensional feature vector for an (x, y) pair is $\mathbf{\Phi}(x, y) = \{\phi_1(x, y), \phi_2(x, y) \ldots \phi_N(x, y)\}$
- Note the similarity to global linear models: in GLM's we'd have a parameter vector $\mathbf{W} \in \mathbb{R}^N$, and define

$$\mathbf{\Phi}(x) = \text{argmax}_{y \in \mathbf{GEN}(x)} \mathbf{\Phi}(x, y) \cdot \mathbf{W}$$

- OT uses an alternative to a parameter vector $\mathbf{W} \in \mathbb{R}^d$: instead, uses a strict **ranking** over the constraints

Constraint Rankings

- A constraint ranking is simply a total order over the constraints

- Notation: U_j for $j = 1 \ldots N$ is the ranking of the j'th constraint. Each $U_j \in \{1 \ldots N\}$. Each $U_j \in \{1 \ldots N\}$, and no two values $j \neq j'$ can have $U_j = U_{j'}$

- Example: say

  $\phi_1 = \phi_{\text{ONSET}}$, $\phi_2 = \phi_{\text{NOCODA}}$, $\phi_3 = \phi_{\text{FILL-NUC}}$,
  $\phi_4 = \phi_{\text{PARSE}}$, $\phi_5 = \phi_{\text{FILL-ONS}}$

  and

  $\{U_1, U_2, U_3, U_4, U_5\} = \{1, 4, 3, 2, 5\}$

  then the ordering on constraints is

  $\phi_1 \gg \phi_4 \gg \phi_3 \gg \phi_2 \gg \phi_5$

More Formalities

- Notation: we use $y_1 \succ y_2$ to mean that y_1 is preferred to y_2

- Define $Cost_i(x, y, \mathbf{U})$ for an input x, and a candidate (x, y), to be

$$Cost_i(x, y, \mathbf{U}) = \sum_{j:U_j=i} \mathbf{\Phi}_j(x, y)$$

  i.e., $Cost_i(x, y, \mathbf{U})$ is the total number of constraints with weight $\mathbf{U}_j = i$ which are violated by (x, y)

- Under these definitions, to choose which of forms y_1 and y_2 are more optimal:
  - Define $d = \min_i\{i : Cost_i(x, y_1, \mathbf{U}) \neq Cost_i(x, y_2, \mathbf{U})\}$
  - **If** $Cost_d(x, y_1) < Cost_d(x, y_2)$ then $y_1 \succ y_2$
  - **If** $Cost_d(x, y_2) > Cost_d(x, y_1)$ then $y_2 \succ y_1$

The basic idea: find the highest ranked constraint on which two output forms differ, and take the output form which violates this constraint the least number of times

Examples

Under

$$\phi_{\text{ONSET}} \gg \phi_{\text{NOCODA}} \gg \phi_{\text{FILL-NUC}} \gg \phi_{\text{PARSE}} \gg \phi_{\text{FILL-ONS}}$$

Candidates	Onset	NoCoda	Fill-Nuc	Parse	Fill-Ons
$/VCVC/ \rightarrow$					
$[_\sigma V][_\sigma CVC]$	1	1	0	0	0
$V[_\sigma CV]C$	0	0	0	2	0
$V[_\sigma CV][_\sigma C\acute{\square}]$	0	0	1	1	0
$[_\sigma \square V][_\sigma CV]C$	0	0	0	1	1

Examples

Under

$$\phi_{\text{ONSET}} \gg \phi_{\text{FILL-ONS}} \gg \phi_{\text{NOCODA}} \gg \phi_{\text{FILL-NUC}} \gg \phi_{\text{PARSE}}$$

Candidates	Onset	NoCoda	Fill-Nuc	Parse	Fill-Ons
$/VCVC/ \rightarrow$					
$[_\sigma V][_\sigma CVC]$	1	1	0	0	0
$V[_\sigma CV]C$	0	0	0	2	0
$V[_\sigma CV][_\sigma C\acute{\square}]$	0	0	1	1	0
$[_\sigma \square V][_\sigma CV]C$	0	0	0	1	1

Summary

- Task is to define a mapping from $\mathcal{X}$ to $\mathcal{Y}$
- Three components to an OT model:
  - $\mathbf{GEN}(x)$ enumerates a set of candidates for x
  - $\mathbf{\Phi}(x, y)$ maps any (x, y) pair in $\mathcal{X} \times \mathcal{Y}$ to a vector in $\mathbb{N}^N$ ($\mathbf{\Phi}(x, y)$ is a vector summarizing the number of violations of each of the N constraints)
  - $\mathbf{U} \in \mathbb{N}^N$ is a ranking over the N constraints ($\mathbf{U}_j$ is the ranking of the j'th constraint)
- $F(x, \mathbf{U})$ is then the member of $\mathbf{GEN}(x)$ which dominates every other member of $\mathbf{GEN}(x)$

Learning in OT

- Task is to learn a mapping from $\mathcal{X}$ to $\mathcal{Y}$
- Three components to an OT model: $\mathbf{GEN}(x)$, $\mathbf{\Phi}(x, y)$, and $\mathbf{U}$.
- $F(x, \mathbf{U})$ is the member of $\mathbf{GEN}(x)$ which dominates every other member of $\mathbf{GEN}(x)$
- Each ranking $\mathbf{U}$ defines a different function $F(x, \mathbf{U})$
- There are $N!$ possible rankings (N is number of constraints)
- Learning $=$ choose a value for $\mathbf{U}$, given some form of supervision

The Constraint Demotion Algorithm

- Assume we have a sequence (x_i, y_i) for $i = 1 \ldots n$
- How do we learn a ranking **W** from these examples?
- First algorithm we'll consider:
  Constraint Demotion [Tesar and Smolensky]

Partial Orders on Constraints

- Algorithm deals with *partial orders* over the contraints
- A partial order is again defined by a vector $\mathbf{W} \in \mathbb{N}^N$
- For example: $\{\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{W}_4, \mathbf{W}_5\} = \{1, 2, 1, 2, 1\}$, represents the partial ordering

$$\{\phi_1, \phi_3, \phi_5\} \gg \{\phi_2, \phi_4\}$$

- In the constraint demotion algorithm, initial partial order is $\mathbf{W} = \bar{1}$, so that partial order is

$$\{\phi_1, \phi_2, \phi_3, \phi_4, \phi_5\}$$

Applying a Partial Order

Our method for deciding between candidates y_1 and y_2 is well-defined under a partial order:

- Define $Cost_i(x, y, \mathbf{U})$ for an input x, and a candidate (x, y), to be

$$Cost_i(x, y, \mathbf{U}) = \sum_{j:\mathbf{U}_j=i} \mathbf{\Phi}_j(x, y)$$

- Under these definitions, to choose which of forms y_1 and y_2 are more optimal:
  - Define $d = \min_i\{i : Cost_i(x, y_1, \mathbf{U}) \neq Cost_i(x, y_2, \mathbf{U})\}$
  - **If** $Cost_d(x, y_1) < Cost_d(x, y_2)$ then $y_1 \succ y_2$
  - **If** $Cost_d(x, y_2) > Cost_d(x, y_1)$ then $y_2 \succ y_1$

An Example

Under

$$\{\phi_{\text{ONSET}}, \phi_{\text{NOCODA}}, \phi_{\text{FILL-NUC}}\} \gg \{\phi_{\text{PARSE}}, \phi_{\text{FILL-ONS}}\}$$

Candidates	Onset	NoCoda	Fill-Nuc	Parse	Fill-Ons
$/VCVC/ \rightarrow$					
$[_\sigma V][_\sigma CVC]$	1	1	0	0	0
$V[_\sigma CV]C$	0	0	0	2	0
$V[_\sigma CV][_\sigma C\acute{\square}]$	0	0	1	1	0
$[_\sigma \square V][_\sigma CV]C$	0	0	0	1	1

The Constraint Demotion Algorithm

Inputs: Training set (x_i, y_i) for $i = 1 \ldots n$

Initialization: $\mathbf{W} = \bar{1}$

Define: $F(x, \mathbf{W}) =$ optimal output for input x under partial order $\mathbf{W}$

Algorithm:

For $i = 1 \ldots n$

$z_i = F(x_i, \mathbf{W})$

If $(z_i \neq y_i)$

- Set $d = \arg\min_{j \in \{j : \phi_j(x, z_i) < \phi_j(x, y_i)\}} \mathbf{W}_j$
- Set $e = \arg\min_{j \in \{j : \phi_j(x, z_i) > \phi_j(x, y_i)\}} \mathbf{W}_j$ (Note that $\mathbf{W}_e \geq \mathbf{W}_d$)
- Set $\mathbf{W}_d = \mathbf{W}_e + 1$

Note: d is the highest ranked constraint which prefers z_i over y_i

e is the highest ranked constraint which prefers y_i over z_i

Output: Partial order $\mathbf{W}$

A Theorem Underlying the Constraint Demotion Algorithm

Theorem: (Equivalent to results in Tesar and Smolensky). For any sequence (x_i, y_i) for $i = 1 \ldots n$, for any **GEN**, **Φ** combination where **Φ** involves N constraints, if there exists some total order **U** such that $F(x_i, \mathbf{U}) = y_i$ for all $i = 1 \ldots n$, then the constraint demotion algorithm makes at most $N(N-1)/2$ mistakes on the sequence.

A Proof

- Define $\mathbf{W}^k$ to be the partial order after the k'th mistake has been made
  From initialization: $\mathbf{W}^0 = \bar{1}$

- **First property:** for all k, $\quad k + N \leq \sum_{j=1}^{N} \mathbf{W}_j^k$
  Follows by induction. True for $k = 0$. Each mistake implies that one component of $\mathbf{W}$ is increased by at least a value of 1, so the property is maintained

- **Second property:** for all j, k, $\mathbf{W}_j^k \leq \mathbf{U}_j$
  We'll prove this shortly

- **Theorem then follows easily:**

  for all k, $\quad \sum_j \mathbf{W}_j^k \leq \sum_j \mathbf{U}_j = N(N+1)/2$

  $\Rightarrow$ for all k, $\quad k + N \leq \sum_{j=1}^{N} \mathbf{W}_j^k \leq N(N+1)/2$

  $\Rightarrow$ for all k, $\quad k \leq N(N-1)/2$

Proof of the Second Property

- Proof by induction on k:
  true for the base case of $k = 0$, as $\mathbf{W}_j^0 = 1$ for all j, and $\mathbf{U}_j \geq 1$ for all j

- Inductive case: assume $\mathbf{W}_j^k \leq \mathbf{U}_j$ for all j, and prove that the demotion algorithm gives $\mathbf{W}_j^{k+1} \leq \mathbf{U}_j$ for all j

- **The demotion step, plus one additional definition:**

  $d = \arg\min_{j \in \{j : \phi_j(x, z_i) < \phi_j(x, y_i)\}} \mathbf{W}_j^k$
  $e = \arg\min_{j \in \{j : \phi_j(x, z_i) > \phi_j(x, y_i)\}} \mathbf{W}_j^k$ (Note that $\mathbf{W}_e^k \geq \mathbf{W}_d^k$)
  $f = \arg\min_{j \in \{j : \phi_j(x, z_i) > \phi_j(x, y_i)\}} \mathbf{U}_j$
  Set $\mathbf{W}_d^{k+1} = \mathbf{W}_e^k + 1$
  Set $\mathbf{W}_j^{k+1} = \mathbf{W}_j^k$ for all $j \neq d$

  Note: d is the highest ranked constraint under $\mathbf{W}$ which prefers z_i over y_i
  e is the highest ranked constraint under $\mathbf{W}$ which prefers y_i over z_i
  f is the highest ranked constraint under $\mathbf{U}$ which prefers y_i over z_i

- We then have

$$\mathbf{W}_e^k \leq \mathbf{W}_f^k \leq \mathbf{U}_f < \mathbf{U}_d$$

hence

$$\mathbf{W}_d^{k+1} = \mathbf{W}_e^k + 1 \leq \mathbf{U}_d$$

hence the property $\mathbf{W}_d^{k+1} \leq \mathbf{U}_d$ holds for the constraint d, and $\mathbf{W}_j^{k+1} = \mathbf{W}_j^k$ for $j \neq d$ means that $\mathbf{W}_j^{k+1} \leq \mathbf{U}_j$ for all j by the inductive hypothesis.

- The inequalities hold because of the following arguments:
  - $\mathbf{W}_e^k \leq \mathbf{W}_f^k$ because we must have $\phi_f(x, z_i) > \phi_f(x, y_i)$ for f to be $\arg\min_{j \in \{j : \phi_j(x,z_i) > \phi_j(x,y_i)\}} \mathbf{U}_j$, and e is defined as $\arg\min_{j \in \{j : \phi_j(x,z_i) > \phi_j(x,y_i)\}} \mathbf{W}_j^k$
  - $\mathbf{W}_f^k \leq \mathbf{U}_f$ by the inductive hypothesis
  - $\mathbf{U}_f < \mathbf{U}_d$ because $\mathbf{U}$ defines a total order, and if $\mathbf{U}_d > \mathbf{U}_f$ then constraint d would be chosen ahead of f

The (Minimal) Gradual Learning Algorithm

Inputs: Training set (x_i, y_i) for $i = 1 \ldots n$

Initialization: $\mathbf{W} = \bar{1}$

Define: $F(x, \mathbf{W})$ = optimal output for input x under partial order $\mathbf{W}$

Algorithm:

For $i = 1 \ldots n$

$z_i = F(x_i, \mathbf{W})$

If $(z_i \neq y_i)$

- Set $d = \arg\min_{j \in \{j : \phi_j(x, z_i) < \phi_j(x, y_i)\}} \mathbf{W}_j$
- Set $\mathbf{W}_d = \mathbf{W}_d + 1$

Note: d is the highest ranked constraint which prefers z_i over y_i

Output: Partial order $\mathbf{W}$

This algorithm also makes at most $N(N-1)/2$ mistakes, through a very similar proof to that for constraint demotion

An Example from Finnish

- [Boersma and Hayes, 1999]:
  Empirical tests of the gradual learning algorithm.

- Finnish genitive plurals can be formed in two ways:
  with a weak ending (usually /-jen/) or a strong ending (typically /-iden/)

- In some cases, *both* endings are possible, and are seen in corpora
  - e.g., *naapuri* (neighbour) $\Rightarrow$ $náapurien$, or $náapurèiden$

- Questions:
  - How can we build models that handle free variation?
  - How can we learn these models from data?

Stochastic Optimality Theory [Boersma]

- Before, we had a deterministic method for calculating $F(x, \mathbf{W})$

- A stochastic method:
  - Randomly draw a new vector $\mathbf{W}'$, where each $\mathbf{W}'_j = \mathbf{W}_j +$ *gaussian noise with variance* σ
  - Output is then $F(x, \mathbf{W}')$, where F is as defined before (a deterministic function of x and $\mathbf{W}'$

  We'll call this method $F_\sigma(x, \mathbf{W})$

- Intuition: if two values $\mathbf{W}_j$ and $\mathbf{W}_{j'}$ are close together in value (i.e., close w.r.t. σ), then there is some chance that they will be switched

- Motivation: *free variation*. Sometimes more than one surface form is possible for a given input, and different frequencies are observed in data

The Maximal Gradual Learning Algorithm

Inputs: Training set (x_i, y_i) for $i = 1 \ldots n$
Gaussian variance $\sigma > 0$, "plasticity" parameter $\rho > 0$

Initialization: $\mathbf{W} = \bar{1}$

Define: $F_\sigma(x, \mathbf{W})$ = stochastic output for input x under partial order $\mathbf{W}$

Algorithm:
For $i = 1 \ldots n$
  $z_i = F(x_i, \mathbf{W})$
  If $(z_i \neq y_i)$
    - For any j such that $\phi_j(x, z_i) < \phi_j(x, y_i)$, set $\mathbf{W}_j = \mathbf{W}_j + \rho$
    - For any j such that $\phi_j(x, z_i) > \phi_j(x, y_i)$, set $\mathbf{W}_j = \mathbf{W}_j - \rho$

Output: Partial order $\mathbf{W}$

Properties of The Maximal Gradual Learning Algorithm

- Questions:
  - Does the maximal gla converge?
  - If so, does it converge to a model that matches the frequencies seen in training data?
- Main results appear to be empirical (rather than theoretical)

Experiments on Finnish

- Constraints used:
  - Number of stressed syllables that aren't heavy
  - Number of heavy syllables that aren't stressed
  - Number of stressed syllables with the vowel I/O/A (3 constraints, one for each vowel)
  - Number of unstressed syllables with the vowel I/O/A (3 constraints, one for each vowel)
  - Number of instances of consecutive heavy syllables
  - Number of instances of consecutive light syllables
  - Number of instances of consecutive unstressed syllables

Experiments on Finnish

- Data set has 5698 tokens, 22 different phonological stem structures

- Algorithm run over 388,000 instances,
  with parameter schedule:

Data	Plasticity	σ
First 22,000	2	10
Second 22,000	2	2
Third 22,000	0.2	2
Fourth 22,000	0.02	10
Last 300,000	0.002	10

- Results: final model does a good job of modeling frequencies of the different forms

A Log-Linear Model

- [Goldwater and Johnson, 2003]:
  Learning OT Constraint Rankings Using a Maximum Entropy Model

- Work defines a distribution over possible outputs:

$$P(y \mid x, \mathbf{W}) = \frac{e^{\mathbf{\Phi}(x,y)\cdot\mathbf{W}}}{\sum_{y' \in \mathbf{GEN}(x)} e^{\mathbf{\Phi}(x,y')\cdot\mathbf{W}}}$$

- Parameters are estimated by maximizing

$$L(\mathbf{W}) = \sum_i \log P(y_i \mid x_i, \mathbf{W}) - C \sum_k \mathbf{W}_k^2$$

- Output under parameter values $\mathbf{W}$ is

$$F(x, \mathbf{W}) = \text{argmax}_{y \in \mathbf{GEN}(x)} \mathbf{\Phi}(x, y) \cdot \mathbf{W}$$

- Results are very close to that of the GLA

Equivalence Between Constraint Rankings and Linear Models

- In OT, we have $F(x, \mathbf{U})$ where $\mathbf{U}$ is a *ranking* or *partial order* over the constraints

- In linear models, we have

$$F(x, \mathbf{W}) = \text{argmax}_{y \in \mathbf{GEN}(x)} \mathbf{\Phi}(x, y) \cdot \mathbf{W}$$

  where $\mathbf{W} \in \mathbb{R}^N$

- If for any constraint, there is an upper bound b on the number of violations, i.e.,

$$\mathbf{\Phi}_j(x, y) \leq b \quad \text{for all } x, y, j$$

  then there is a parameter setting $\mathbf{W} \in \mathbb{R}^d$ such that the global linear model defines the same function F as the OT model with parameters $\mathbf{U}$

An Example

- Say the ranking over constraints is

$$\{\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3, \mathbf{U}_4, \mathbf{U}_5\} = \{1, 3, 4, 2, 5\}$$

  i.e.,

$$\phi_1 \gg \phi_4 \gg \phi_2 \gg \phi_3 \gg \phi_5$$

- Say the number of constraint violations is bounded by $b = 9$ for every constraint

- Then we can choose

$$\{\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{W}_4, \mathbf{W}_5\} = \{10^5, 10^3, 10^2, 10^4, 10\}$$

Open Issues

- Language acquisition: How do humans acquire constraint rankings?
  (critical issue: input forms are not seen)

- How robust are the learning algorithms to noise?

Language Acquisition

- The learning algorithms (Tesar and Smolensky, Boersma, Goldwater and Johnson) are intended as models of *human* language acquisition

- Address a basic question in linguistics: how do people acquire language?

- An unrealistic part of these algorithms: **they assume that both input and output forms** (x_i, y_i) **pairs are observed by the learning algorithm**. In human language learning, only **outputs** are seen.

One Proposed Solution to This Problem

- Tesar and Smolensky talk quite a bit about this issue

- Their assumption: given a surface form y_i, and the current constraint ranking **W**, we take x'_i to be the input form such that $F(x'_i, \mathbf{W}) = y_i$. The pair (x'_i, y_i) forms a training example.

- This requires a *parsing algorithm*: a method that takes an output form y_i, a constraint ranking **W**, and returns an input form that produces y_i

- **Assumption:** $x'_i = x_i$,
  i.e., the recovered input is the same as the "true" input
  This appears to be a very strong assumption

Robustness to Noise

- What happens if the (x_i, y_i) sequence is not consistent with *any* constraint ranking **U**?
- This means there is *noise* in the training examples
- "Noise" examples could occur for a number of reasons:
  - The speaker makes an error in production
  - The hearer makes an error in perception of the phoneme sequence
  - The hearer interacts with a community of speakers with slightly different grammars
  - There is some acoustic noise in the environment, which leads to the hearer making a perceptual error

A New Framework: Relative Loss Bounds

- In the case that the (x_i, y_i) sequence **is** consistent with some constraint ranking $\mathbf{U}$, we've shown that the algorithms make at most $N(N-1)/2$ mistakes on any sequence

- In the noisy case, consider the *minimum-error ranking:*

$$\mathbf{U}^* = \arg\min_{\mathbf{U}} \sum_i [[F(x_i, \mathbf{U}) \neq y_i]]$$

  and define $Error(\mathbf{U}^*) = \sum_i [[F(x_i, \mathbf{U}^*) \neq y_i]]$

- For any online algorithm A, protocol is:
  - Initialize $\mathbf{W}$ to some value
  - Receive an input x_i
  - Make a prediction $y_i' = F(x_i, \mathbf{W})$
  - Receive the true output y_i, and make a mistake if $y_i \neq y_i'$
  - Update $\mathbf{W}$ if necessary according to the algorithm A

- We define $L(A)$ to be the number of mistakes made by the algorithm
  Our aim is to come up with an algorithm A such that

$$L(A) \leq \alpha Error(\mathbf{U}^*) + \beta$$

  holds for **all** (x_i, y_i) sequences, where α and β are "small"

- **Intuition**: we can't achieve 0 errors, but we can at least get close to the number of errors of the optimal ranking $\mathbf{U}^*$

Results for Constraint Demotion Algorithms

- For constraint demotion (Tesar and Smolensky), it can be shown that

$$L(A) \leq O(N^2) Error(\mathbf{U}^*) + O(N^2)$$

  and the bound is tight for some sequences

- For the minimal gradual learning algorithm (Boersma) it can be shown that

$$L(A) \leq O(N) Error(\mathbf{U}^*) + O(N^2)$$

  and the bound is tight for some sequences

- $\Rightarrow$ Boersma's algorithm is more robust to noise than Tesar and Smolensky's

- **But** in the worst case, if N is large (say 100), then both algorithms can be badly mislead by noisy examples
  (with 100 constraints, with a noise rate of 1% the bound is vacuous)

Are There Better Algorithms?

- It's probably difficult to come up with *efficient* algorithms which do better than $O(N)Error(\mathbf{U}^*)$ (the problem is very closely related to decision list learning, and as far as I know efficient algorithms with better coefficents are not known)

- **But** we'll consider algorithms which require a vote to be taken across all $N!$ rankings
  (the weighted majority algorithm)

- The algorithm is not efficient ($N!$ is large!) but it's interesting that the algorithm is *much* more robust to noise

- The algorithm has a loss bound of

$$\begin{aligned} L(A) &\leq 2.63Error(\mathbf{U}^*) + 2.63 \log N! \\ &\leq 2.63Error(\mathbf{U}^*) + 2.63N \log N \end{aligned}$$

- e.g., see the tutorial by Manfred Warmuth at COLT 1999

A Warm Up for the Weighted-Majority Algorithm: The Halving Algorithm

Simple case: assume our training sequence is a series of (x_i, y_i, z_i) triples, where either y_i or z_i is the correct output for x_i, and the algorithm must predict which of y_i or z_i is correct

- **Initialize:** Take $\mathcal{S}$ to be the set of all possible grammars (constraint ranking)

- **For** $i = 1 \ldots n$**:**

  - Define $\mathcal{S}_1 = \{\mathbf{U} : F(x_i, \mathbf{U}) = y_i\}$
    $\mathcal{S}_2 = \{\mathbf{U} : F(x_i, \mathbf{U}) = z_i\}$
  - If $|\mathcal{S}_1| \geq |\mathcal{S}_2|$, then predict y_i as the output, otherwise predict z_i
  - If algorithm proposes y_i, and this is a mistake, then $\mathcal{S} = \mathcal{S}_2$
  - If algorithm proposes z_i, and this is a mistake, then $\mathcal{S} = \mathcal{S}_1$
  - Else S remains unchanged

- If a grammar with 0 errors exists, the algorithm makes at most $\log_2 N!$ errors
- Proof sketch: every time a mistake is made, the size of $\mathcal{S}$ is at least halved. At the end of the training sequence, all 0-error hypotheses remain in $\mathcal{S}$, so $|\mathcal{S}| > 0$ at the end of the algorithm. Initial size is $|\mathcal{S}| = N!$, so at most $\log_2 N!$ mistakes can be made

The Weighted Majority Algorithm

- **Initialize:** Take $\mathcal{S}$ to be the set of all possible grammars (constraint rankings). For each grammar $s \in \mathcal{S}$, assign an initial weight of $v_s = 1$

- **For** $i = 1 \ldots n$**:**
  - Define $\mathcal{S}_1 = \{\mathbf{U} : F(x_i, \mathbf{U}) = y_i\}$
    $\mathcal{S}_2 = \{\mathbf{U} : F(x_i, \mathbf{U}) = z_i\}$
  - Define $V_1 = \sum_{s \in \mathcal{S}_1} v_s$
    $V_2 = \sum_{s \in \mathcal{S}_2} v_s$
  - If $V_1 \geq V_2$, then predict y_i as the output, otherwise predict z_i
  - If true output is y_i, then for all $s \in \mathcal{S}_2$, set $v_s = v_s \times \beta$ where $0 \leq \beta < 1$
  - If true output is z_i, then for all $s \in \mathcal{S}_1$, set $v_s = v_s \times \beta$

Intuition: take a weighted vote across the possible grammars. Each time a grammar makes an error, its weight is decreased by a factor of β

Theorem:
If $Error(\mathbf{U}^*)$ is the number of errors of the best grammar on the sequence (x_i, y_i, z_i), and there are $N!$ grammars under consideration, then the number of errors of the weighted majority algorithm A is such that

$$L(A) \leq \frac{\log \frac{1}{\beta}}{\log \frac{2}{1+\beta}} Error(\mathbf{U}^*) + \frac{1}{\log \frac{2}{1+\beta}} \log N!$$

if $\beta = 1/e$, then this gives

$$L(A) \leq 2.63 Error(\mathbf{U}^*) + 2.63 \log N!$$

This is much better than the constraint demotion bounds, $L(A) \leq O(N)Error(\mathbf{U}^*) + O(N^2)$ **but** the algorithm requires dealing explicitly with all $N!$ possible grammars. So *computationally* the algorithm is impractical, but it is *very* robust to noise. Could we find some efficient/approximate implementation of this algorithm?

The Weighted Majority Algorithm: Proof

(See tutorial by Manfred Warmuth at COLT 99)

- Define $n =$ number of training examples
- Define $S =$ number of grammars ($N!$ in our case)
- $m_{s,i}$ = number of mistakes made by grammar s before i'th example
- $v_{s,i} = \beta^{m_{s,i}} =$ weight of s before i'th example
- $V_i = \sum_s v_{s,i} =$ total weight before i'th example

Then if no mistake is made at the i'th example,

$$V_i \leq V_{i-1}$$

(because 0 or more hypotheses are downweighted by β)
Else if a mistake is made at the i'th example,

$$V_i \leq 0.5V_{i-1} + 0.5\beta V_{i-1} = \frac{1+\beta}{2} V_{i-1}$$

(because all grammars in the majority vote are downweighted by β, and these grammars account for at least half of V_{i-1})

Hence if M is the total number of mistakes made by the algorithm,

$$V_{n+1} \leq \left(\frac{1+\beta}{2}\right)^M V_1$$

We also have

$$V_{n+1} = \sum_s v_{s,n+1} = \sum_s \beta^{M_{s,n+1}} \geq \beta^{M^*}$$

where M^* is the minimum number of errors made by *any* of the grammars
Hence

$$\beta^{M^*} \leq \left(\frac{1+\beta}{2}\right)^M V_1 = \left(\frac{1+\beta}{2}\right)^M S$$

where $S = N!$ is the number of grammars under consideration
Solving for M gives

$$M \leq \frac{\log \frac{1}{\beta}}{\log \frac{2}{1+\beta}} M^* + \frac{1}{\log \frac{2}{1+\beta}} \log S$$