Learning as Search Optimization: Approximate Large Margin Methods for Structured Prediction

Hal Daumé III and **Daniel Marcu**

Information Sciences Institute

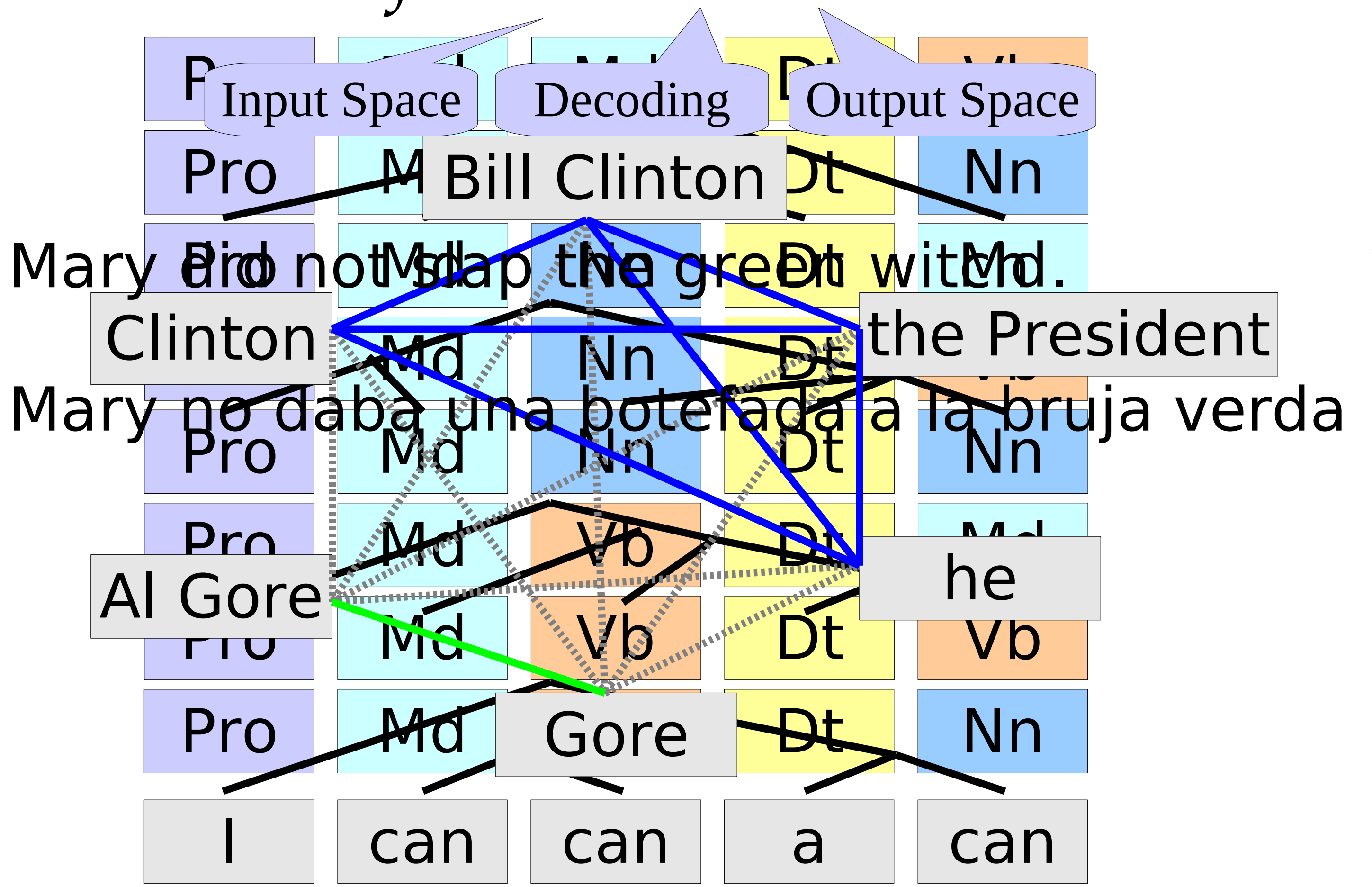
University of Southern California

`{hdaume,marcu}@isi.edu`

Structured Prediction 101

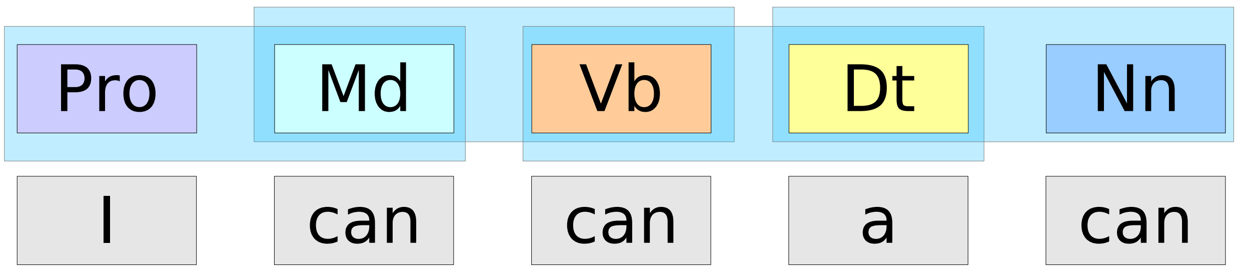
- Learn a function mapping inputs to complex outputs:

$$f : X \rightarrow Y$$



Problem Decomposition

- Divide problem into *regions*
  - Express both the *loss function* and the *features* in terms of regions:



- Decoding:
  - Tractable using dynamic programming when regions are *simple* *(max-product algorithm)*
- Parameter estimation (linear models – CRF, M3N, SVMSO, etc):
  - Tractable using dynamic programming when regions are *simple* *(sum-product algorithm)*

Problem

- In many (most?) problems, decoding is hard:
  - Coreference resolution
  - Machine translation
  - Automatic document summarization
  - Even joint sequence labeling!

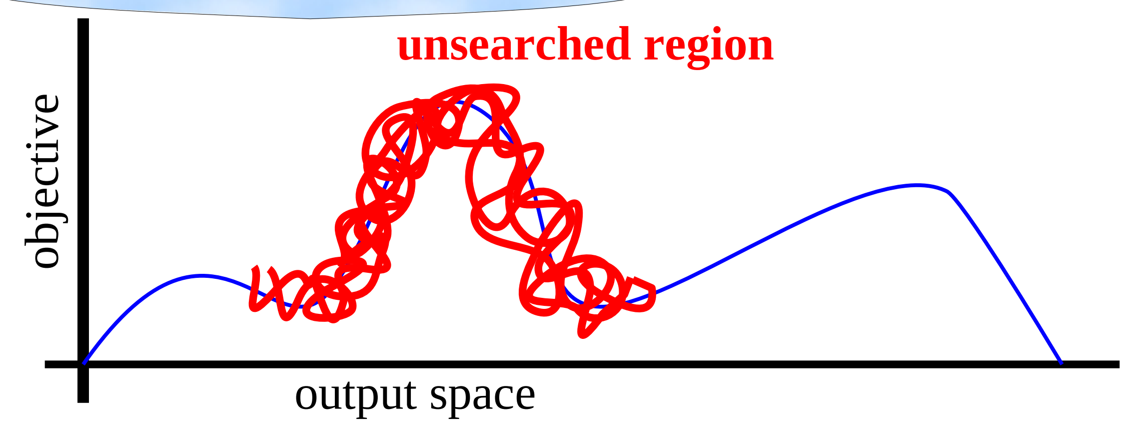
Suboptimal heuristic search

NP VP

Want weights that are *optimal*

for a *suboptimal* search procedure

- I… …e, optimality is gone



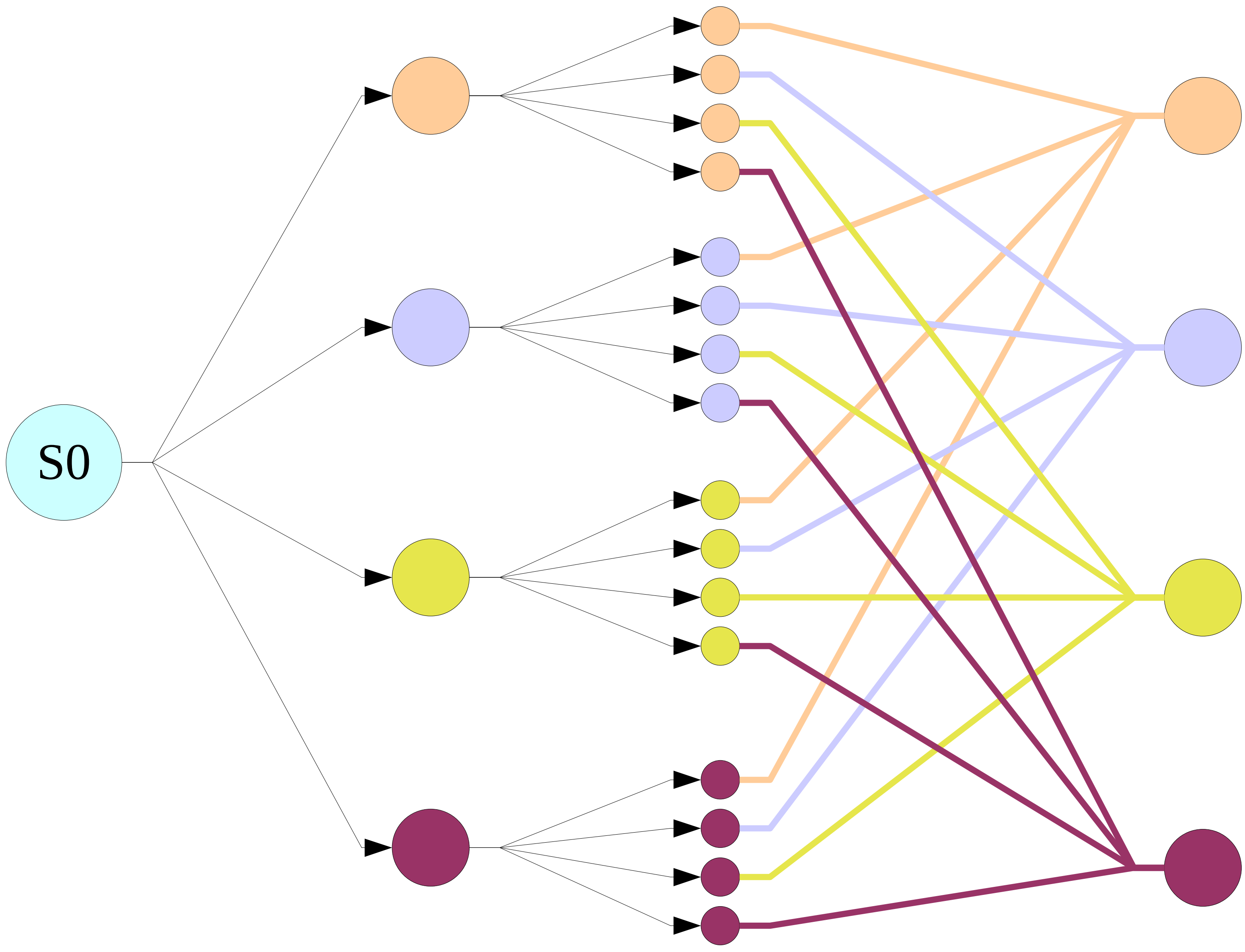
Generic Search Formulation

- Search Problem:
  - Search space
  - Operators
  - Goal-test function
  - Path-cost function
- Search Variable:
  - Enqueue function

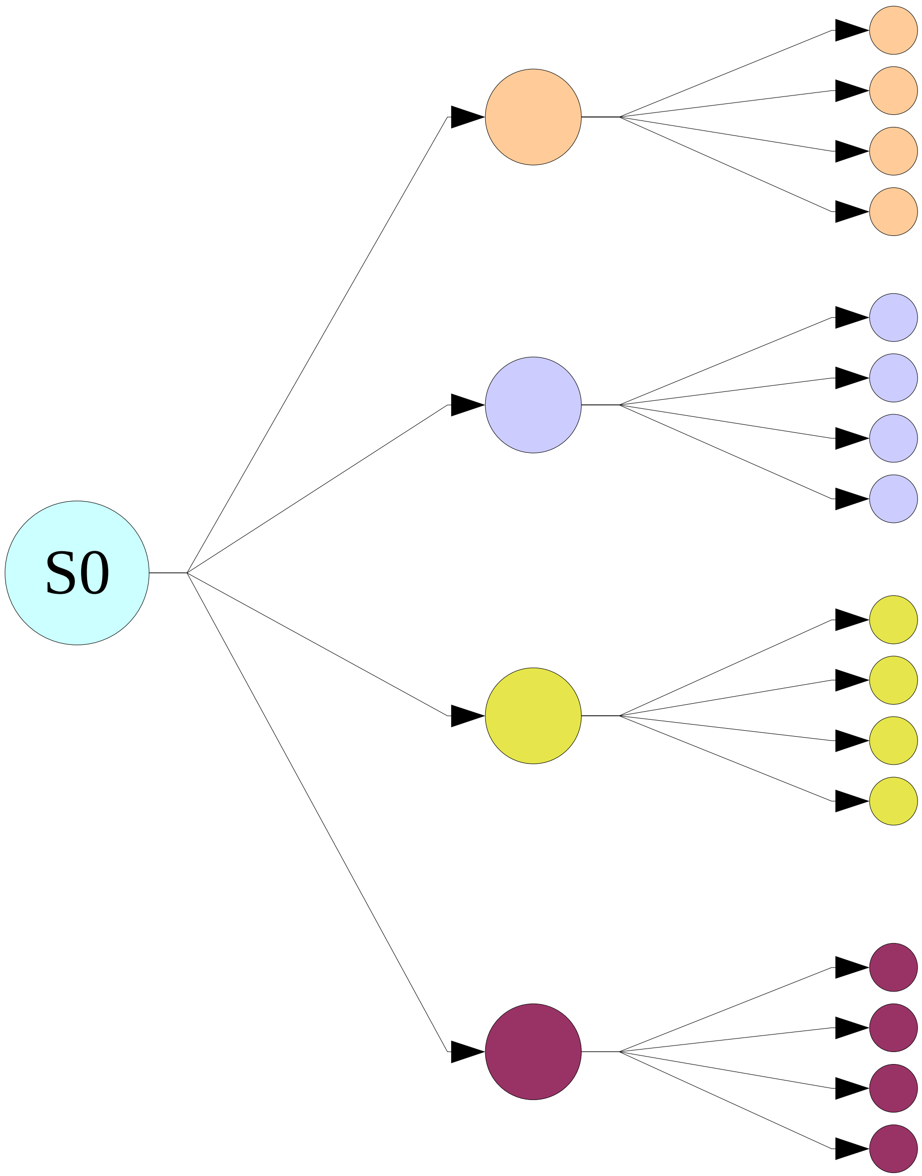
- nodes := MakeQueue(S0)
- **while** nodes is not empty
  - node := RemoveFront(nodes)
  - **if** node is a goal state **return** node
  - next := Operators(node)
  - nodes := Enqueue(nodes, next)
- **fail**

Varying the **Enqueue** function can give us DFS, BFS, beam search, A* search, etc...

Exact (DP) Search



Beam Search



Inspecting Enqueue

- Generally, we sort nodes by:

$$f(n) = g(n) + h(n)$$

Node value

Path cost

Future cost

Assume this is given

Assume this is a linear function of features:

$$g(n) = \bar{w}^T \Phi(x, n)$$

Formal Specification

- Given:
  - An input space X, output space Y, and search space S
  - A parameter function $\Phi : X \times S \rightarrow \mathbb{R}^D$
  - A loss function that decomposes over search: $l : X \times Y \times Y \rightarrow \mathbb{R}^{\geq 0}$

$$\begin{aligned} l(x,y,\hat{y}) &\leq l(x,y,n) \quad (\forall n \rightarrow \hat{y}) \\ l(x,y,n) &\leq l(x,y,\tilde{n}) \quad (\forall n \rightarrow \tilde{n}) \end{aligned}$$

(not absolutely necessary)

(monotonicity)

- Find weights w to minimize:

$$\left.\begin{aligned} L &= \sum_{m=1}^{M} l(x_m, y_m, \hat{y}=search(x_m; w)) \\ &\leq \sum_{m=1}^{M} \sum_{n \rightarrow \hat{y}} \left[l(x_m, y_m, n) - l(x_m, y_m, par(n)) \right] \end{aligned}\right\} + \text{regularization term}$$

We focus on 0/1 loss

Online Learning Framework (LaSO)

- nodes := MakeQueue(S0)
- **while** nodes is not empty
  - node := RemoveFront(nodes)
  - **if** none of {node} ∪ nodes is y-good **or** node is a goal **&** not y-good
    - sibs := siblings(node, y)
    - w := update(w, x, sibs, {node} ∪ nodes)
    - nodes := MakeQueue(sibs)
  - **else**
    - **if** node is a goal state **return** w
    - next := Operators(node)
    - nodes := Enqueue(nodes, next)

Monotonicity: for any node, we can tell if it can lead to the correct solution or not

If we erred...

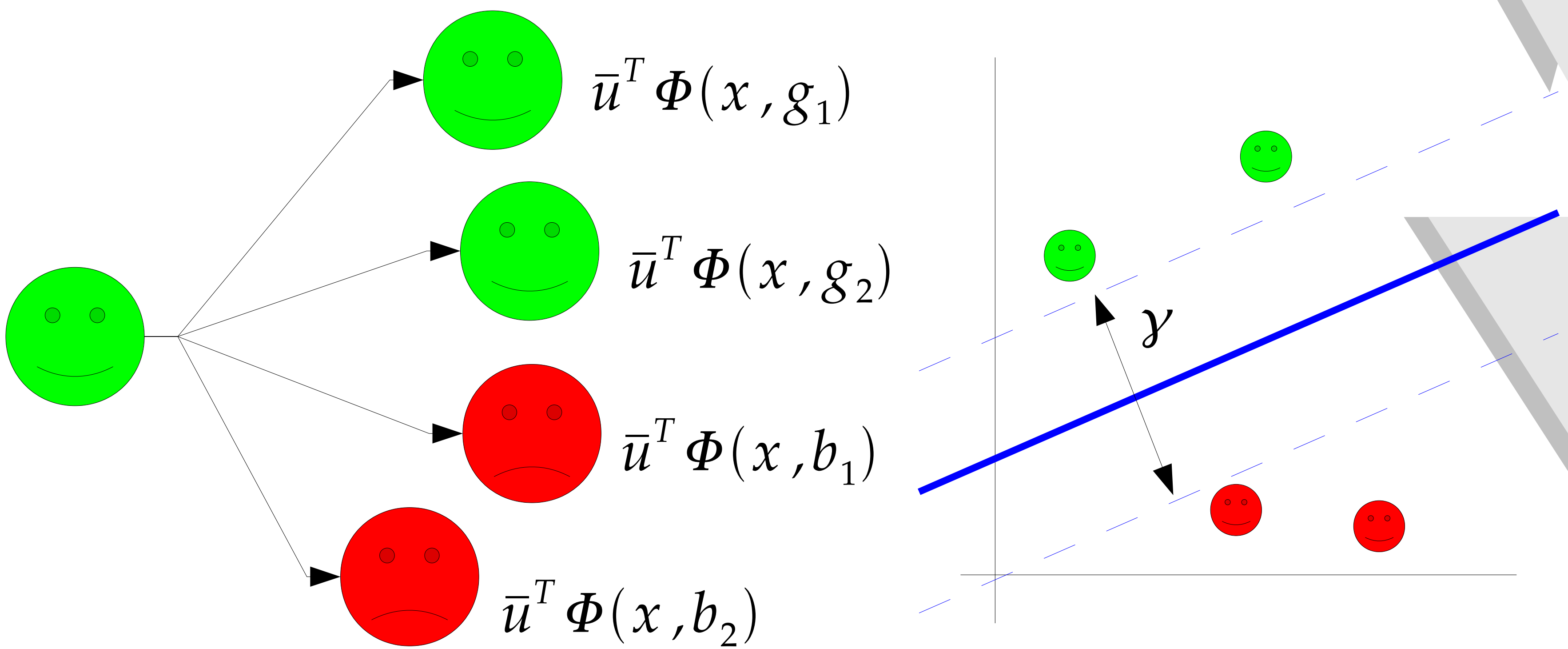
Where should we have gone?

Update our weights based on the good and the bad choices

Continue search...

Search-based Margin

- The *margin* is the amount by which we are correct:



Note that the *margin* and hence *linear separability* is also a function of the *search algorithm!*

Update Methods:

- Perceptron updates:

[Rosenblatt 1958; Freund+Shapire 1999; Collins 2002]

$$\bar{w} \leftarrow \bar{w} + \underbrace{\left[\sum_{n \in good} \frac{\Phi(x,n)}{|good|}\right] - \left[\sum_{n \in bad} \frac{\Phi(x,n)}{|bad|}\right]}_{\Delta}$$

- Approximate large margin updates:

[Gentile 2001]

Nuisance param, use $\sqrt{2}$

$$\bar{w} \leftarrow \wp\left(\bar{w} + \frac{C}{\sqrt{k}} \wp(\Delta)\right)$$

Generation of weight vector

Project into unit sphere

$$\wp(\bar{u}) = \bar{u} \,/\, max\{0, \|\bar{u}\|\}$$

- Also downweight y-good nodes by:

Nuisance param, use $1/\alpha$

$$(1-\alpha)\frac{B}{\sqrt{k}}$$

Ratio of desired margin

Convergence Theorems

[Rosenblatt 1958; Freund+Shapire 1999; Collins 2002]

- For linearly separable data:
  - For perceptron updates, $K \leq \gamma^{-2}$

    Number of updates

  - For large margin updates,

[Gentile 2001]

$$K \leq \frac{2}{\gamma^2}\left(\frac{2}{\alpha} - 1\right)^2 + \frac{8}{\alpha} - 4$$

$$= 2\gamma^{-2} + 4 \qquad (\alpha = 1)$$

- Similar bounds for inseparable case

Experimental Results

- Two related tasks:
  - Syntactic chunking

    (exact search + estimation is possible)
  - Joint chunking + part of speech tagging

    (search + estimation intractable)

[Sutton + McCallum 2004]

- Data from CoNLL 2000 data set
  - 8936 training sentences (212k words)
  - 2012 test sentences (47k words)
  - The usual suspects as features:
    - Chunk length, word identity (+lower-cased, +stemmed), case pattern, {1,2,3}-letter prefix and suffix
    - Membership on lists of names, locations, abbreviations, stop words, etc
    - Applied in a window of 3
    - For syntactic chunking, we also use output of Brill's tagger as POS information

Syntactic Chunking

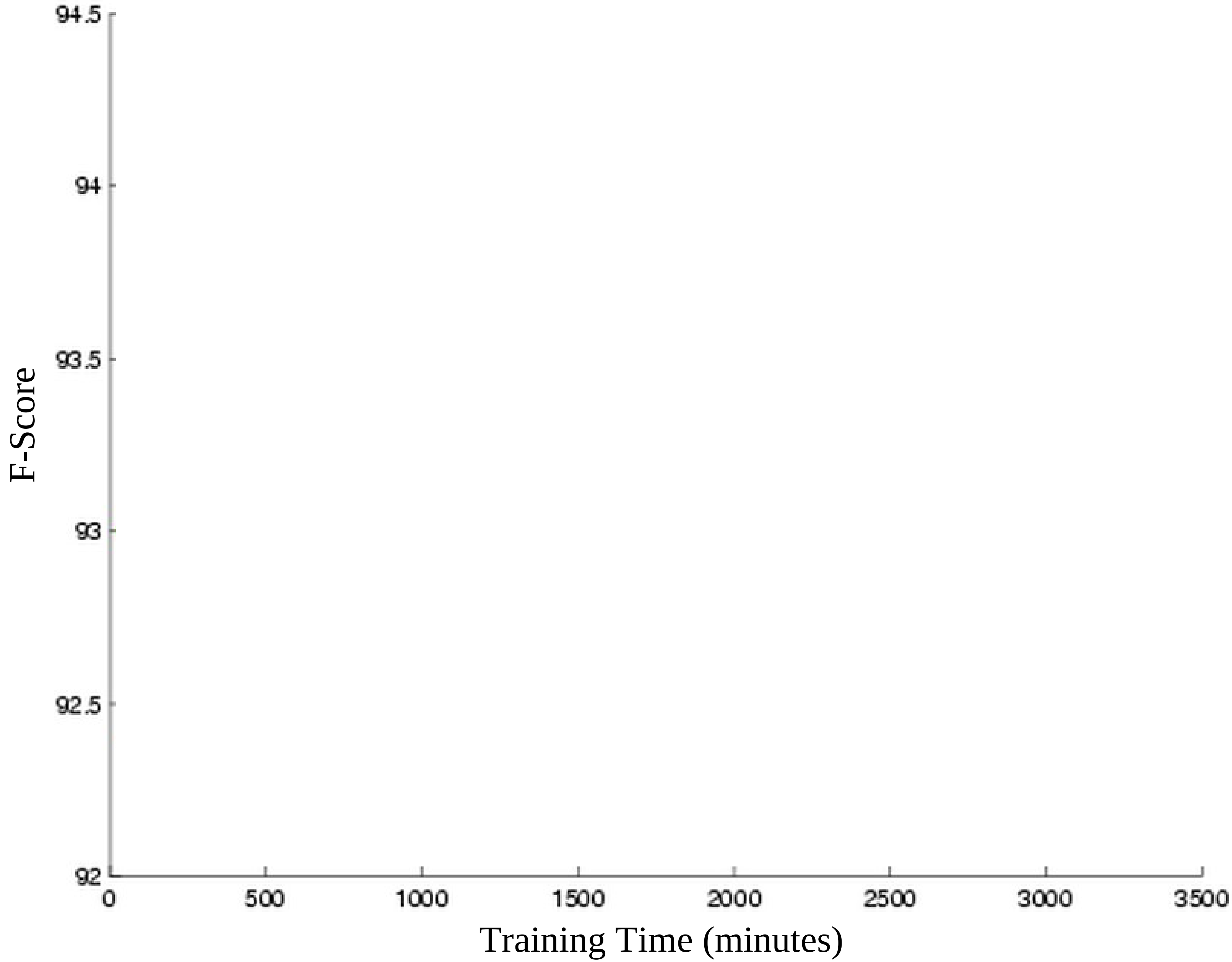
- Search:
  - Left-to-right, hypothesizes entire chunk at a time:

    (Great American)$_{NP}$ (said)$_{VP}$ (it)$_{NP}$ (increased)$_{VP}$ (its loan-loss reserves)$_{NP}$ (by)$_{PP}$ ($ 93 million)$_{NP}$ (after)$_{PP}$ (reviewing)$_{VP}$ (its loan portfolio)$_{NP}$, ...

- Enqueue functions:
  - Beam search: sort by cost, keep only top *k* hypotheses after each step
    - An error occurs exactly when none of the beam elements are good
  - Exact search: store costs in dynamic programming lattice
    - An error occurs *only* when the fully-decoded sequence is wrong
    - Updates are made by summing over the *entire lattice*
    - *This is nearly the same as the CRF/M3N/SVMISO updates, but with evenly weighted errors*

$$\Delta = \left[\sum_{n \in good} \frac{\Phi(x,n)}{|good|} \right] - \left[\sum_{n \in bad} \frac{\Phi(x,n)}{|bad|} \right]$$

Syntactic Chunking Results



Joint Tagging + Chunking

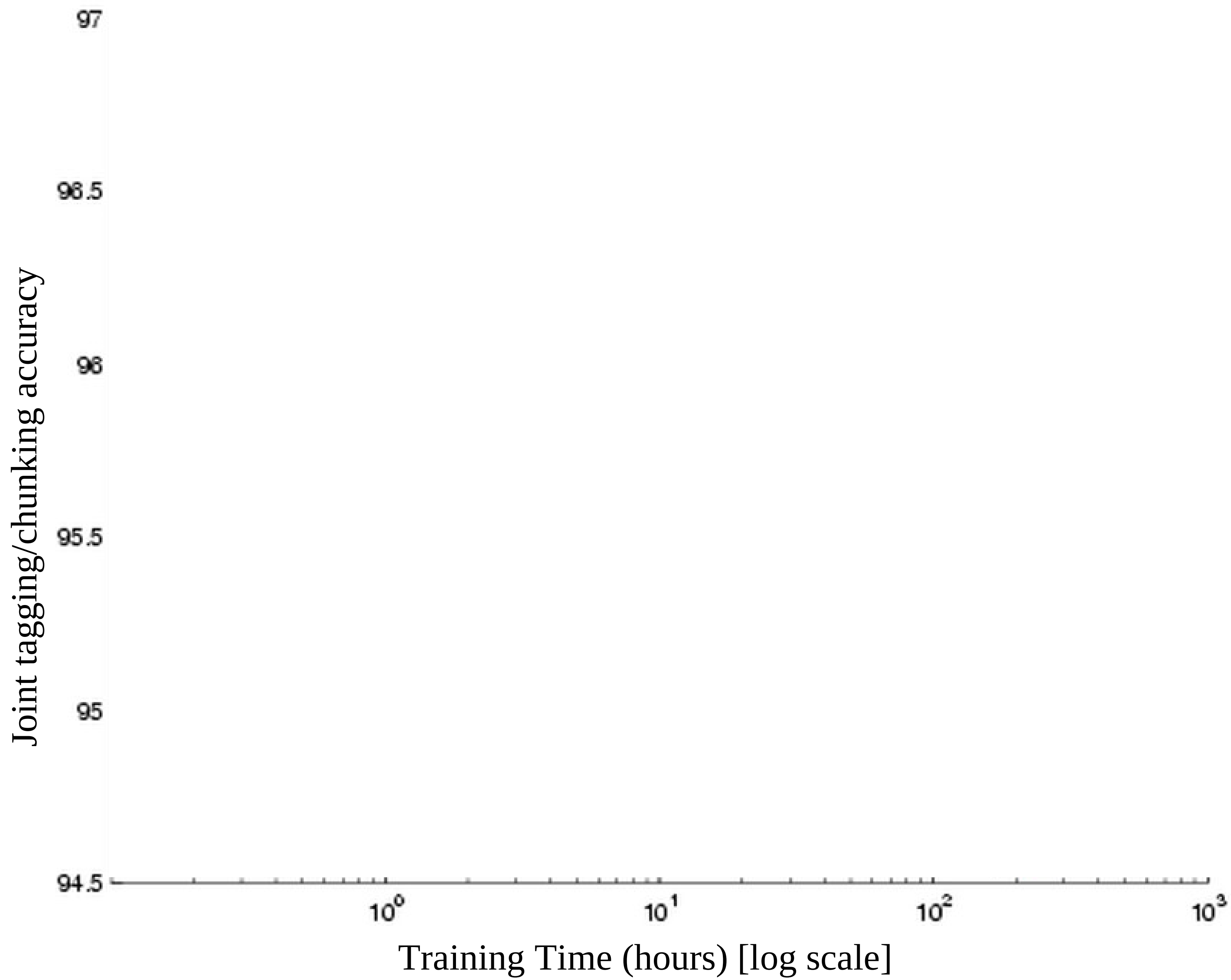
- Search: left-to-right, hypothesis POS and BIO-chunk

Great	American	said	it	increased	its	loan-loss	reserves	by	...
NNP	NNP	VBD	PRP	VBD	PRP$	NN	NNS	IN	...
B-NP	I-NP	B-VP	B-NP	B-VP	B-NP	I-NP	I-NP	B-PP	...

  - Previous approach: Sutton+McCallum use belief propagation algorithms (eg., tree-based reparameterization) to perform inference in a double-chained CRF (13.6 hrs to train on 5%: 400 sentences)

- Enqueue: beam search

Joint T+C Results



Variations on a Beam

- Observation:
  - We needn't use the same beam size for training and decoding
  - Varying these values independently yields:

Training Beam \ Decoding Beam	1	5	10	25	50
1	93.9	92.8	91.9	91.3	90.9
5	90.5	94.3	94.4	94.1	94.1
10	89.5	94.3	94.4	94.2	94.2
25	88.7	94.2	94.5	94.3	94.3
50	88.4	94.2	94.4	94.2	94.4

Conclusions

- Problem:
  - Solving most problems is intractable
  - How can we learn effectively for these problems?
- Solution:
  - Integrate learning with *search* and learn parameters that are both good for identifying correct hypotheses and guiding search
- Results: State-of-the-art performance at low computational cost

- Current work:
  - Apply this framework to more complex problems
  - Explore alternative loss functions
  - Better formalize the optimization problem
    - Connection to CRFs, M3Ns and SVMSOs
    - Reductionist strategy