Machine Learning

CUNY Graduate Center, Spring 2013

Lectures 9-10: Structured Learning

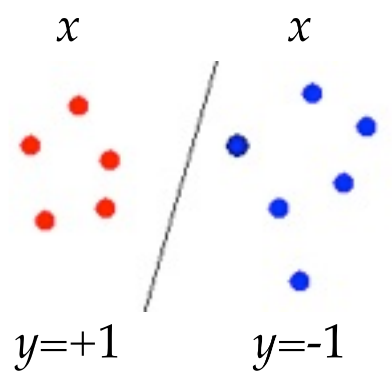
(structured perceptron, HMM, learning w/ inexact search)

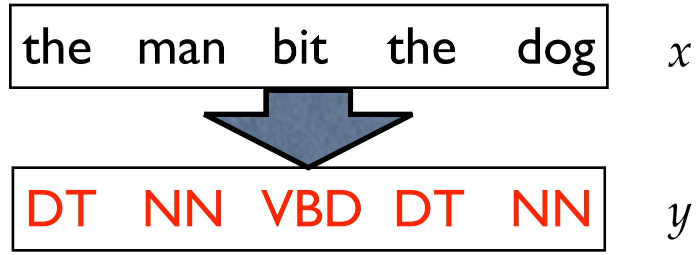
Professor Liang Huang

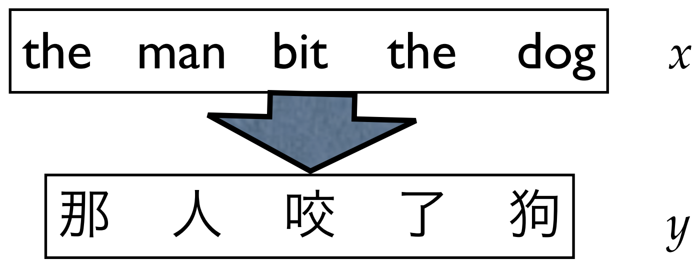
huang@cs.qc.cuny.edu

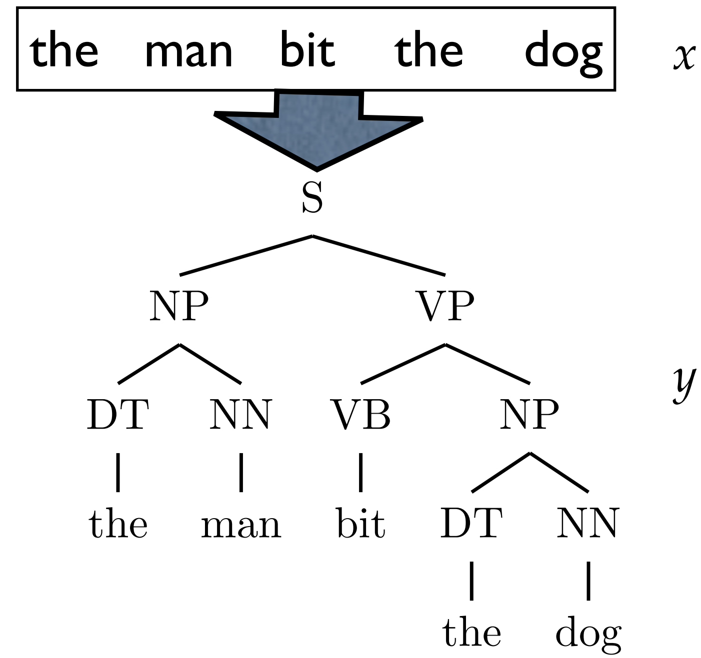
http://acl.cs.qc.edu/~lhuang/teaching/machine-learning

Structured Prediction





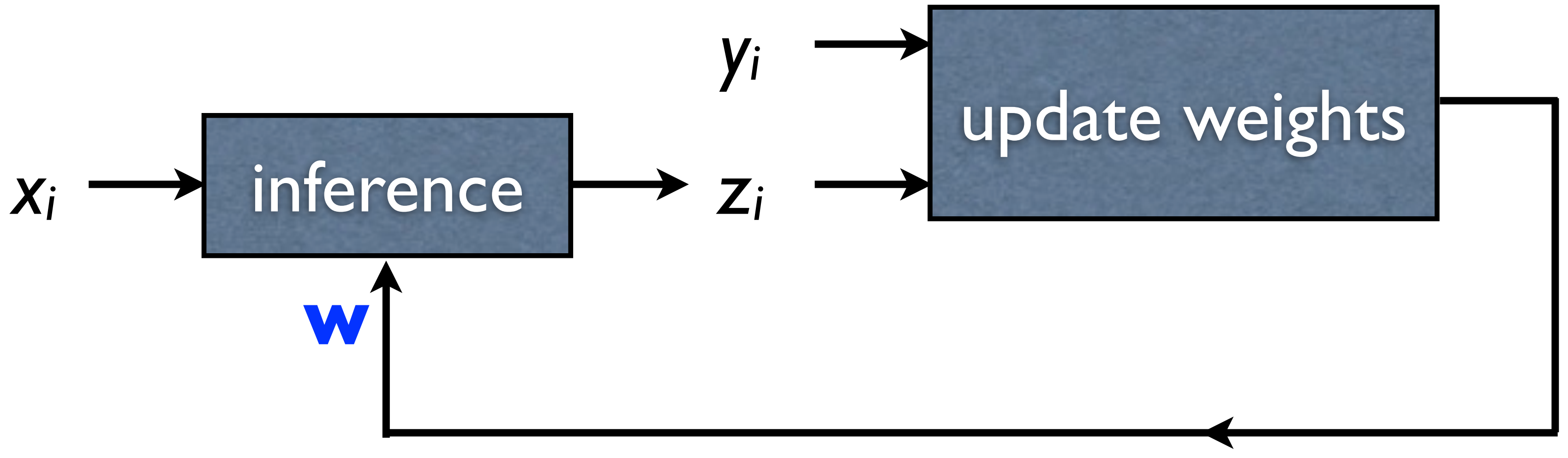




- binary classification: output is binary
- multiclass classification: output is a number (small # of classes)
- structured classification: output is a structure (seq., tree, graph)
    - part-of-speech tagging, parsing, summarization, translation
    - exponentially many classes: *search* (inference) efficiency is crucial!

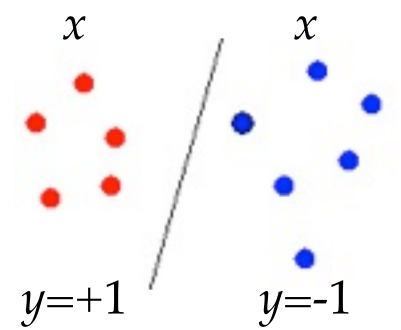
Generic Perceptron

- online-learning: one example at a time
- learning by doing
  - find the best output under the current weights
  - update weights at mistakes



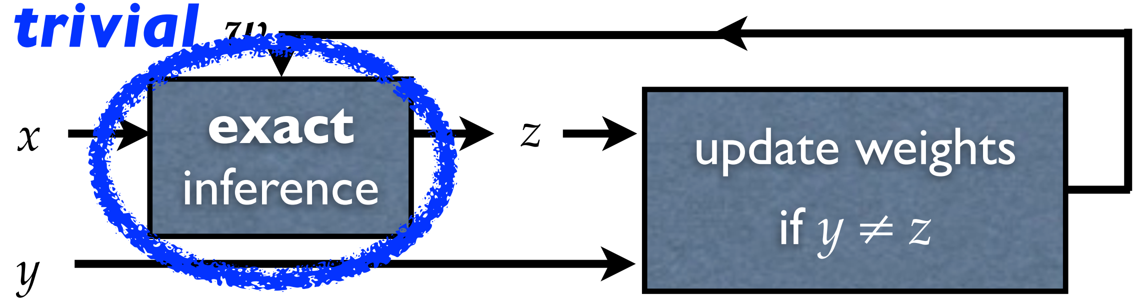
Perceptron: from binary to structured

binary classification



2 classes

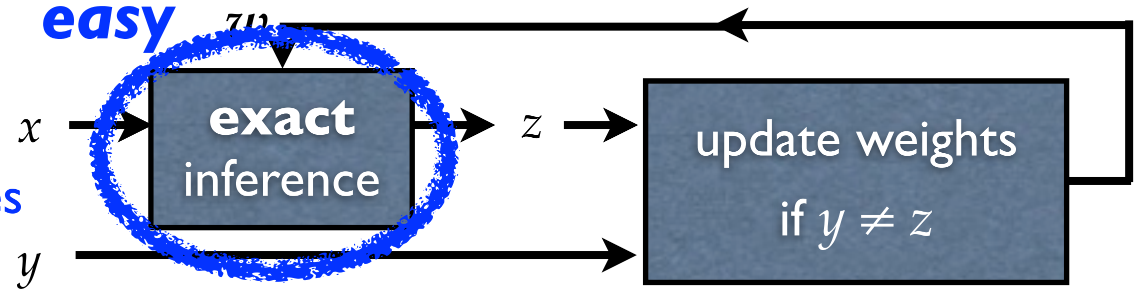
trivial



multiclass classification

constant
# of classes

easy

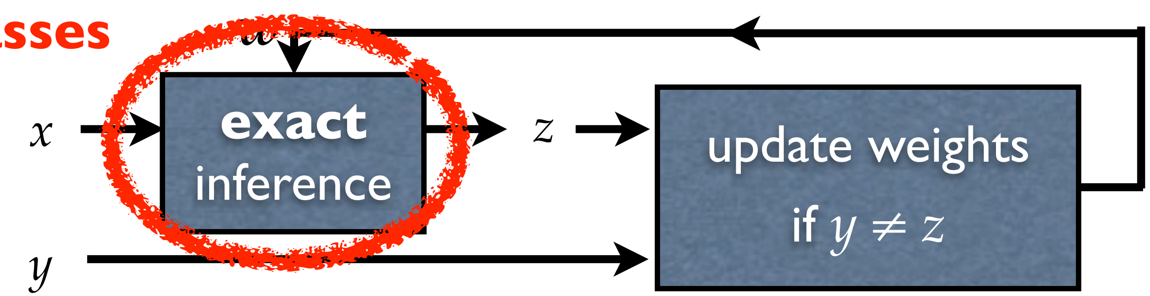


structured classification

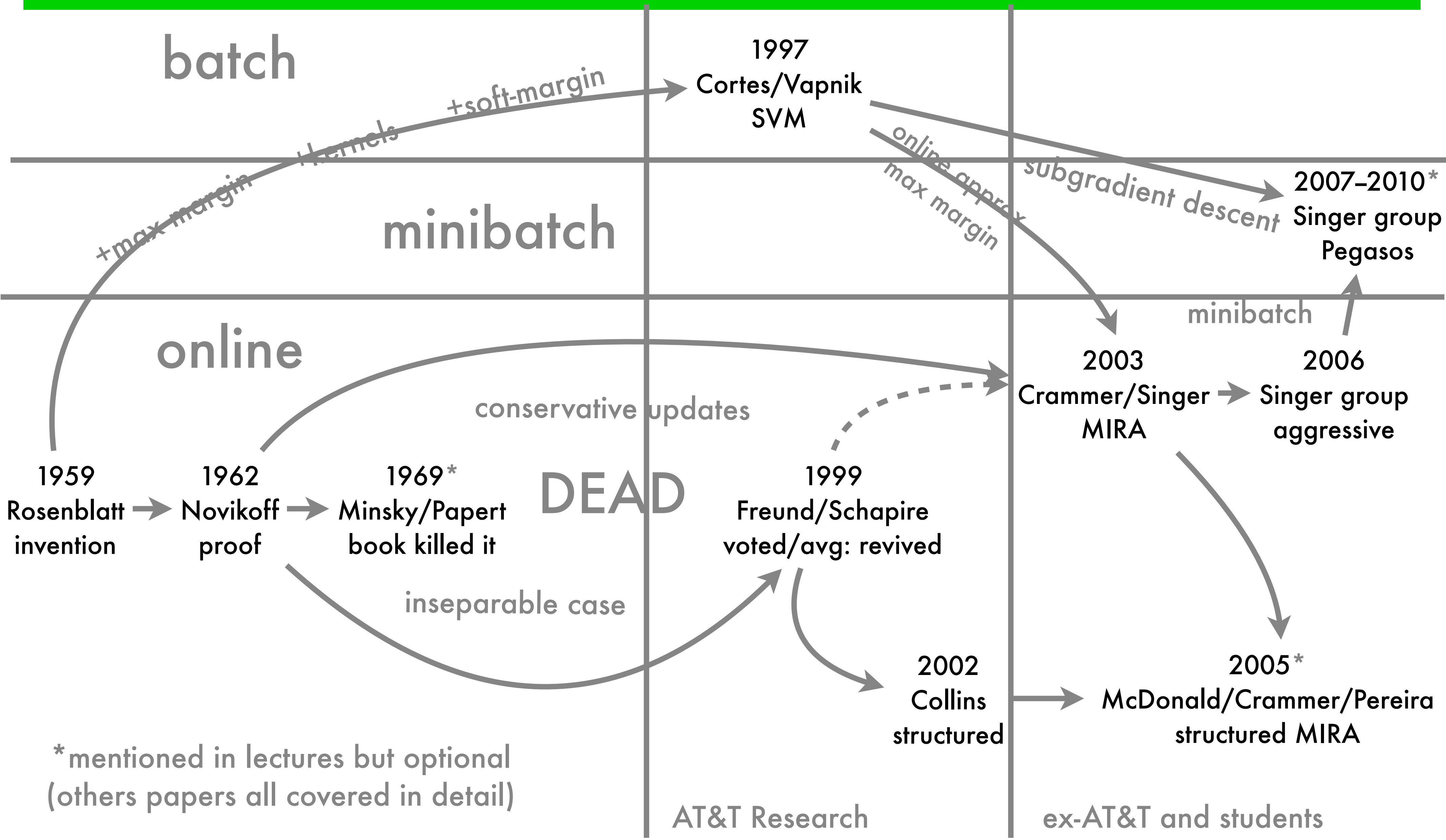
the	man	bit	the	dog	x
DT	NN	VBD	DT	NN	y

exponential
# of classes

hard



Brief History of Perceptron



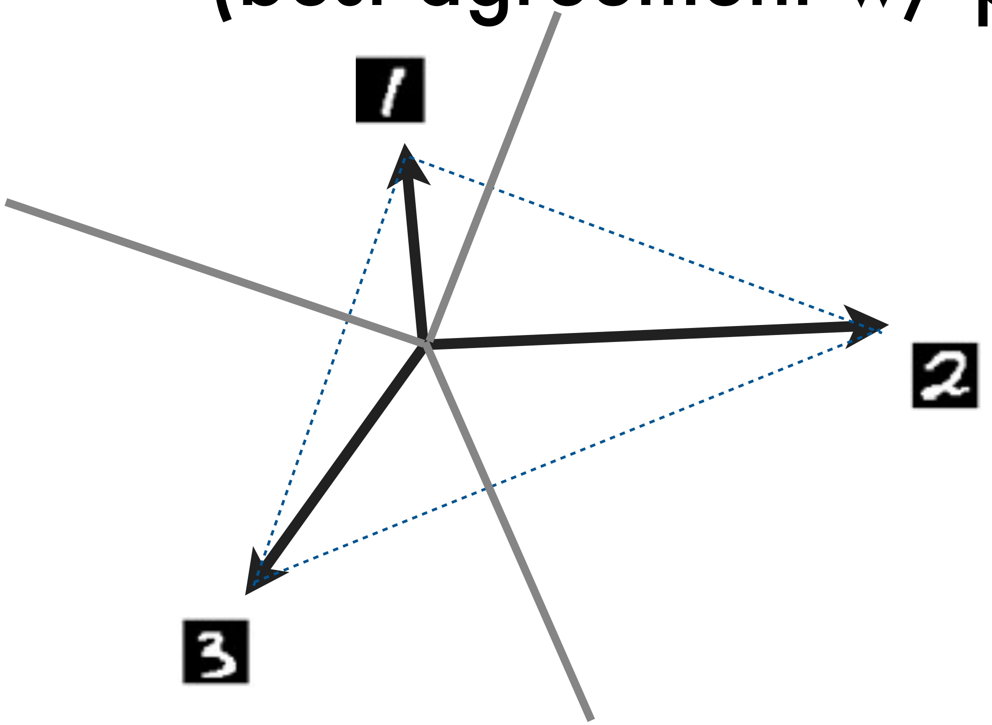
*mentioned in lectures but optional
(others papers all covered in detail)

Multiclass Classification: Review

- one weight vector ("prototype") for each class:

$$\mathbf{w} = (\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \ldots, \mathbf{w}^{(M)}),$$

- multiclass decision rule: (best agreement w/ prototype) $\hat{y} = \mathop{\mathbf{argmax}}_{z \in 1 \ldots M} w^{(z)} \cdot x$

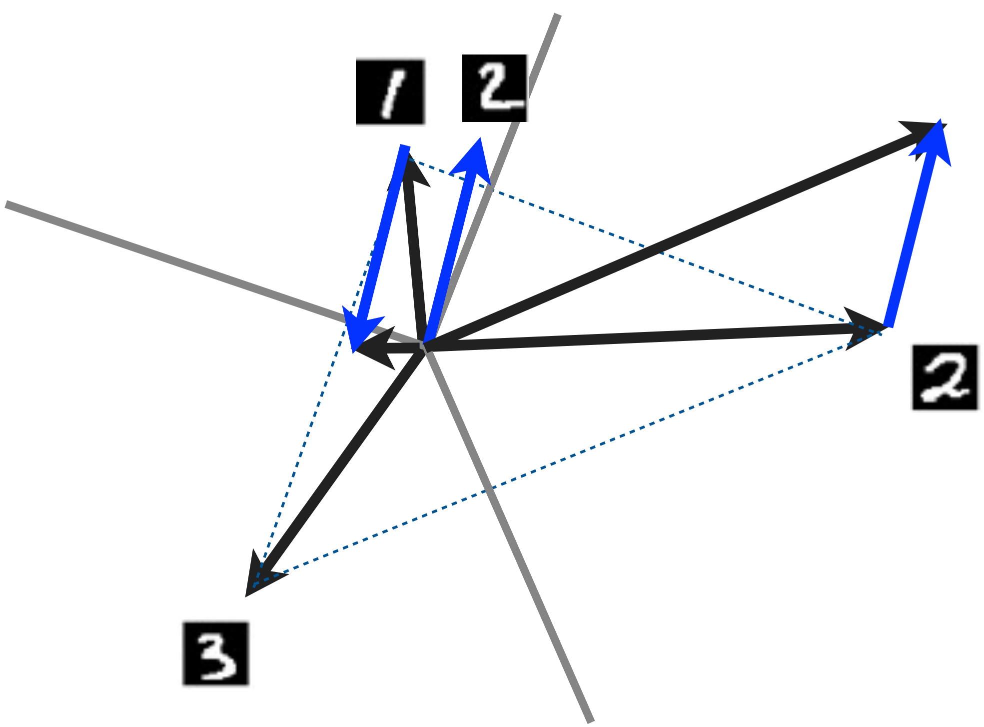


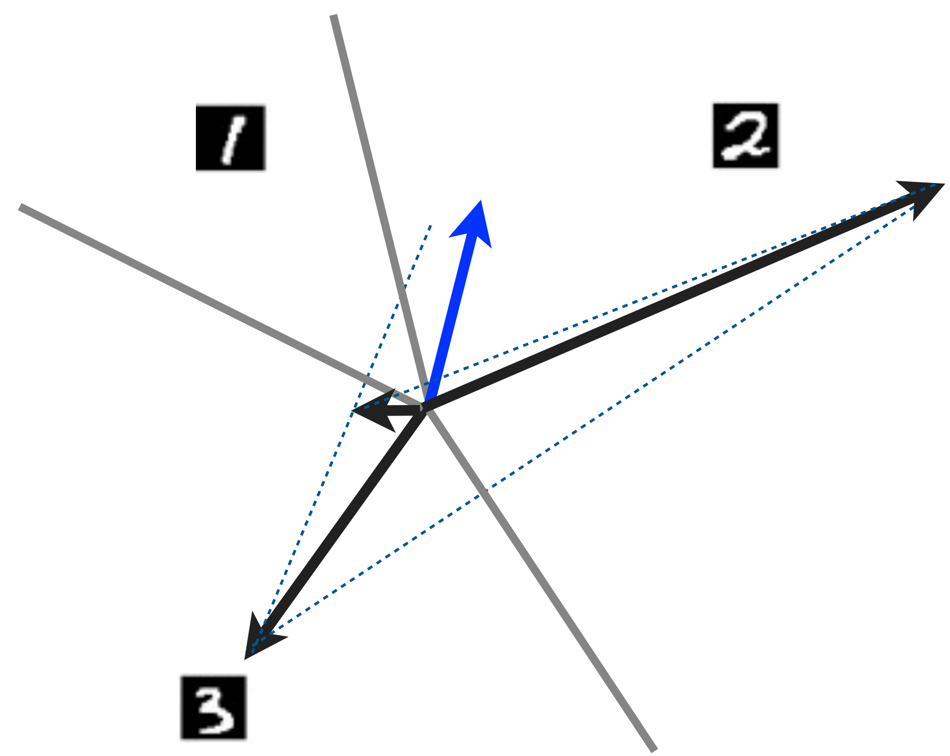
Q1: what about 2-class?

Q2: do we still need augmented space?

Multiclass Perceptron: Review

- on an error, penalize the weight for the wrong class, and reward the weight for the true class





Convergence of Multiclass

$$\mathbf{w} = (\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \ldots, \mathbf{w}^{(M)}),$$

where $\mathbf{w}^{(i)}$ is used to calculate the functional margin for training example with label i;

for a given training example $\mathbf{x}$ and a label y, we define feature map function $\mathbf{\Phi}$ as

$$\mathbf{\Phi}(\mathbf{x}, y) = (\mathbf{0}^{(1)}, \ldots, \mathbf{0}^{(y-1)}, \mathbf{x}, \mathbf{0}^{(y+1)}, \ldots, \mathbf{0}^{(M)}).$$

such that $\mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x}, y) = \mathbf{w}^{(y)} \cdot \mathbf{x}$.

We also define that, with a given training example $\mathbf{x}$, the difference between two feature vectors for labels y and z as $\Delta\mathbf{\Phi}$:

$$\Delta\mathbf{\Phi}(\mathbf{x}, y, z) = \mathbf{\Phi}(\mathbf{x}, y) - \mathbf{\Phi}(\mathbf{x}, z).$$

update rule:

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{\Phi}(\mathbf{x}, y, z)$$

separability:

$$\exists \mathbf{u}, \text{ s.t. } \forall (\mathbf{x}, y) \in D, z \neq y$$

$$\mathbf{u} \cdot \Delta\mathbf{\Phi}(\mathbf{x}, y, z) \geq \delta$$

Example: POS Tagging

- gold-standard: DT NN VBD DT NN y
  - the man bit the dog x

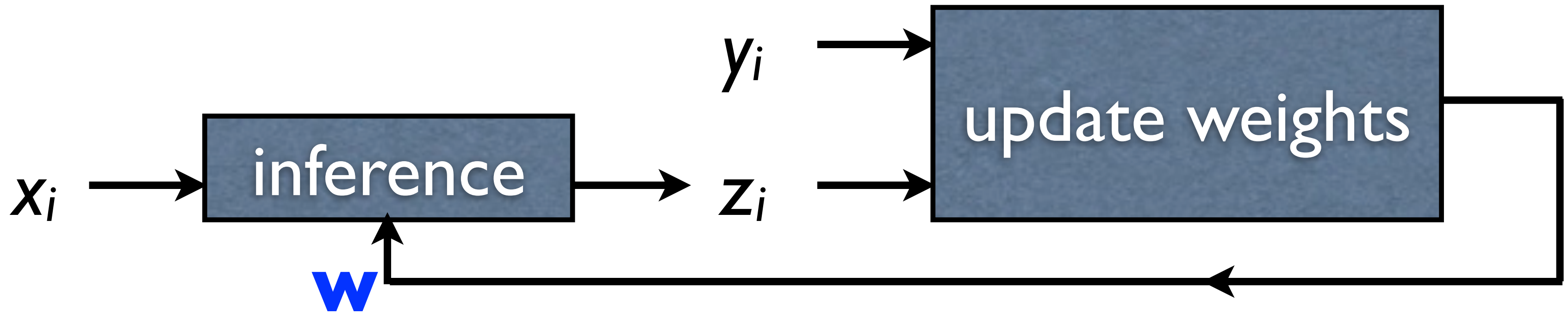
$\Phi(x, y)$

- current output: DT NN NN DT NN z
  - the man bit the dog x

$\Phi(x, z)$

- assume only two feature classes
  - tag bigrams t_{i-1} t_i
  - word/tag pairs w_i
- weights ++: (NN, VBD) (VBD, DT) (VBD → bit)
- weights −−: (NN, NN) (NN, DT) (NN → bit)

Structured Perceptron



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

Initialization: $\mathbf{W} = 0$

Define: $F(x) = \boxed{\operatorname{argmax}_{y \in \mathbf{GEN}(x)}} \, \mathbf{\Phi}(x, y) \cdot \mathbf{W}$

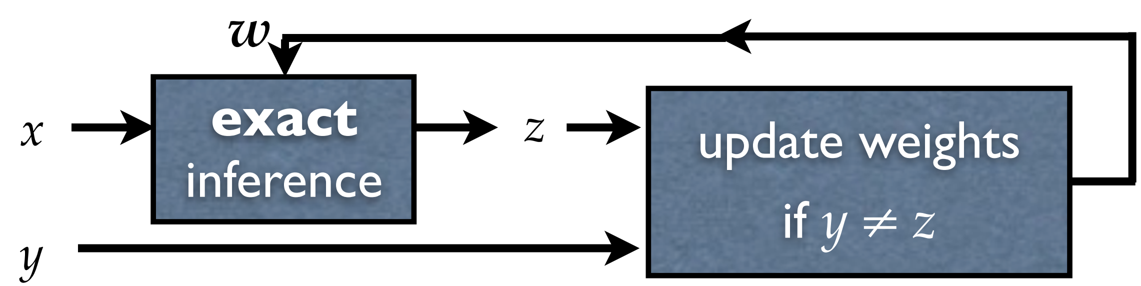
Algorithm: For $t = 1 \ldots T$, $i = 1 \ldots n$

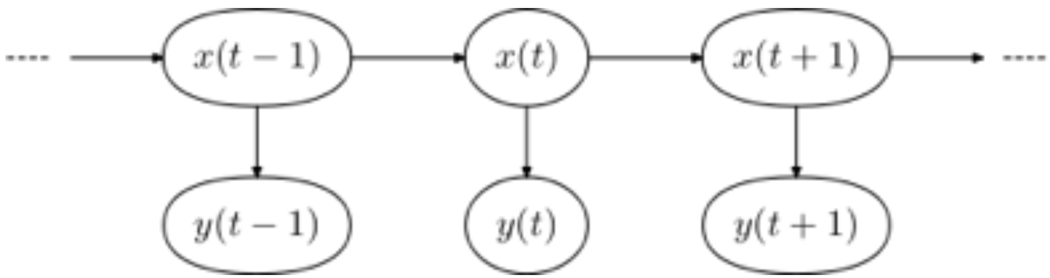
$z_i = F(x_i)$

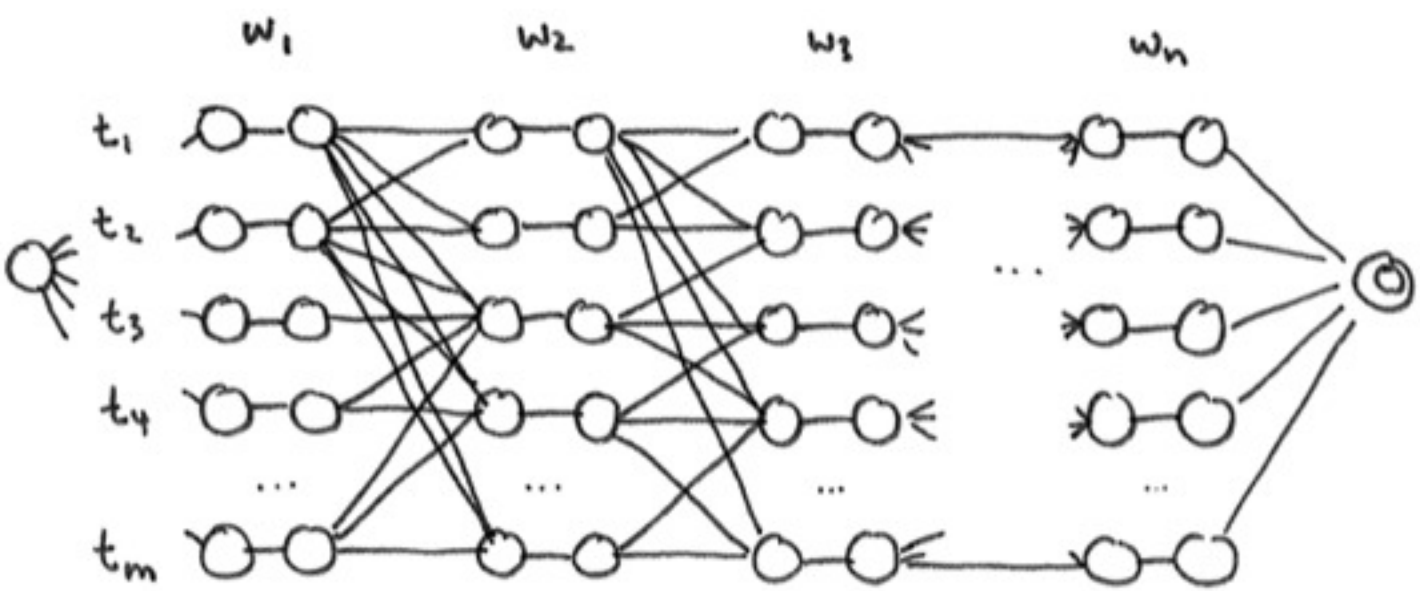
If $(z_i \neq y_i)$ $\boxed{\mathbf{W} \leftarrow \mathbf{W} + \mathbf{\Phi}(x_i, y_i) - \mathbf{\Phi}(x_i, z_i)}$

Output: Parameters $\mathbf{W}$

Inference: Dynamic Programming







Viterbi for argmax

Viterbi search for $\underset{t \cdots t}{\operatorname{argmax}} \; P(t \cdots t) \cdot P(w \cdots w \mid t \cdots t)$:

```
for j = 1 to m
   Q[1,j] = P(t_j) · P(w_1 | t_j)

for i = 2 to n
   for j = 1 to m
      Q[i,j] = 0
      best-pred[i,j] = 0
      best-score = -∞
      for k = 1 to m
         r = P(t_j | t_k) · P(w_i | t_j) · Q[i-1, k]
         if r > best-score
            best-score = r
            best-pred[i,j] = k
            Q[i,j] = r
final-best = 0
final-score = -∞
for j = 1 to m
   if Q[n,j] > final-score
      final-score = Q[n,j]
      final-best = j

print t_final-best
current = final-best
for i = n-1 down to 1
   current = best-prev[i+1, current]
   print t_current
```

Q[i,j] = cost of shortest path ending with word i getting assigned tag j.

sets back pointers

prints best tags in reverse order

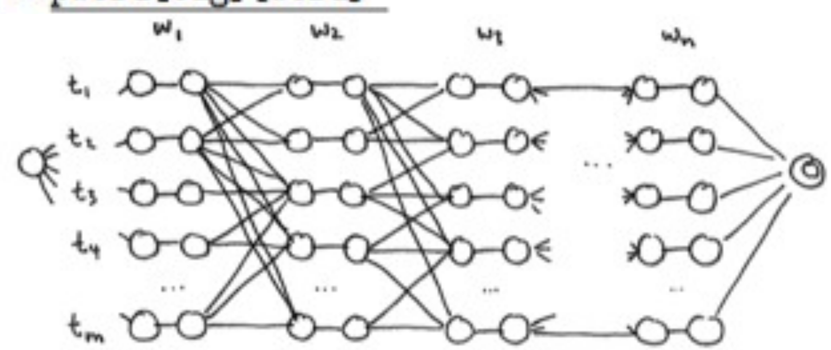
how about unigram?

Python implementation

Complete this Python code implementing the Viterbi algorithm for part-of-speech tagging. It should print a list of word/tag pairs, e.g. `[('a', 'D'), ('can', 'N'), ('can', 'A'), ('can', 'V'), ('a', 'D'), ('can', 'N')]`.

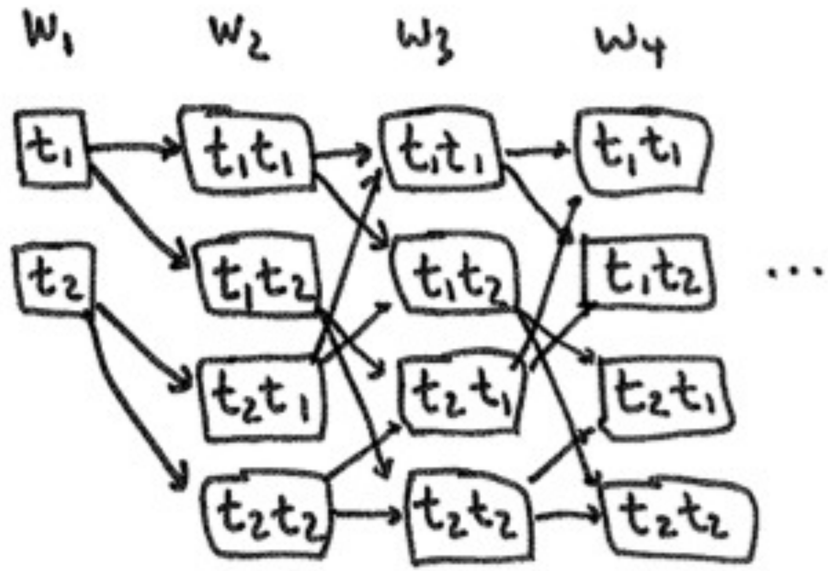
Q: what about top-down recursive + memoization?

```
from collections import defaultdict

best = defaultdict(lambda : defaultdict(float))
best[0]["<s>"] = 1
back = defaultdict(dict)

words = "<s> a can can can a can </s>".split()

tags = {"a": ["D"], "can": ["N", "A", "V"], "</s>": ["</s>"]}  # possible tags for each word
ptag = {"D": {"N": 1}, "V": {"</s>": 0.5, "D":0.5}, ... }      # ptag[x][y] = p(y | x)
pword = {"D": {"a": 0.5}, "N": {"can": 0.1}, ...        }      # pword[x][w] = p(w | x)

for i, word in enumerate(words[1:], 1):                        # i=1,2...; word=a,can,...
    for tag in tags[word]:
        for prev in best[i-1] :
            if tag in ptag[prev] :
                score = best[i-1][prev] * ptag[prev][tag] * pword[tag][word]
                if score > best[i][tag]:
                    best[i][tag] = score
                    back[i][tag] = prev

def backtrack(i, tag):
    if i == 0:
        return []
    return backtrack(i-1, back[i][tag]) + [(words[i], tag)]

print backtrack(len(words)-1, "</s>")[:-1]
```



Trigram HMM

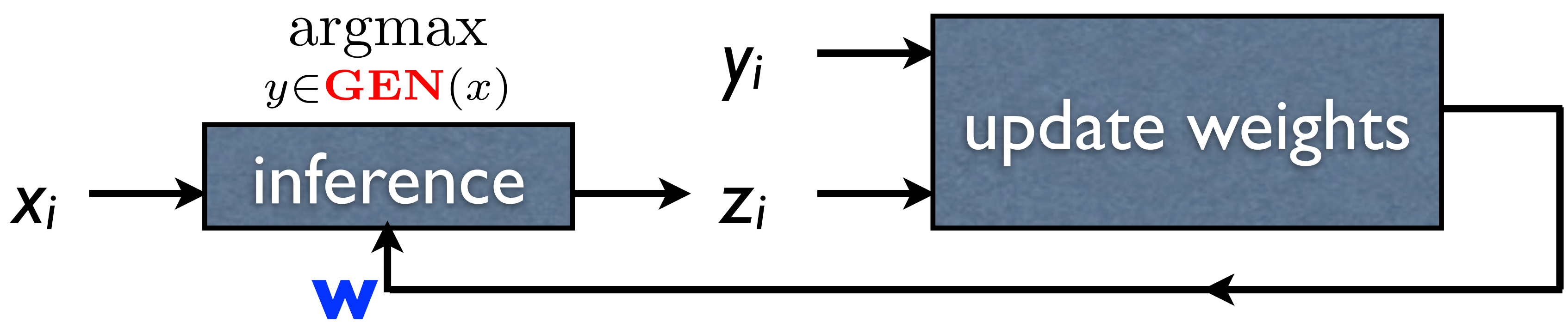
```
for j = 1 to m
    Q1[1, j] = ...

for j = 1 to m
    for j2 = 1 to m
        Q[2, j, j2] = ...

for i = 3 to n
    for j = 1 to m
        for j2 = 1 to m
            Q[i, j, j2] = 0
            best-pred[i, j, j2] = 0
            best-score = -∞
            for k = 1 to m
                r = P(t_j2 | t_k t_j) · P(w_i | t_j2) · Q[i-1, k, j]
                if r > best-score ...
```



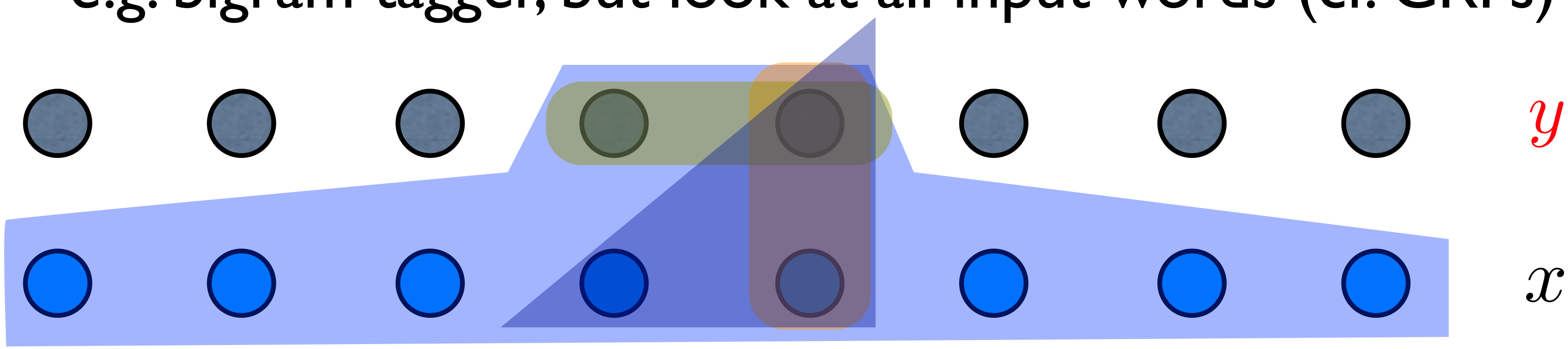
time complexity: $O(nT^3)$

in general: $O(nT^g)$ for g-gram

Efficiency vs. Expressiveness



- the inference (argmax) must be efficient
  - either the search space GEN(x) is small, or factored
  - features must be local to y (but can be global to x)
    - e.g. bigram tagger, but look at all input words (cf. CRFs)



Averaged Perceptron

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

Initialization: $\mathbf{W}_0 = 0$

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

Algorithm: For $t = 1 \dots T, i = 1 \dots n$

$z_i = F(x_i)$

If $(z_i \neq y_i)$ $\mathbf{W}_{j+1} \leftarrow \mathbf{W}_j + \mathbf{\Phi}(x_i, y_i) - \mathbf{\Phi}(x_i, z_i)$

Output: Parameters $\mathbf{W} = \sum_j \mathbf{W}_j$

- more stable and accurate results
- approximation of voted perceptron (Freund & Schapire, 1999)

Averaging Tricks

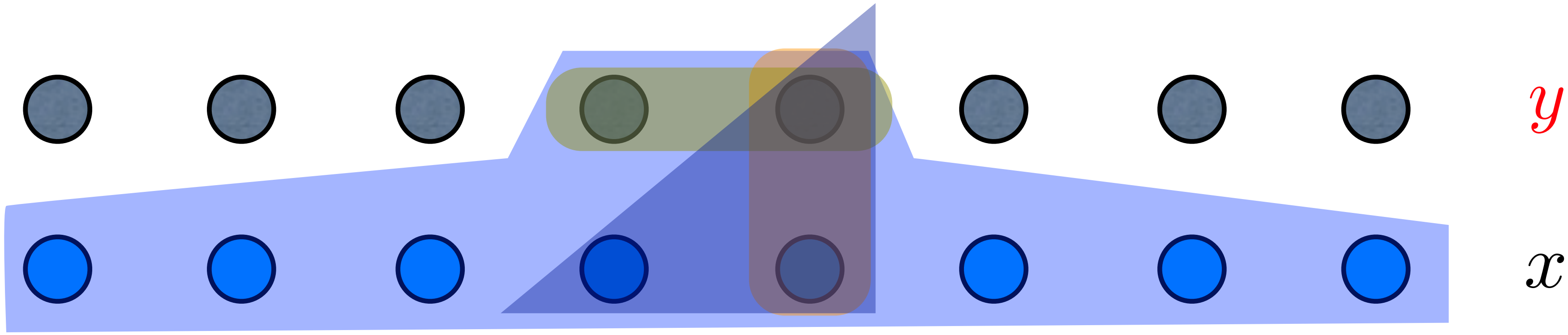
- Daume (2006, PhD thesis)

Algorithm $\textsc{AveragedStructuredPerceptron}(x_{1:N}, y_{1:N}, I)$

$\boldsymbol{w}_0 \leftarrow \langle 0, \ldots, 0 \rangle$
$\boldsymbol{w}_a \leftarrow \langle 0, \ldots, 0 \rangle$
$c \leftarrow 1$
for $i = 1 \ldots I$ **do**
  for $n = 1 \ldots N$ **do**
    $\hat{y}_n \leftarrow \arg\max_{y \in \mathcal{Y}} \boldsymbol{w}_0{}^\top \Phi(x_n, y_n)$
    if $y_n \neq \hat{y}_n$ **then**
      $\boldsymbol{w}_0 \leftarrow \boldsymbol{w}_0 + \Phi(x_n, y_n) - \Phi(x_n, \hat{y}_n)$
      $\boldsymbol{w}_a \leftarrow \boldsymbol{w}_a + c\Phi(x_n, y_n) - c\Phi(x_n, \hat{y}_n)$
    end if
    $c \leftarrow c + 1$
  end for
end for
return $\boldsymbol{w}_0 - \boldsymbol{w}_a / c$

Figure 2.3: The averaged structured perceptron learning algorithm.

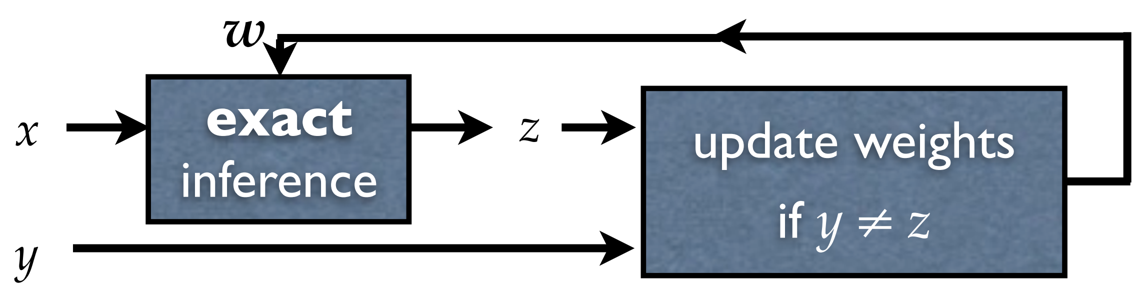
Do we need smoothing?

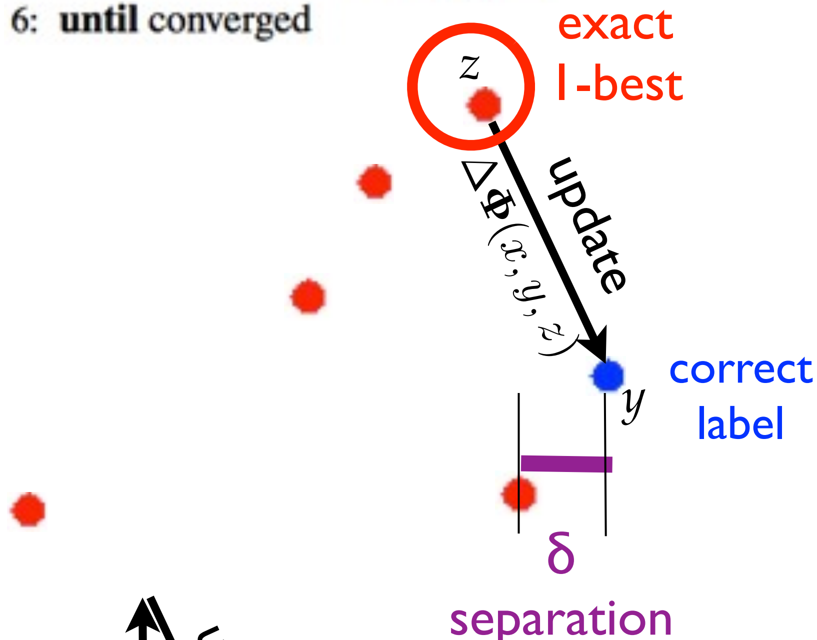


- smoothing is much easier in discriminative models
- just make sure for each feature template, its subset templates are also included
  - e.g., to include (t_0 w_0 w_{-1}) you must also include
  - (t_0 w_0) (t_0 w_{-1}) (w_0 w_{-1})
  - and maybe also (t_0 t_{-1}) because t is less sparse than w

Geometry of Convergence Proof pt 1

repeat
  for each example (x, y) **in** D **do**
    $z \leftarrow \textsc{Exact}(x, \mathbf{w})$
    if $z \neq y$ **then**
      $\mathbf{w} \leftarrow \mathbf{w} + \Delta\Phi(x, y, z)$
until converged

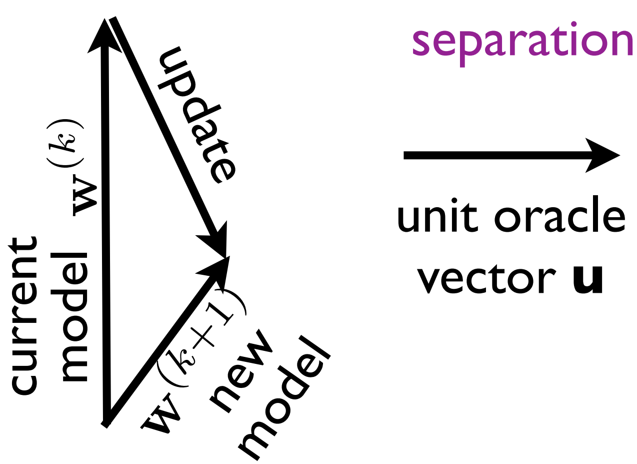




perceptron update:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \Delta\Phi(x, y, z)$$

$$\mathbf{u}\cdot\mathbf{w}^{(k+1)} = \mathbf{u}\cdot\mathbf{w}^{(k)} + \boxed{\mathbf{u}\cdot\Delta\Phi(x, y, z) \geq \delta \quad \text{margin}}$$



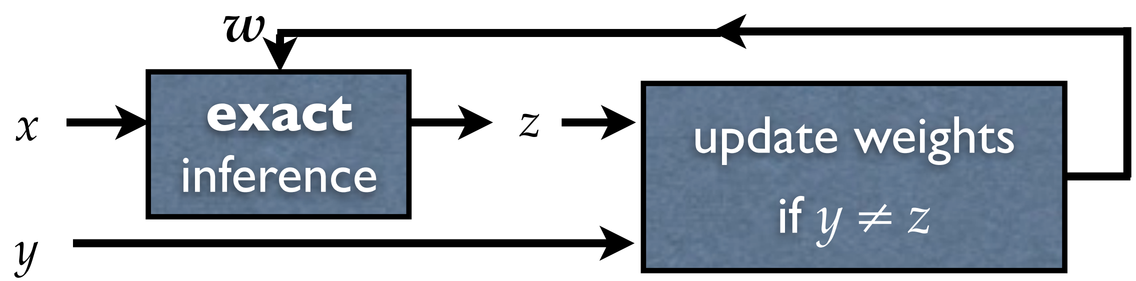
$$\mathbf{u}\cdot\mathbf{w}^{(k+1)} \geq k\delta \qquad \text{(by induction)}$$

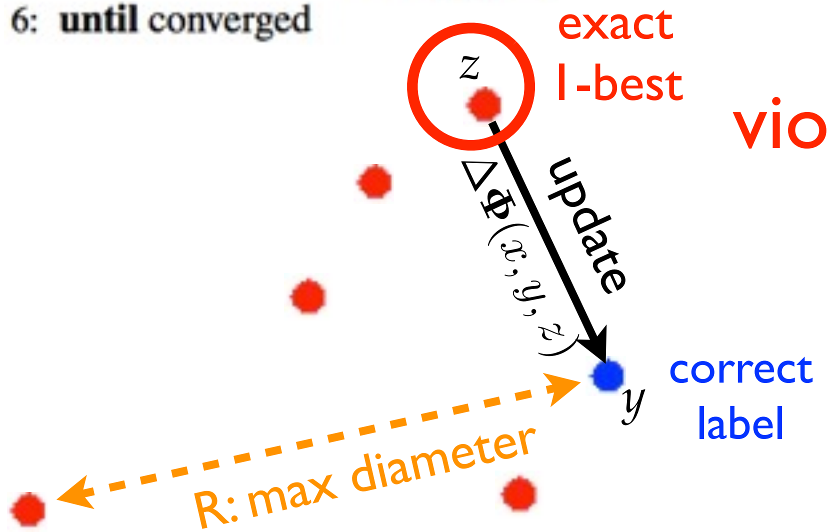
$$\|\mathbf{u}\|\|\mathbf{w}^{(k+1)}\| \geq \mathbf{u}\cdot\mathbf{w}^{(k+1)} \geq k\delta$$

$$\|\mathbf{w}^{(k+1)}\| \geq k\delta \qquad \text{(part 1: upperbound)}$$

Geometry of Convergence Proof pt 2

```
repeat
    for each example (x, y) in D do
        z ← EXACT(x, w)
        if z ≠ y then
            w ← w + ΔΦ(x, y, z)
until converged
```





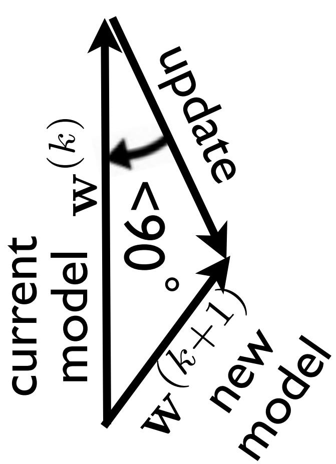
violation: incorrect label scored higher

perceptron update:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \Delta\mathbf{\Phi}(x, y, z)$$

$$\|\mathbf{w}^{(k+1)}\|^2 = \|\mathbf{w}^{(k)} + \Delta\mathbf{\Phi}(x, y, z)\|^2$$

$$= \|\mathbf{w}^{(k)}\|^2 + \underbrace{\|\Delta\mathbf{\Phi}(x, y, z)\|^2}_{\leq R^2 \text{ diameter}} + \underbrace{2\, \mathbf{w}^{(k)} \cdot \Delta\mathbf{\Phi}(x, y, z)}_{\leq 0 \text{ violation}}$$



by induction: $\|\mathbf{w}^{(k+1)}\|^2 \leq kR^2$ (part 2: upperbound)

parts 1+2 => update bounds: $\boxed{k \leq R^2/\delta^2}$

Experiments

Experiments: Tagging

- (almost) identical features from (Ratnaparkhi, 1996)
  - trigram tagger: current tag t_i, previous tags t_{i-1}, t_{i-2}
  - current word w_i and its spelling features
  - surrounding words w_{i-1} w_{i+1} w_{i-2} w_{i+2}..

Method	Error rate/%	Numits
Perc, avg, cc=0	**2.93**	10
Perc, noavg, cc=0	3.68	20
Perc, avg, cc=5	3.03	6
Perc, noavg, cc=5	4.04	17
ME, cc=0	3.4	100
ME, cc=5	**3.28**	200

Experiments: NP Chunking

- B-I-O scheme

Rockwell International Corp.

B I I

's Tulsa unit said it signed

B I I O B O

a tentative agreement ...

B I I

- features:
  - unigram model
  - surrounding words and POS tags

Current word	w_i	$\& \; t_i$
Previous word	w_{i-1}	$\& \; t_i$
Word two back	w_{i-2}	$\& \; t_i$
Next word	w_{i+1}	$\& \; t_i$
Word two ahead	w_{i+2}	$\& \; t_i$
Bigram features	w_{i-2}, w_{i-1}	$\& \; t_i$
	w_{i-1}, w_i	$\& \; t_i$
	w_i, w_{i+1}	$\& \; t_i$
	w_{i+1}, w_{i+2}	$\& \; t_i$
Current tag	p_i	$\& \; t_i$
Previous tag	p_{i-1}	$\& \; t_i$
Tag two back	p_{i-2}	$\& \; t_i$
Next tag	p_{i+1}	$\& \; t_i$
Tag two ahead	p_{i+2}	$\& \; t_i$
Bigram tag features	p_{i-2}, p_{i-1}	$\& \; t_i$
	p_{i-1}, p_i	$\& \; t_i$
	p_i, p_{i+1}	$\& \; t_i$
	p_{i+1}, p_{i+2}	$\& \; t_i$
Trigram tag features	p_{i-2}, p_{i-1}, p_i	$\& \; t_i$
	p_{i-1}, p_i, p_{i+1}	$\& \; t_i$
	p_i, p_{i+1}, p_{i+2}	$\& \; t_i$

Experiments: NP Chunking

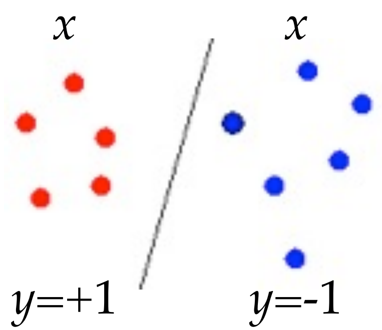
- results

Method	F-Measure	Numits
Perceptron, avg, cc=0	**93.53**	13
Perceptron, noavg, cc=0	93.04	35
Perceptron, avg, cc=5	93.33	9
Perceptron, noavg, cc=5	91.88	39
Max-ent, cc=0	92.34	900
Max-ent, cc=5	**92.65**	200

- (Sha and Pereira, 2003) trigram tagger
  - voted perceptron: 94.09% vs. CRF: 94.38%

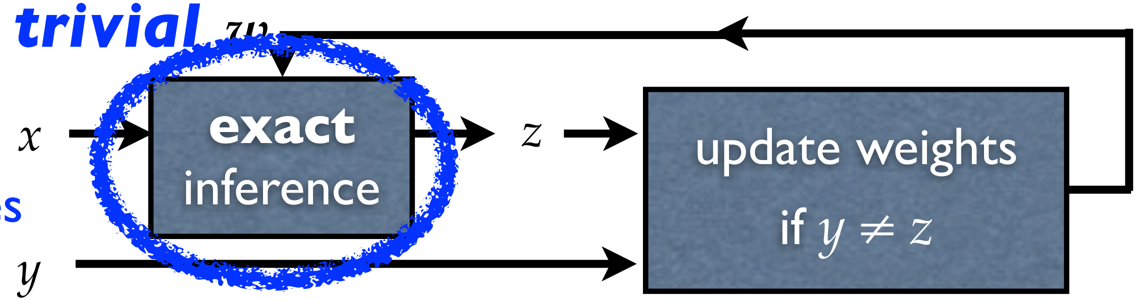
Structured Perceptron (Collins 02)

binary classification



trivial

constant
# of classes

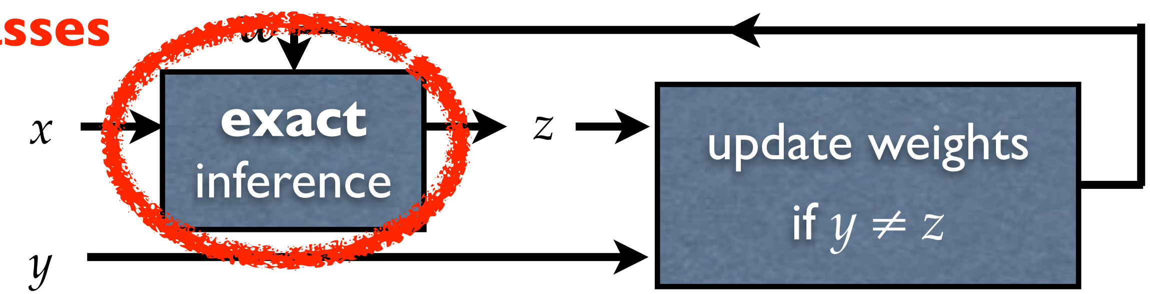


structured classification

exponential
# of classes

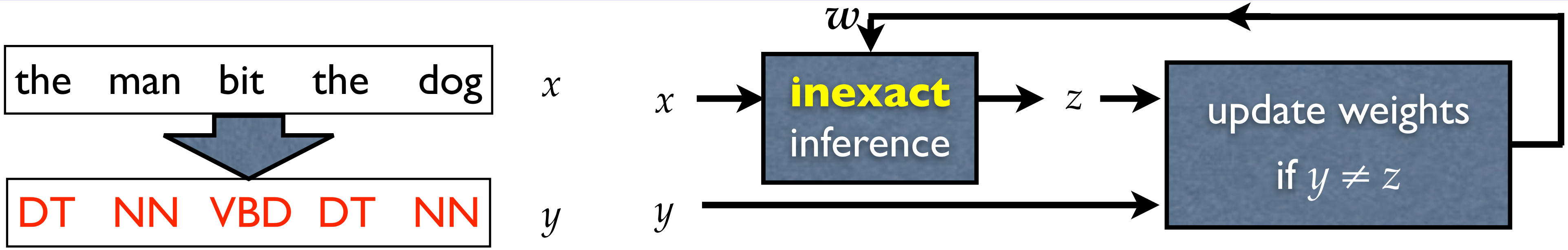
hard

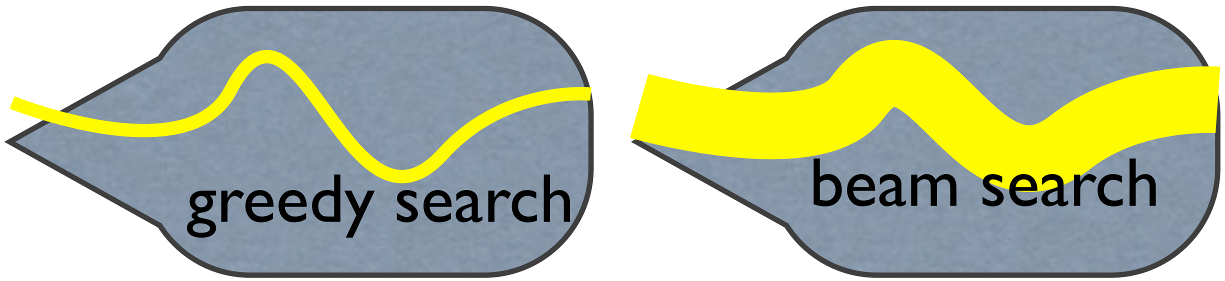
x	the	man	bit	the	dog
y	DT	NN	VBD	DT	NN



- challenge: search efficiency (exponentially many classes)
  - often use dynamic programming (DP)
  - but still too slow for repeated use, e.g. parsing is $O(n^3)$
  - and can't use non-local features in DP

Learning w/ Inexact Inference (Huang et al 2012)

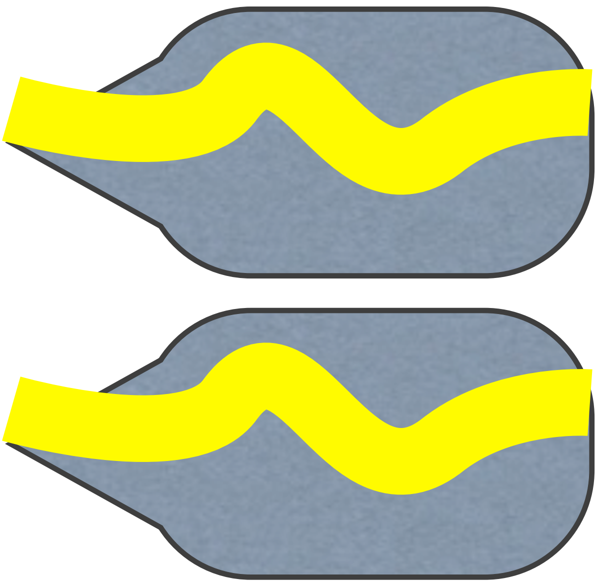


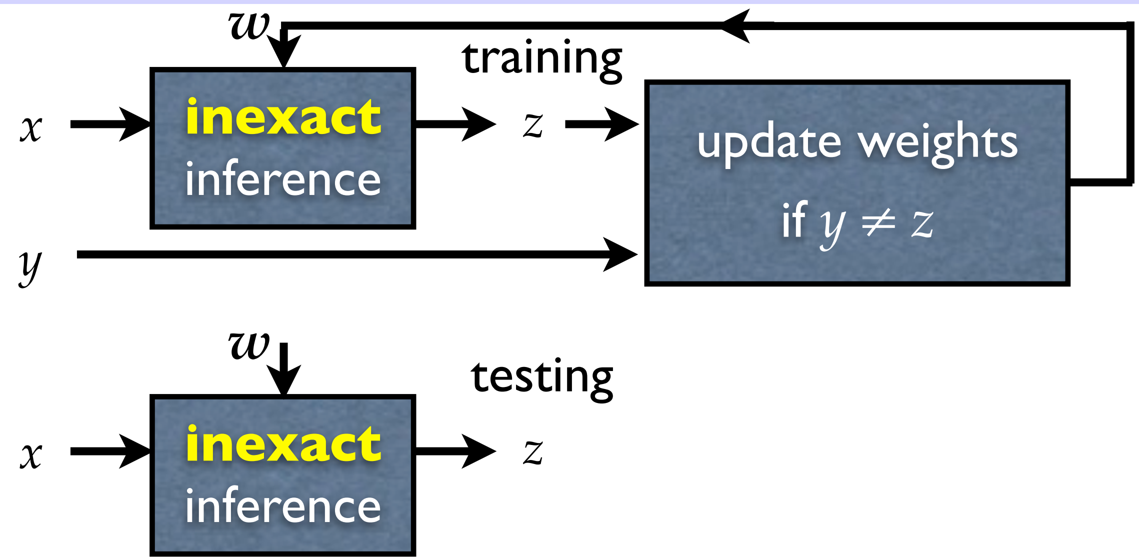


does it still work???

- routine use of inexact inference in NLP (e.g. beam search)
- how does structured perceptron work with inexact search?
  - so far most structured learning theory assume exact search
  - would search errors break these learning properties?
  - if so how to modify learning to accommodate inexact search?

Idea: Search-Error-Robust Model

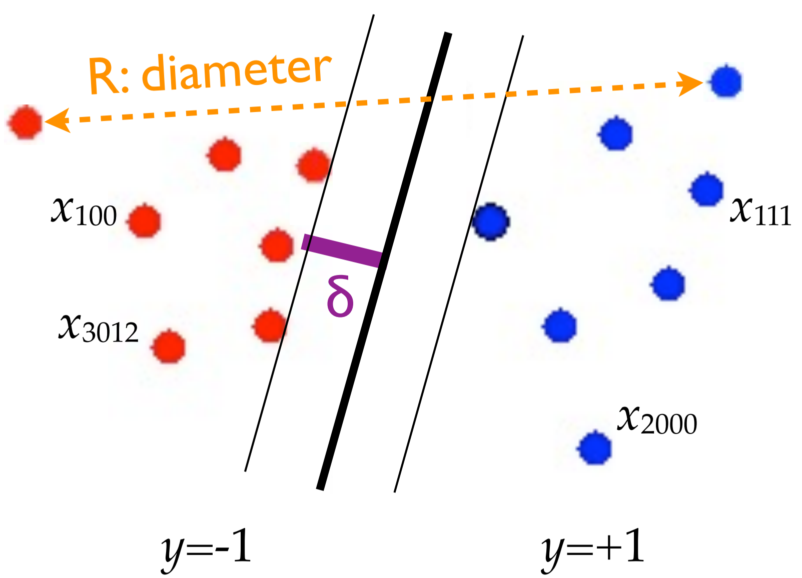




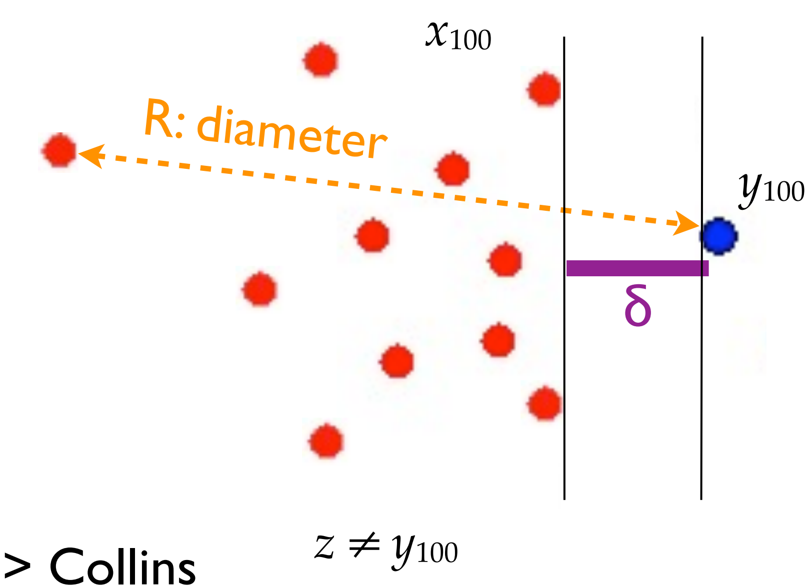
- train a "search-specific" or "search-error-robust" model
  - we assume the same "search box" in training and testing
  - model should "live with" search errors from search box
- exact search => convergence; greedy => no convergence
  - how can we make perceptron converge w/ greedy search?

Convergence with Exact Search

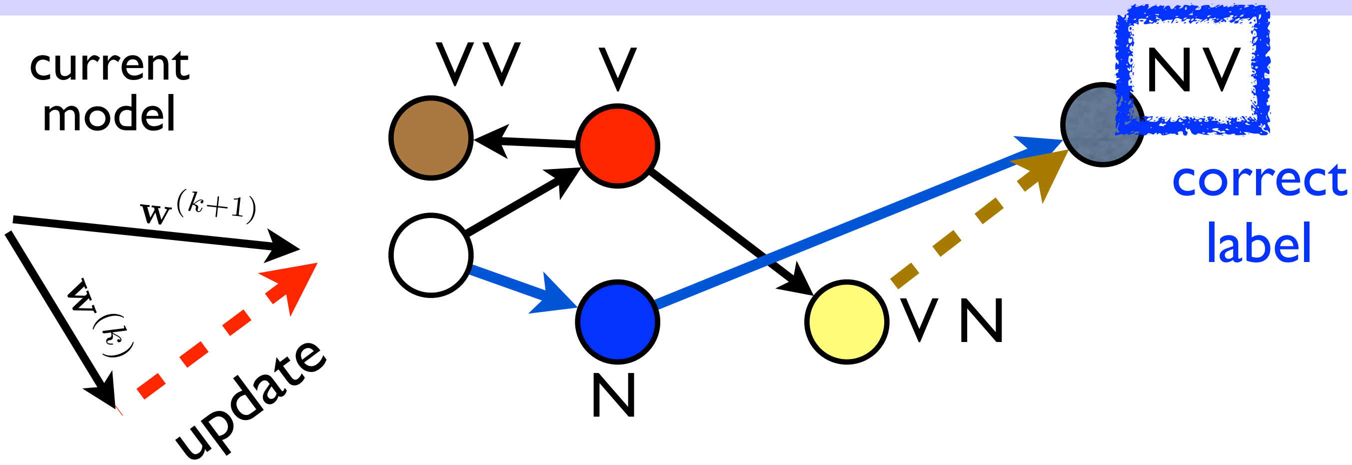
- linear classification: converges iff. data is separable
- structured: converges iff. data separable & search exact
  - there is an oracle vector that correctly labels all examples
  - one vs the rest (correct label better than all incorrect labels)
- theorem: if separable, then **# of updates ≤ R^2 / δ^2** R: diameter



Rosenblatt => Collins
1957 2002



Convergence with Exact Search



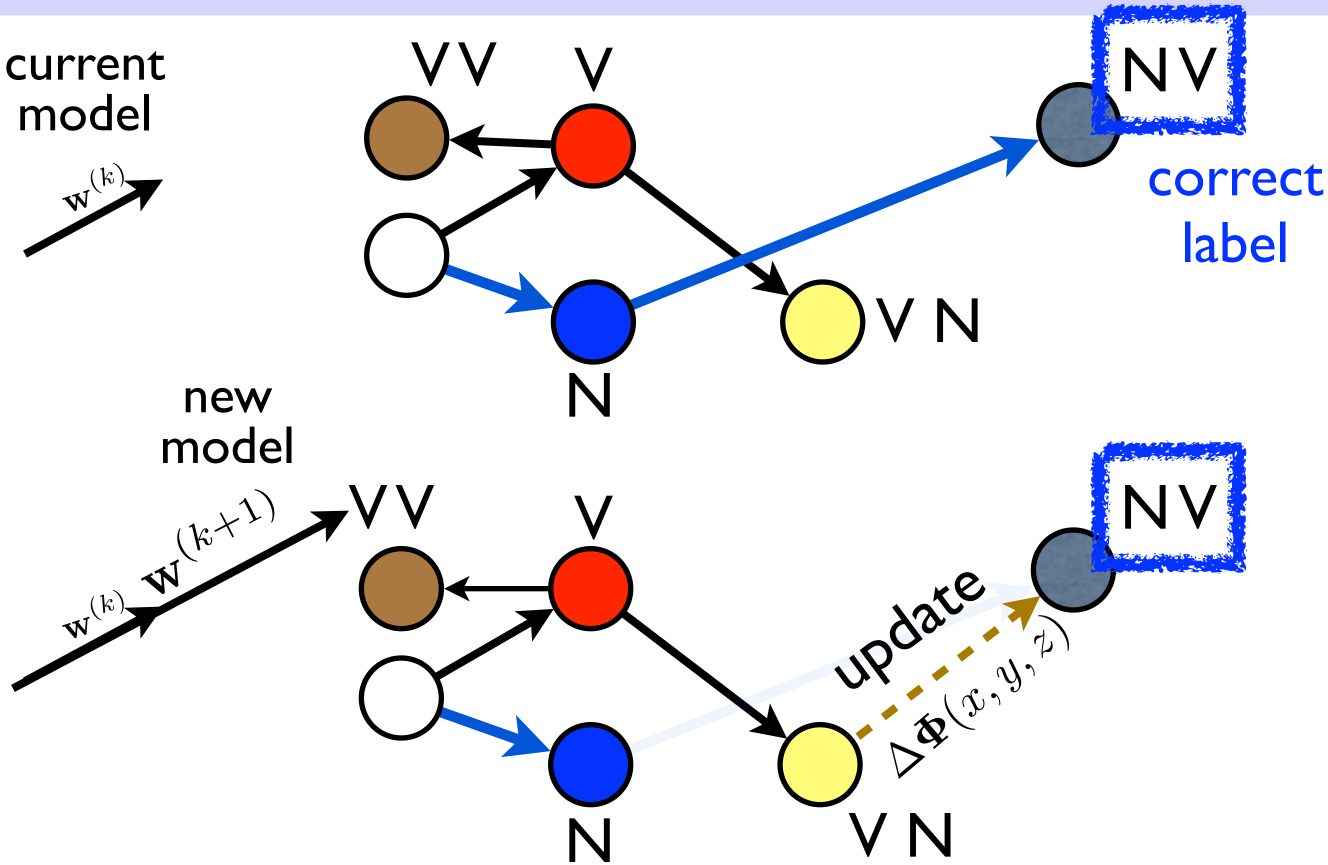
training example

time flies
N V

output space
{N,V} x {N,V}

standard perceptron converges with exact search

No Convergence w/ Greedy Search



training example

time flies

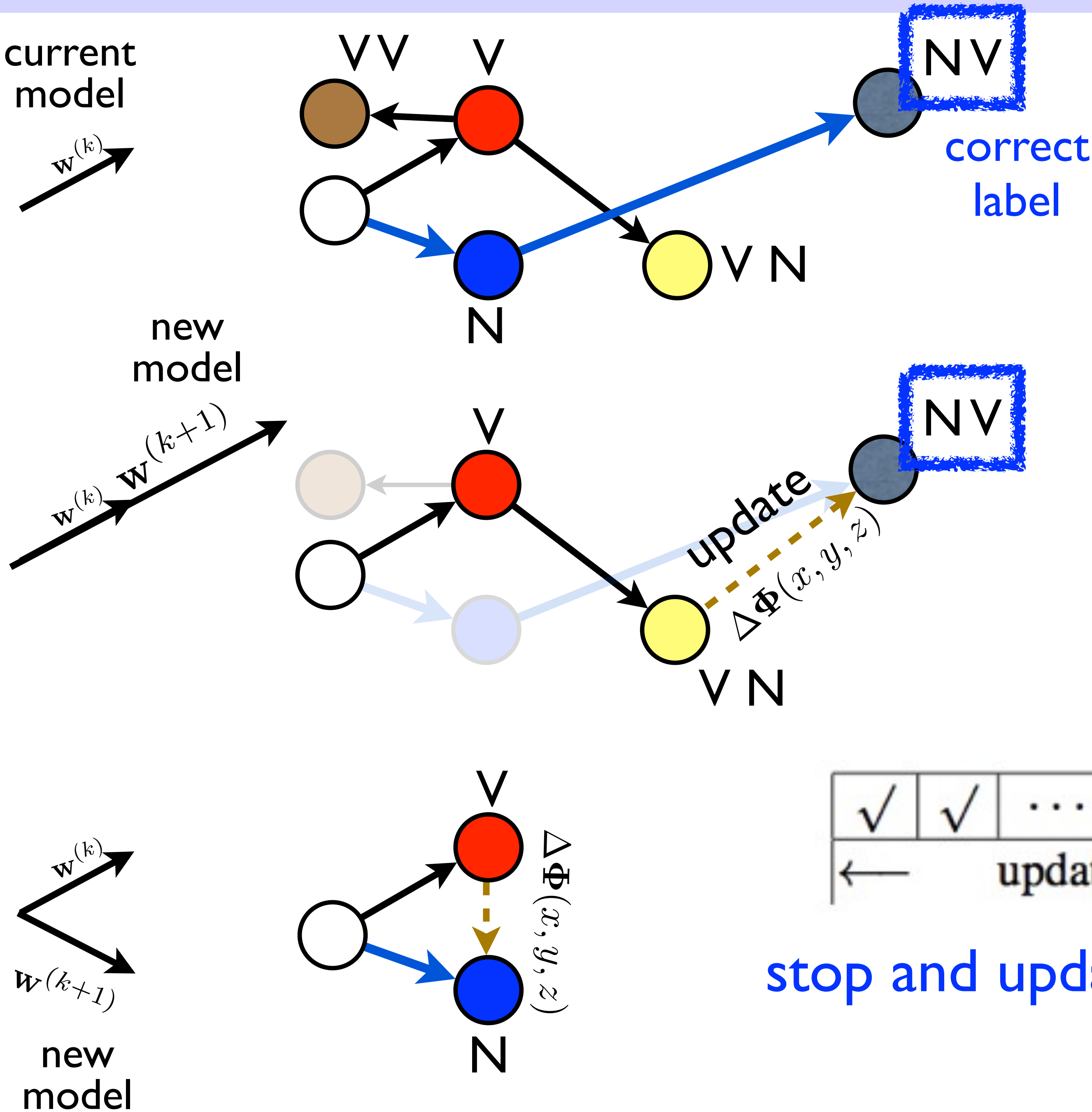
N V

output space

{N,V} x {N,V}

standard perceptron does not converge with greedy search

Early update (Collins/Roark 2004) to rescue



training example

time	flies
N	V

output space

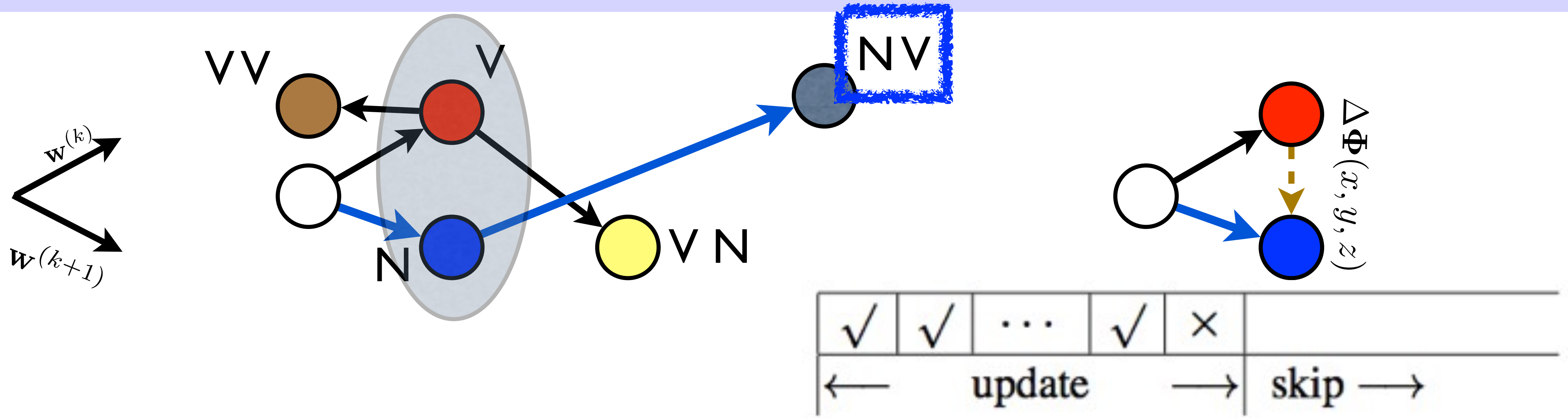
{N,V} x {N,V}

standard perceptron does not converge with greedy search

√	√	⋯	√	×	
⟵	update		⟶	skip ⟶	

stop and update at the first mistake

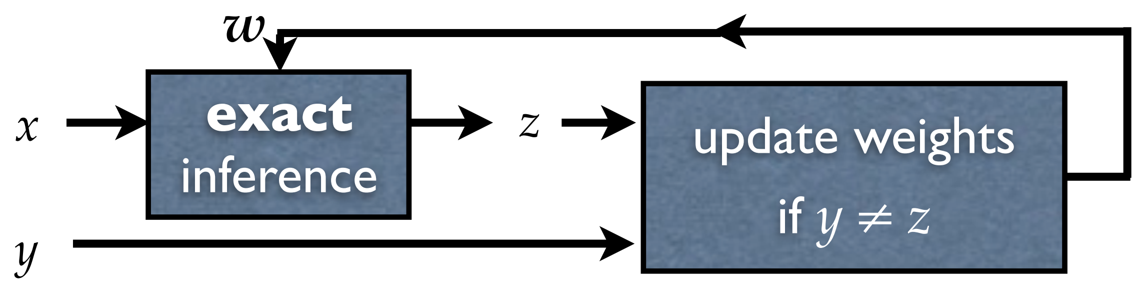
Why?

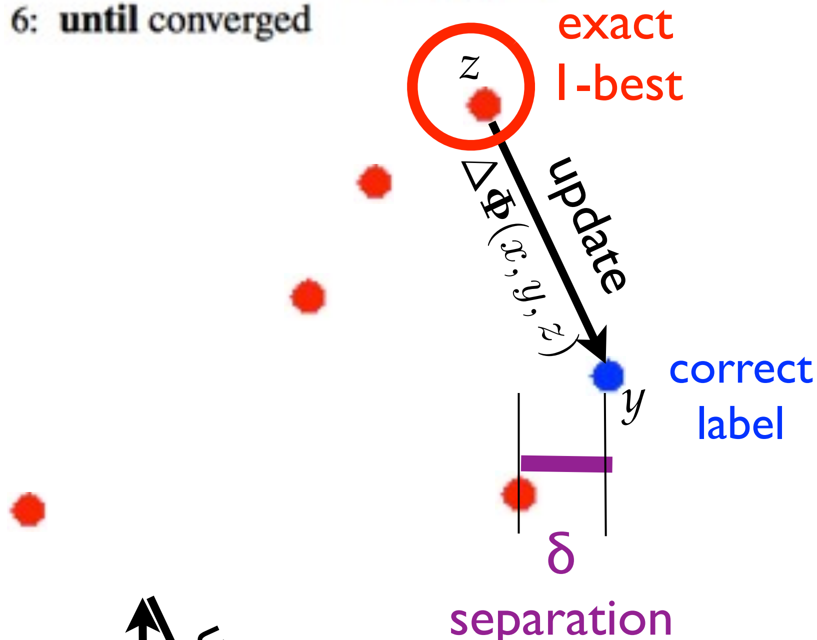


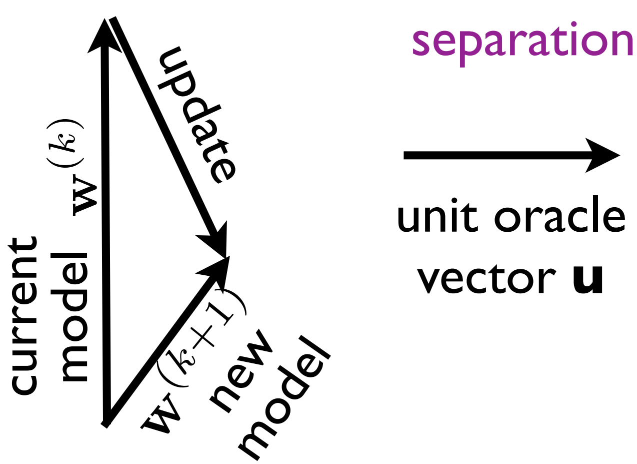
- why does inexact search break convergence property?
  - what is required for convergence? exactness?
- why does early update (Collins/Roark 04) work?
  - it works well in practice and is now a standard method
  - but there has been no theoretical justification
- we answer these Qs by inspecting the convergence proof

Geometry of Convergence Proof pt 1

```
repeat
    for each example (x, y) in D do
        z ← EXACT(x, w)
        if z ≠ y then
            w ← w + ΔΦ(x, y, z)
until converged
```







perceptron update:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \Delta\mathbf{\Phi}(x, y, z)$$

$$\mathbf{u} \cdot \mathbf{w}^{(k+1)} = \mathbf{u} \cdot \mathbf{w}^{(k)} + \boxed{\mathbf{u} \cdot \Delta\mathbf{\Phi}(x, y, z) \geq \delta}$$

margin

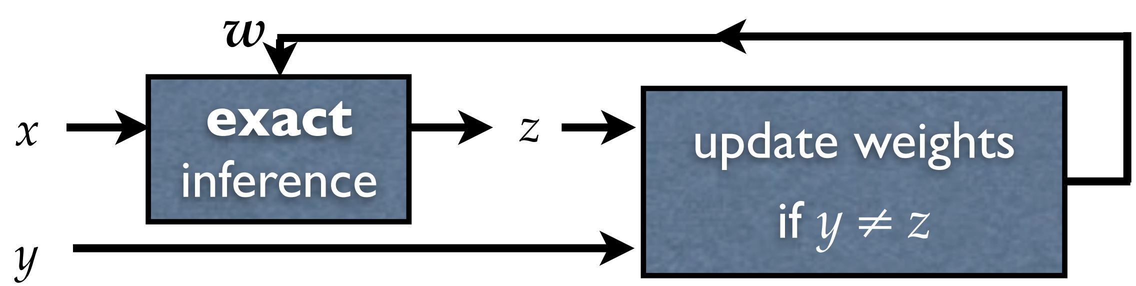
$$\mathbf{u} \cdot \mathbf{w}^{(k+1)} \geq k\delta$$ (by induction)

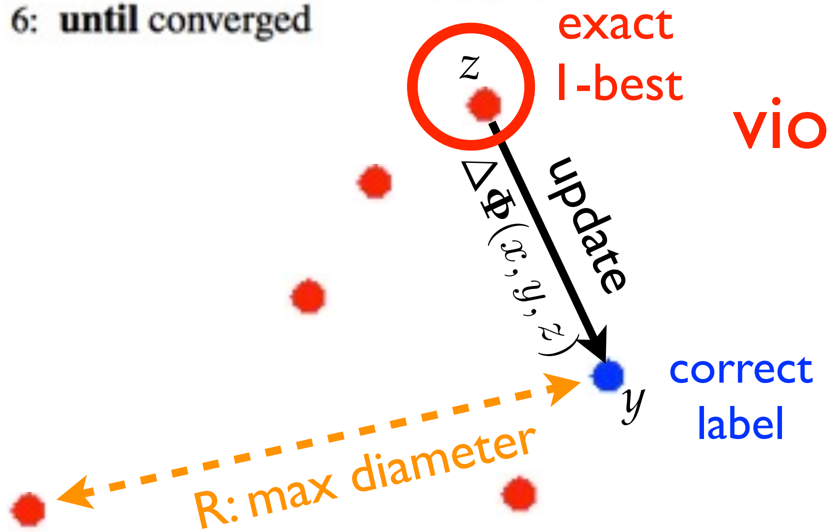
$$\|\mathbf{u}\|\|\mathbf{w}^{(k+1)}\| \geq \mathbf{u} \cdot \mathbf{w}^{(k+1)} \geq k\delta$$

$$\|\mathbf{w}^{(k+1)}\| \geq k\delta$$ (part 1: upperbound)

Geometry of Convergence Proof pt 2

repeat
  for each example (x, y) **in** D **do**
    $z \leftarrow \textsc{Exact}(x, \mathbf{w})$
    if $z \neq y$ **then**
      $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{\Phi}(x, y, z)$
until converged





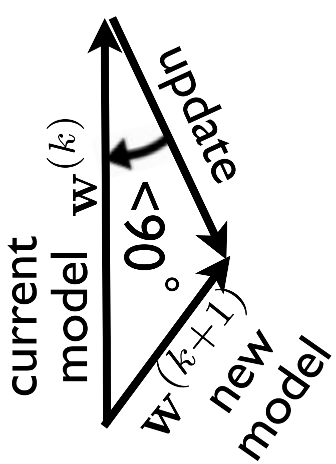
violation: incorrect label scored higher

perceptron update:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \Delta\mathbf{\Phi}(x, y, z)$$

$$\|\mathbf{w}^{(k+1)}\|^2 = \|\mathbf{w}^{(k)} + \Delta\mathbf{\Phi}(x, y, z)\|^2$$

$$= \|\mathbf{w}^{(k)}\|^2 + \underbrace{\|\Delta\mathbf{\Phi}(x, y, z)\|^2}_{\leq R^2 \text{ diameter}} + 2\,\underbrace{\mathbf{w}^{(k)} \cdot \Delta\mathbf{\Phi}(x, y, z)}_{\leq 0 \text{ violation}}$$

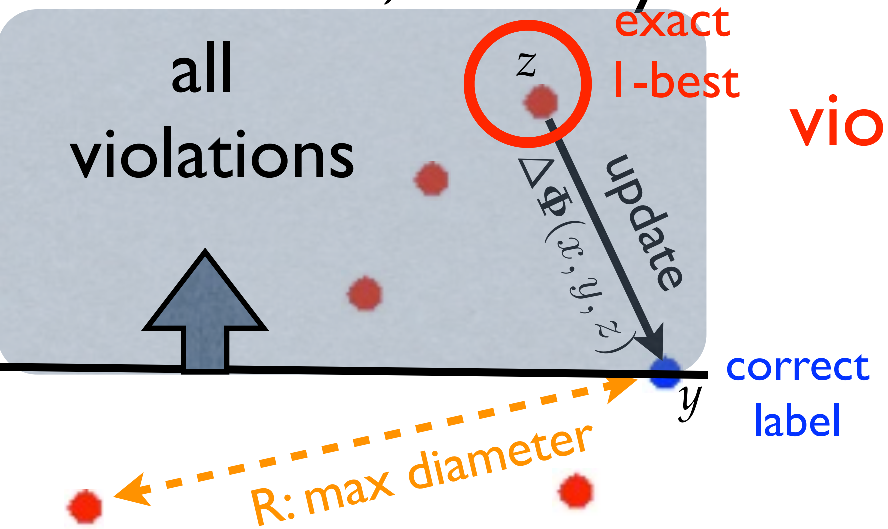


by induction: $\|\mathbf{w}^{(k+1)}\|^2 \leq kR^2$ (part 2: upperbound)

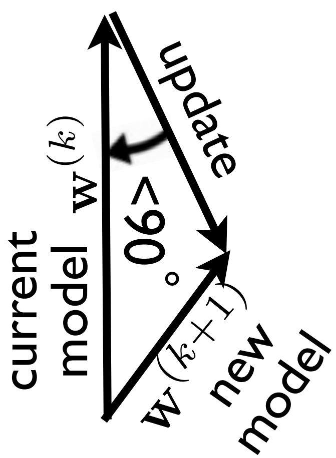
parts 1+2 => update bounds: $\boxed{k \leq R^2/\delta^2}$

Violation is All we need!

- exact search is **not** really required by the proof
  - rather, it is only used to ensure violation!



violation: incorrect label scored higher

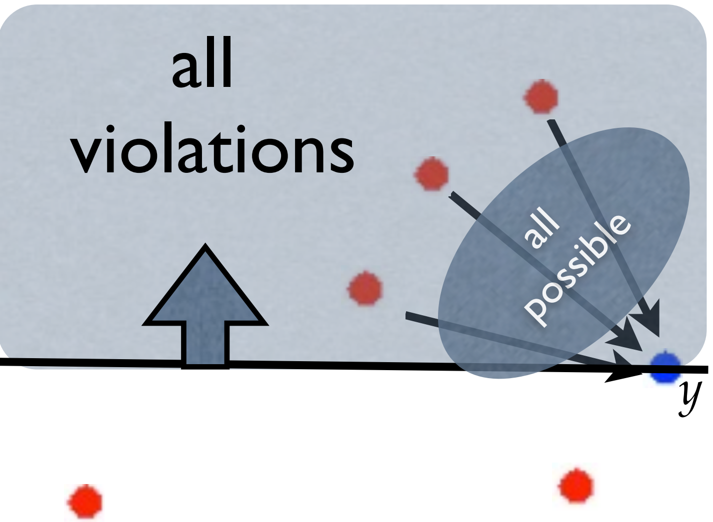


the proof only uses 3 facts:

1. separation (margin)
2. diameter (always finite)
3. violation (but no need for exact)

Violation-Fixing Perceptron

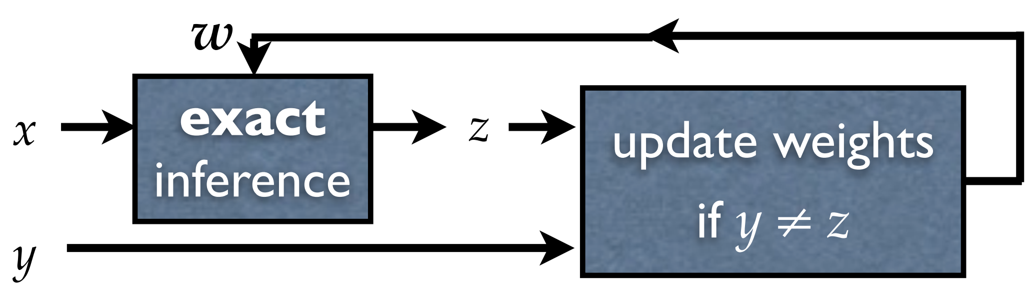
- if we guarantee violation, we don't care about exactness!
  - violation is good b/c we can at least fix a mistake



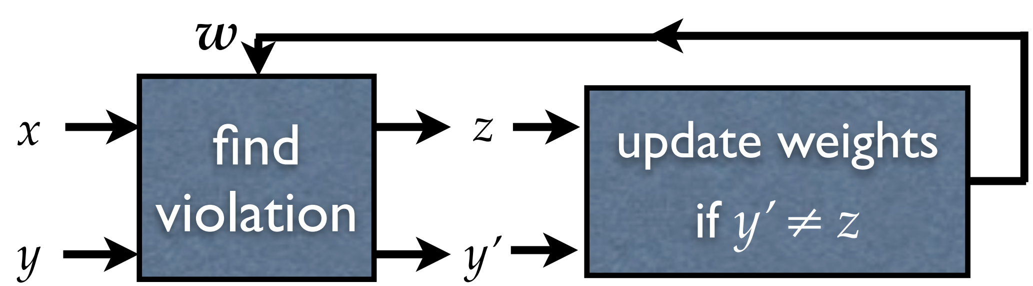
same mistake bound as before!

repeat
  for each example (x, y) **in** D **do**
    $(x, y', z) = \textsc{FindViolation}(x, y, \mathbf{w})$
    if $z \neq y$ **then** $\triangleright$ (x, y', z) is a viol
      $\mathbf{w} \leftarrow \mathbf{w} + \Delta\Phi(x, y', z)$
until converged

standard perceptron

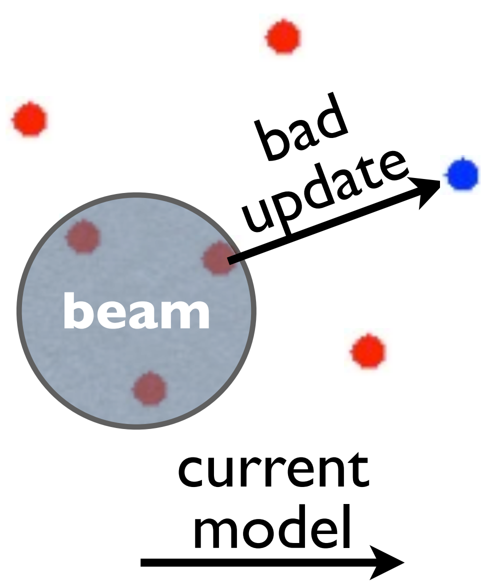


violation-fixing perceptron

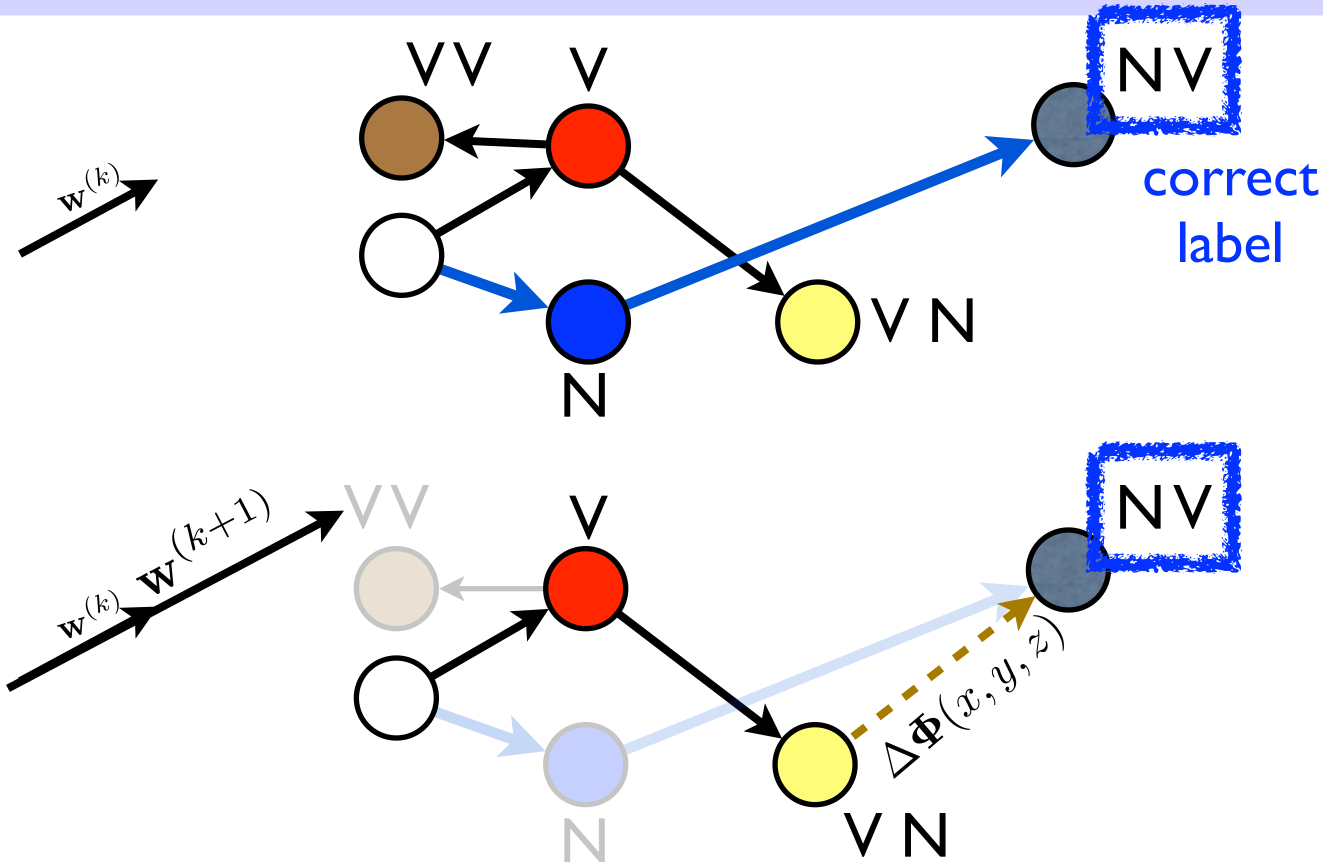


What if can’t guarantee violation

- this is why perceptron doesn’t work well w/ inexact search
  - because not every update is guaranteed to be a violation
  - thus the proof breaks; no convergence guarantee
- example: beam or greedy search
  - the model might prefer the correct label (if exact search)
  - but the search prunes it away
  - such a non-violation update is “bad” because it doesn’t fix any mistake
  - the new model still misguides the search



Standard Update: No Guarantee



training example

time flies

N V

output space

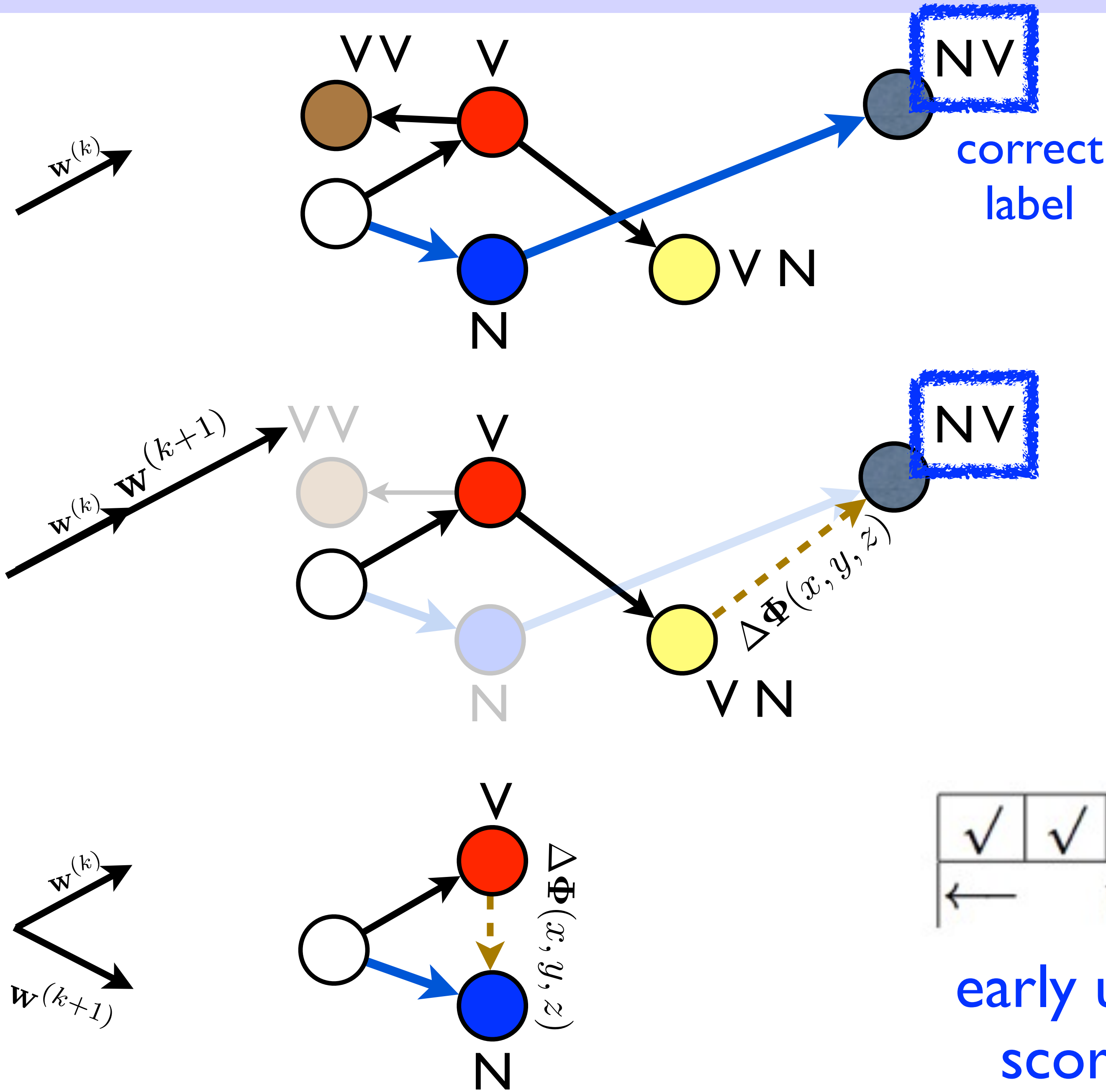
{N,V} x {N,V}

standard update doesn't converge b/c it doesn't guarantee violation

correct label scores higher.

non-violation: bad update!

Early Update: Guarantees Violation



training example

time	flies
N	V

output space

{N,V} x {N,V}

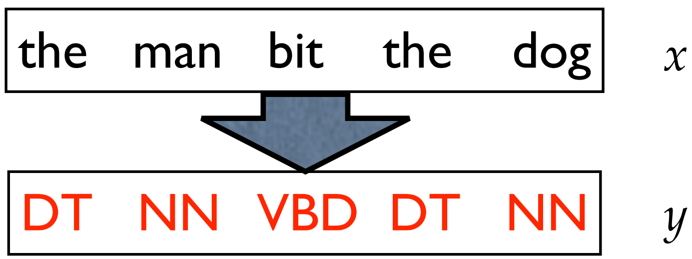
standard update doesn't converge b/c it doesn't guarantee violation

√	√	···	√	×	
←	update			→	skip →

early update: incorrect prefix scores higher: a violation!

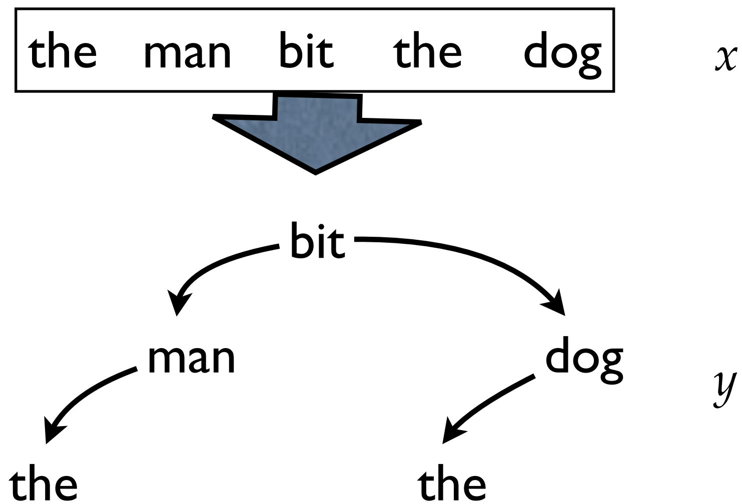
Experiments

trigram part-of-speech tagging



local features only,
exact search tractable
(proof of concept)

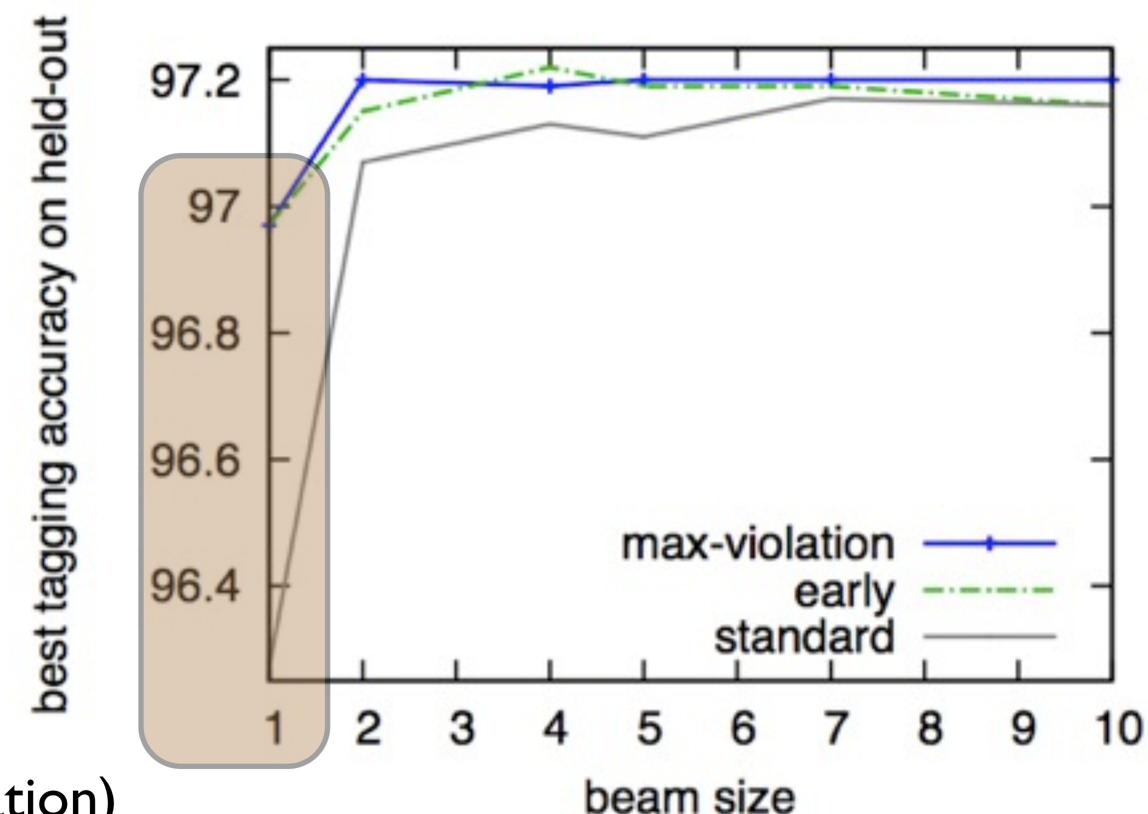
incremental dependency parsing



non-local features,
exact search intractable
(real impact)

1) Trigram Part of Speech Tagging

- standard update performs terribly with greedy search (b=1)
  - because search error is severe at b=1: half updates are bad!
  - no real difference beyond b=2: search error becomes rare

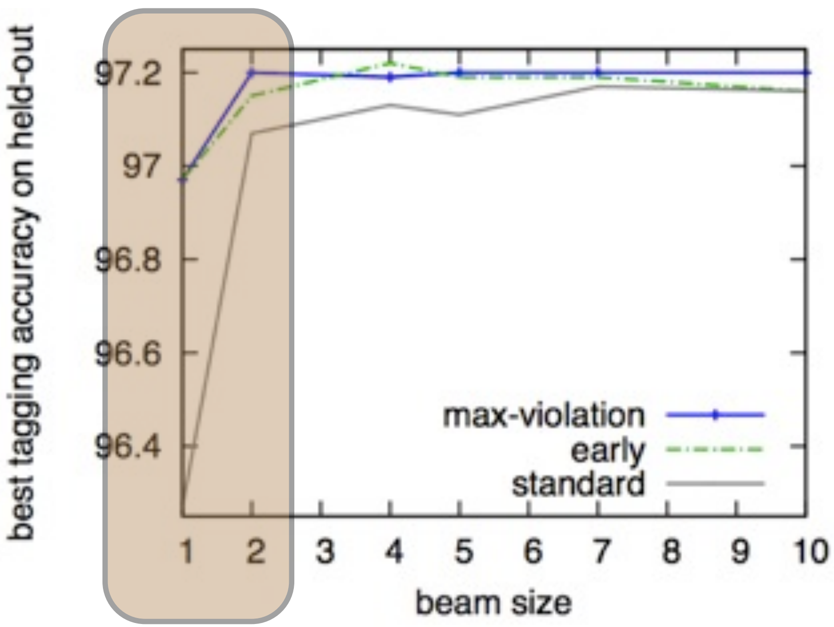


% of bad (non-violation) standard updates

53% 10% 1.5% 0.5%

Max-Violation Reduces Training Time

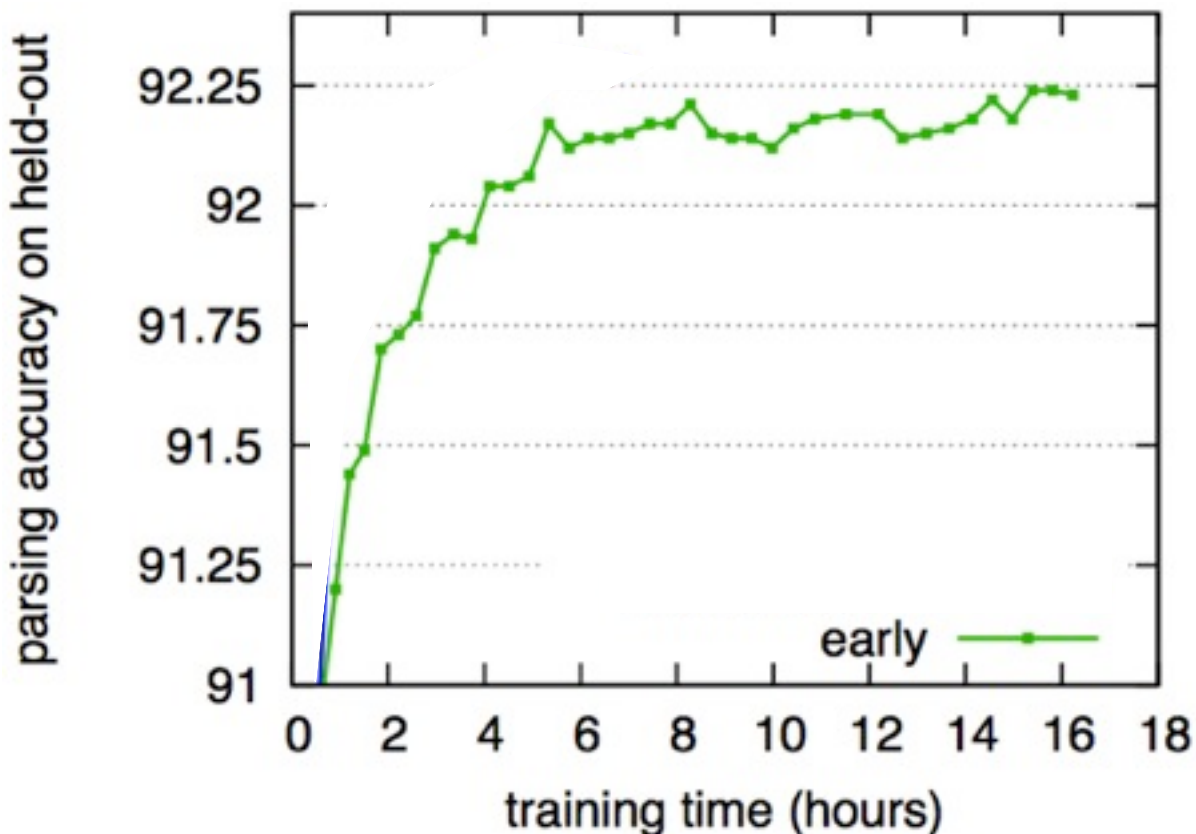
- max-violation peaks at b=2, greatly reduced training time
- early update achieves the highest dev/test accuracy
  - comparable to best published accuracy (Shen et al '07)
- future work: add non-local features to tagging



	beam	iter	time	test
standard	-	6	162m	97.28
early	4	6	37m	97.27
max-violation	2	3	26m	97.27
Shen et al (2007)				97.33

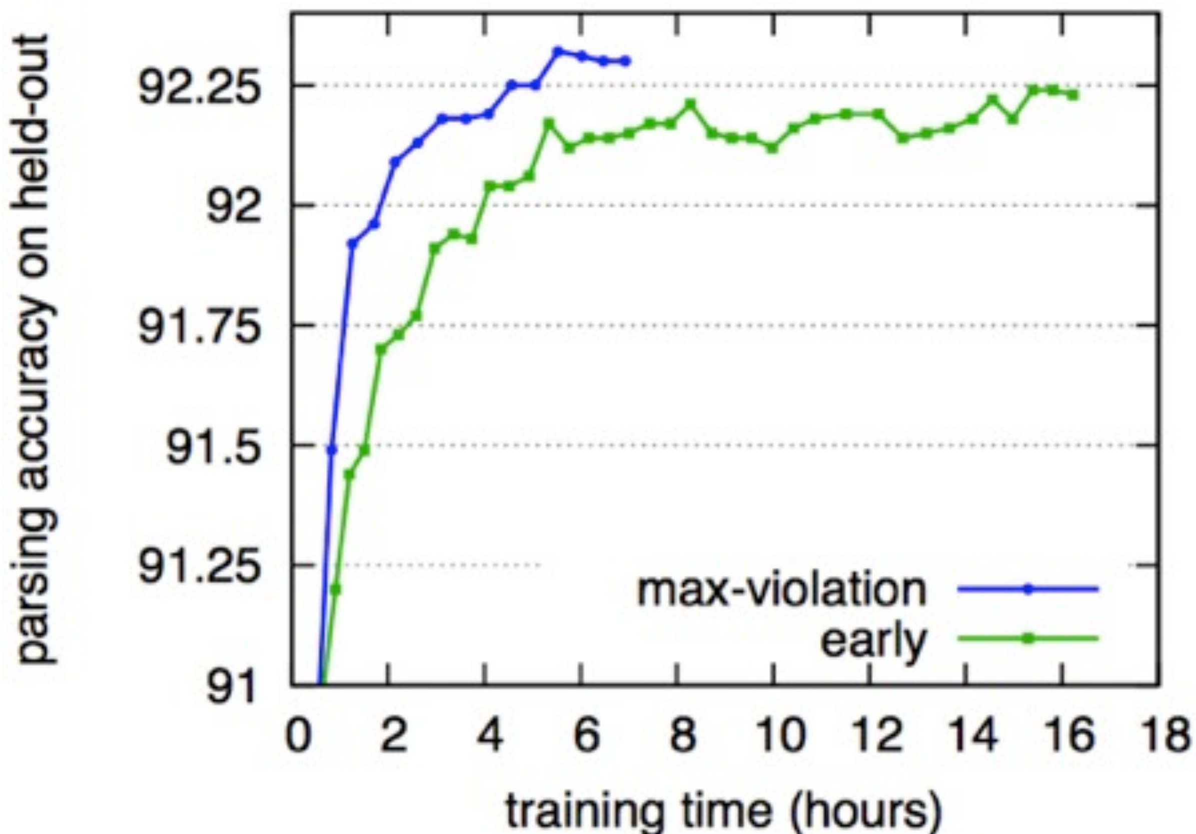
2) Incremental Dependency Parsing

- DP incremental dependency parser (Huang and Sagae 2010)
- non-local history-based features rule out exact DP
  - we use beam search, and search error is severe
  - baseline: early update. extremely slow: 38 iterations



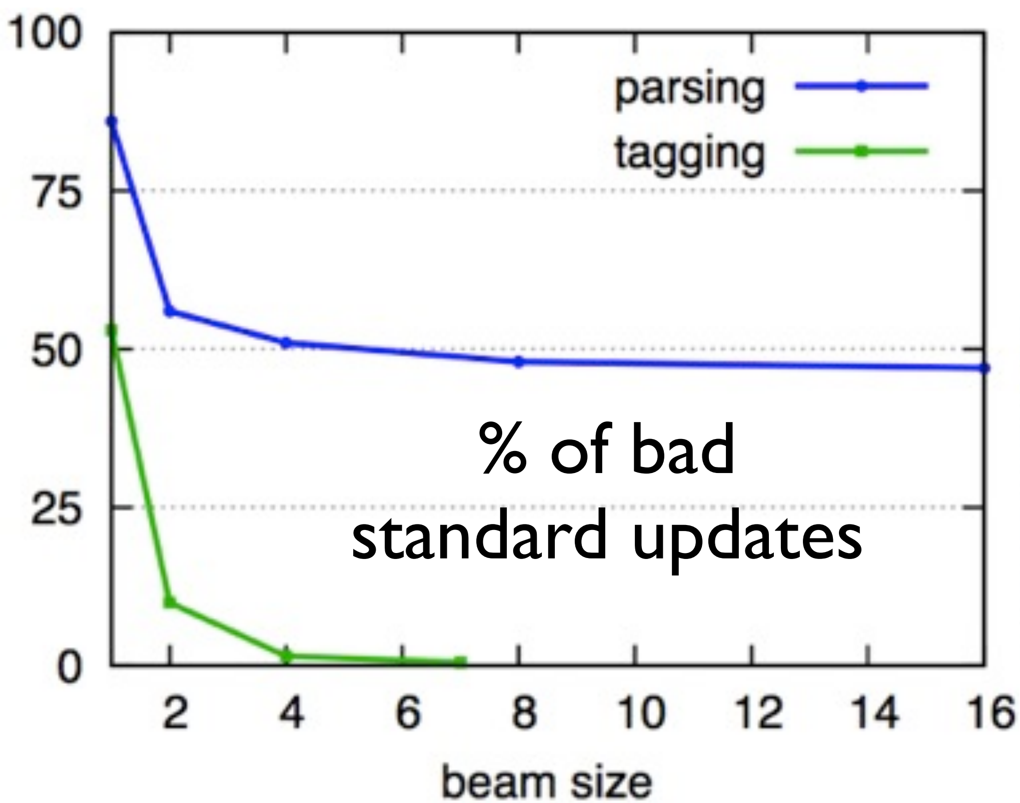
Max-violation converges much faster

- early update: 38 iterations, 15.4 hours (92.24)
- max-violation: 10 iterations, 4.6 hours (92.25)
  12 iterations, 5.5 hours (92.32)

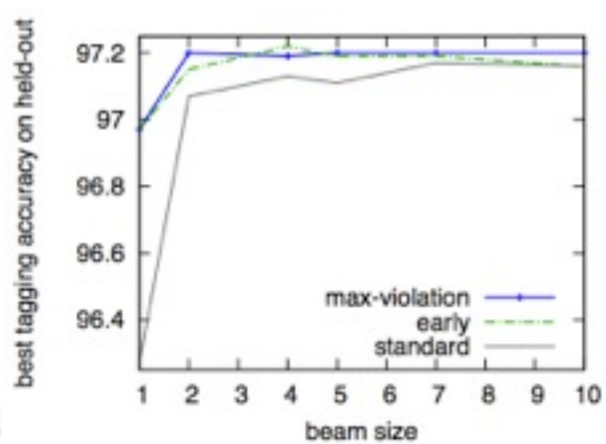


Comparison b/w tagging & parsing

- search error is much more severe in parsing than in tagging
- standard update is OK in tagging except greedy search (b=1)
- but performs horribly in parsing even at large beam (b=8)
  - because ~50% of standard updates are bad (non-violation)!



take-home message:
our methods are more helpful
for harder search problems!



	test
standard	**79.1**
early	92.1
max-violation	92.2

Annotated Bibliography

- Collins, 2002. Discriminative Training for Hidden Markov Models. In Proceedings of EMNLP. (“structured perceptron”)
- Collins and Roark, 2004. Perceptron Algorithm for Incremental Parsing. In Proceedings of ACL (“early-update”)
- Daume and Marcu, 2005. Learning as Search Optimization. In Proceedings of ICML (“LaSO”).
- Daume, 2006. PhD Thesis. (fast “averaging” trick)
- Huang, Phayong, and Guo, 2012. Structured Perceptron with Inexact Search. In Proceedings of NAACL. (“violation-fixing” framework and proofs, “max-violation”)