Machine Learning

Fall 2017

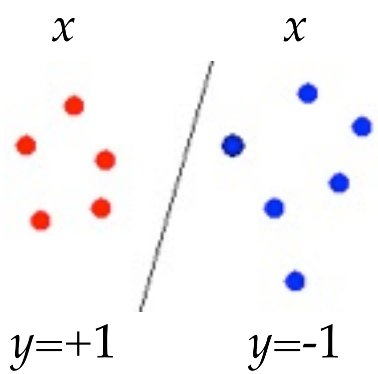
Structured Prediction

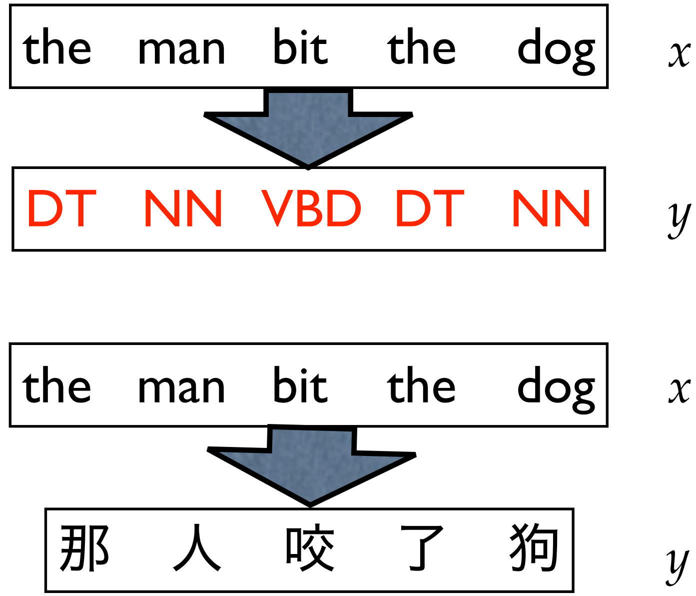
(structured perceptron, HMM, structured SVM)

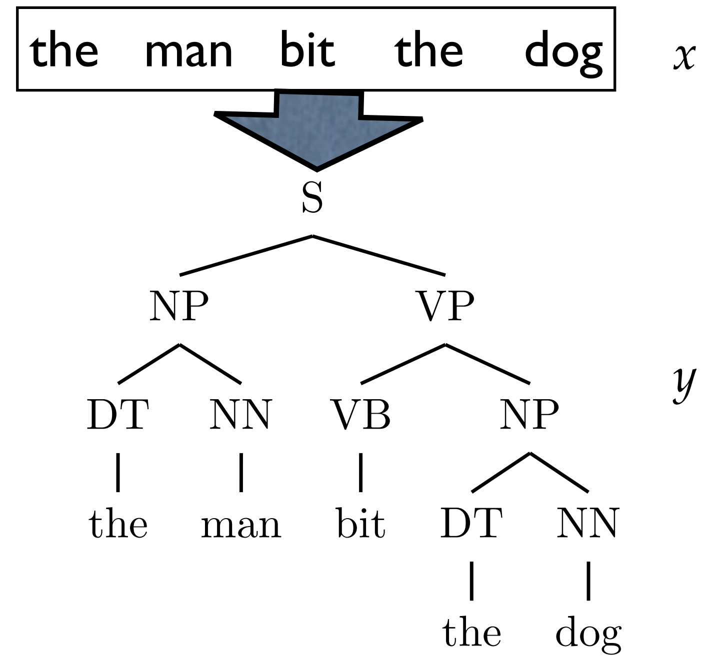
Professor Liang Huang

(Chap. 17 of CIML)

Structured Prediction



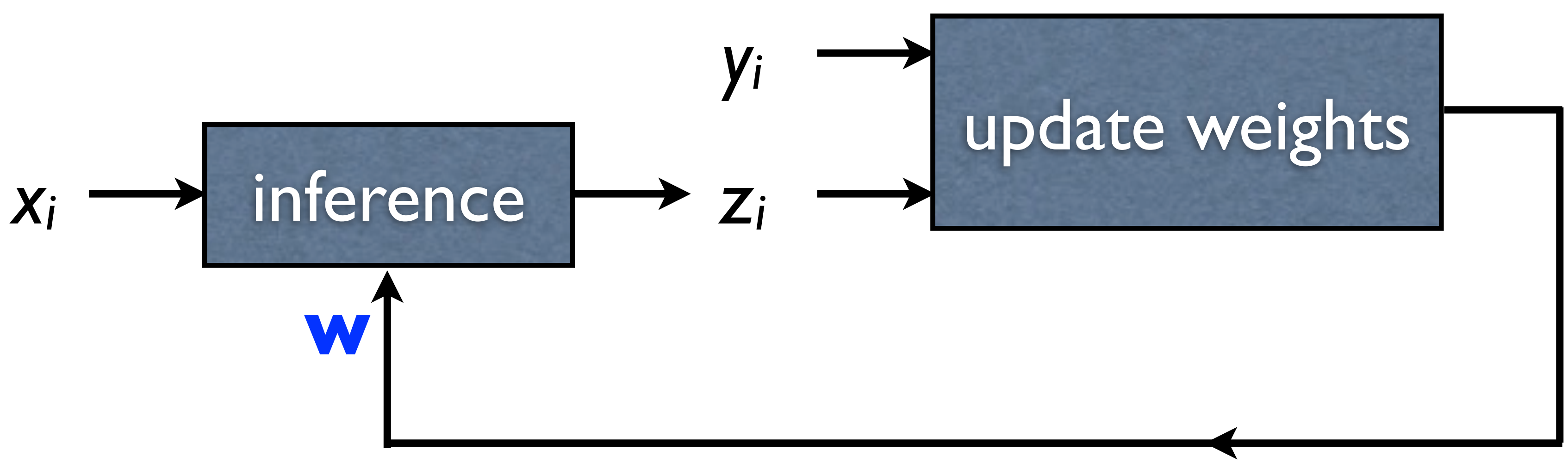




- binary classification: output is binary
- multiclass classification: output is a number (small # of classes)
- structured classification: output is a structure (seq., tree, graph)
  - part-of-speech tagging, parsing, summarization, translation
  - exponentially many classes: *search* (inference) efficiency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

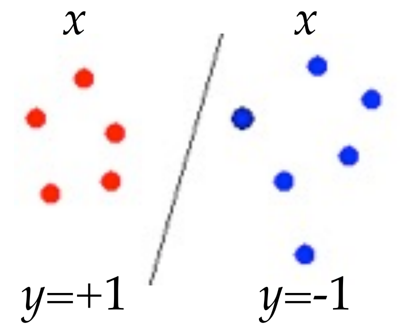
Generic Perceptron

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



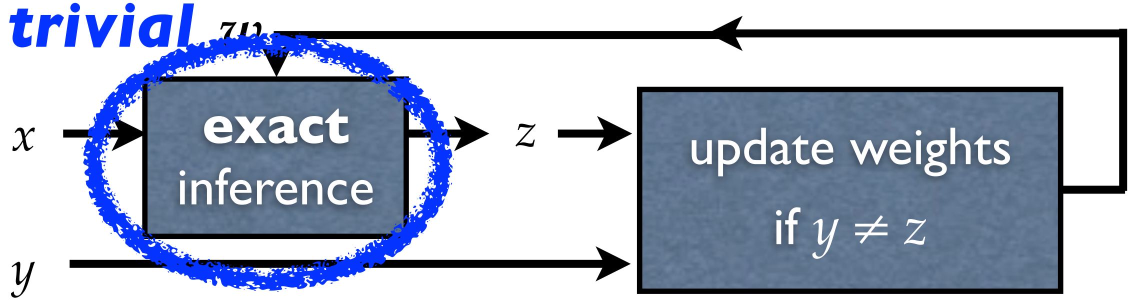
Perceptron: from binary to structured

binary classification



2 classes

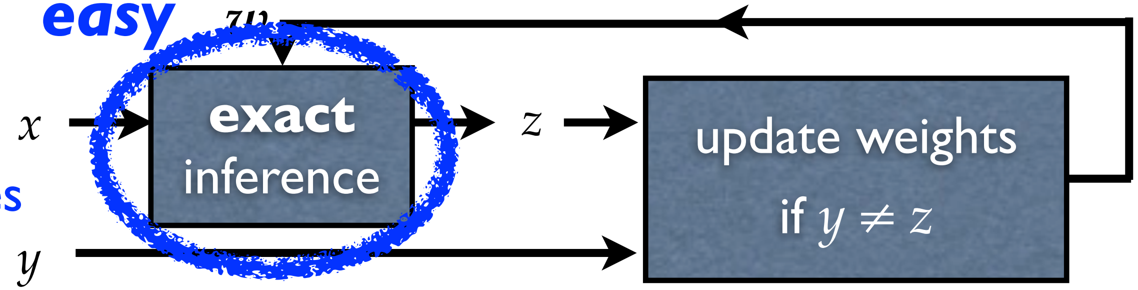
trivial



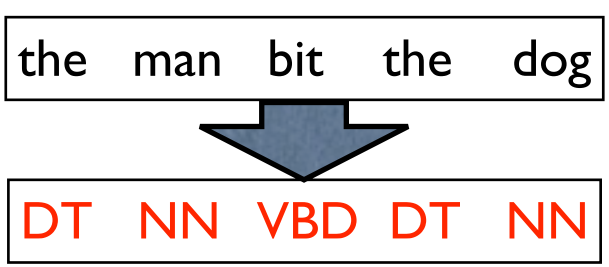
multiclass classification

constant
# of classes

easy



structured classification

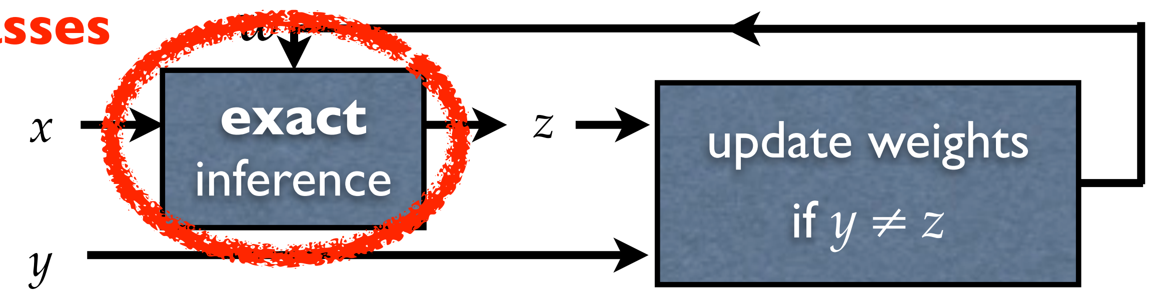


x

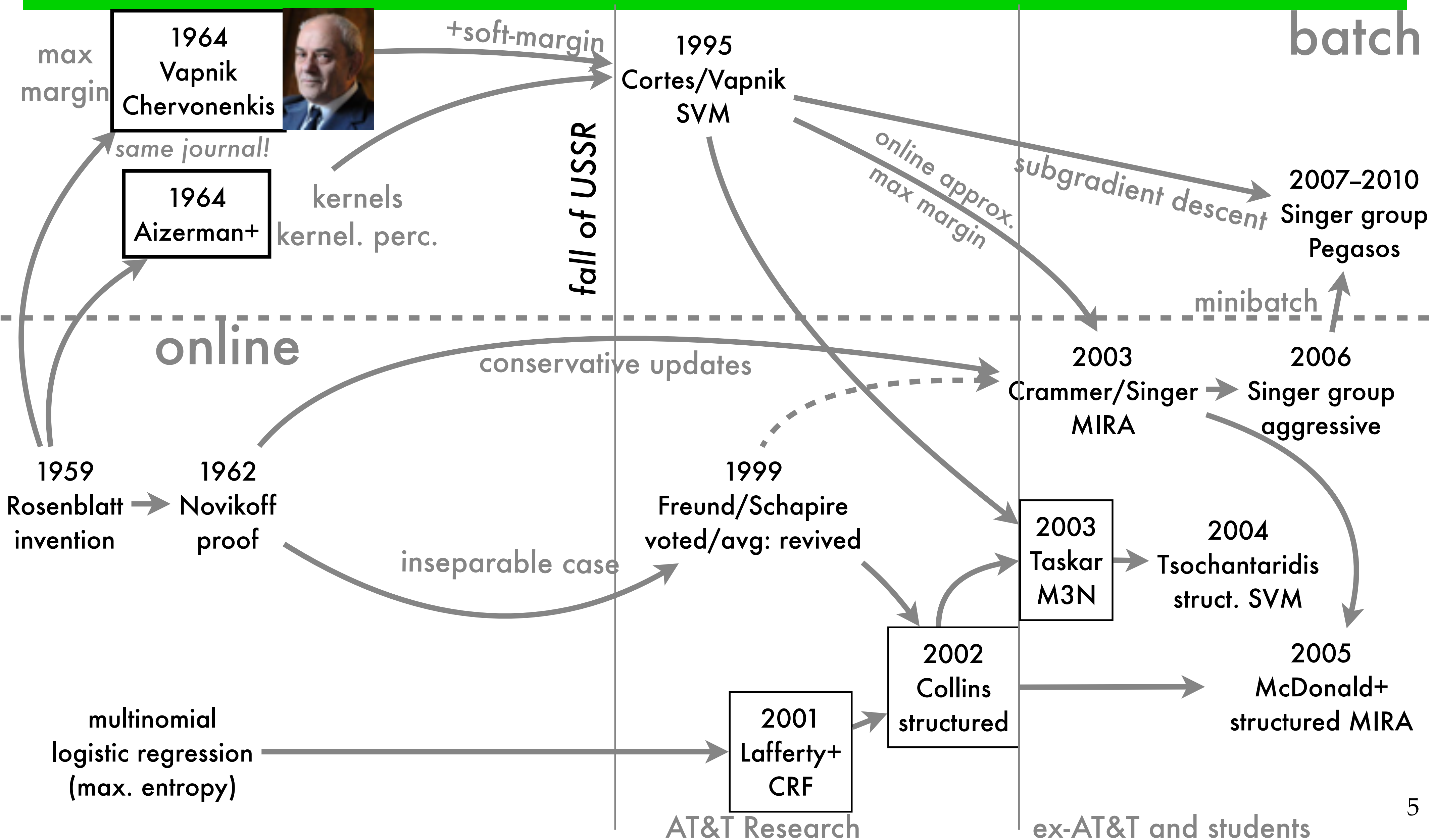
y

exponential
# of classes

hard



From Perceptron to SVM

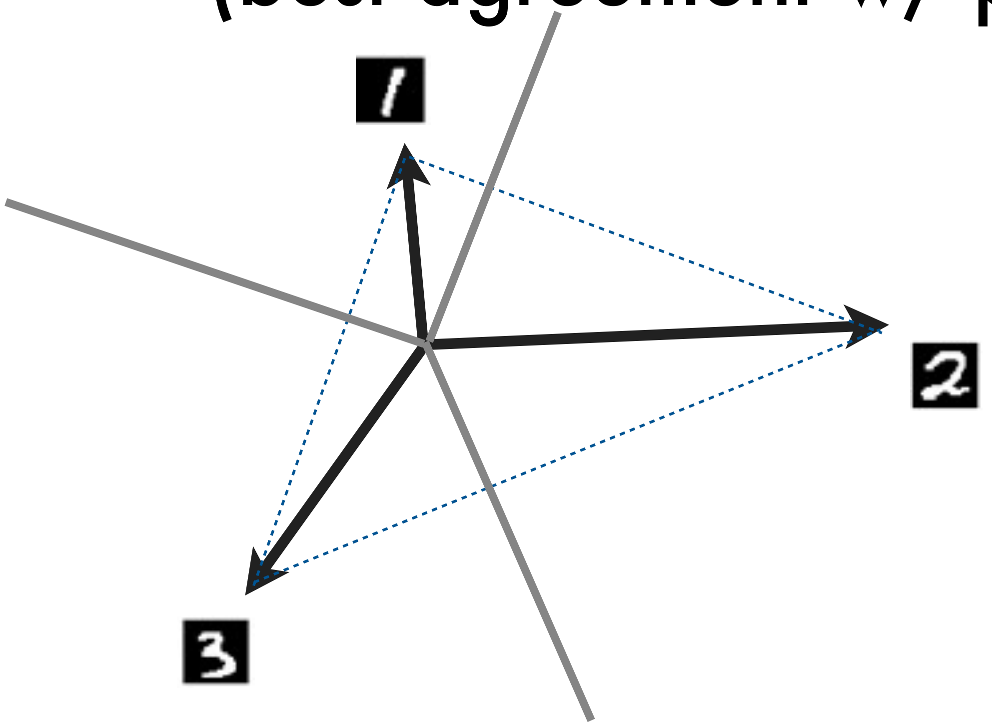


Multiclass Classification

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

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

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

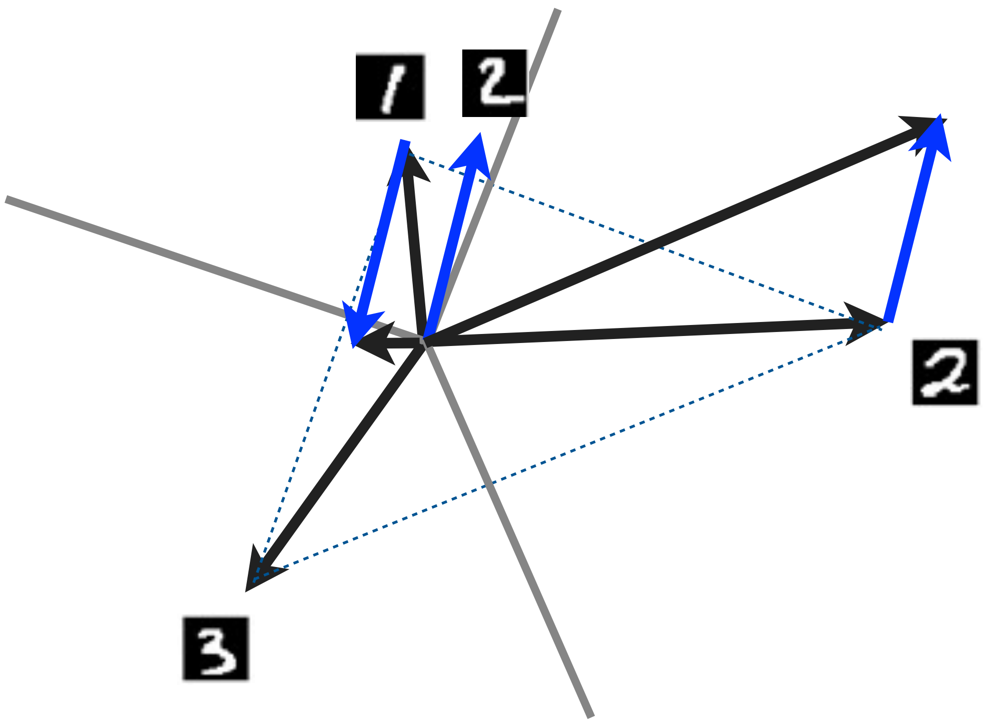


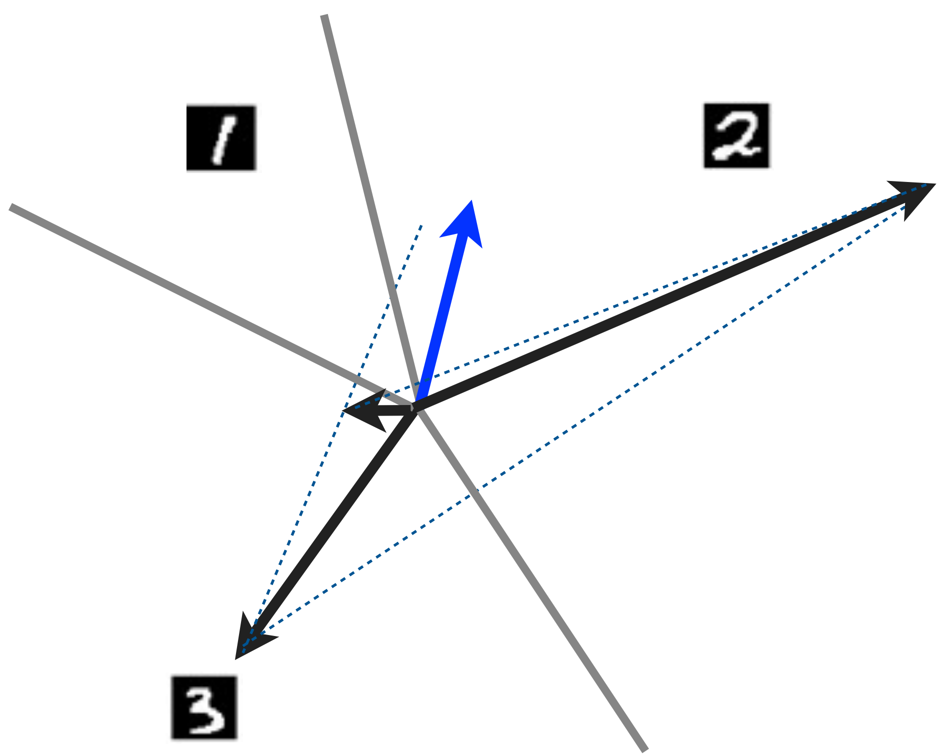
Q1: what about 2-class?

Q2: do we still need augmented space?

Multiclass Perceptron

- 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:

for all

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

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

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

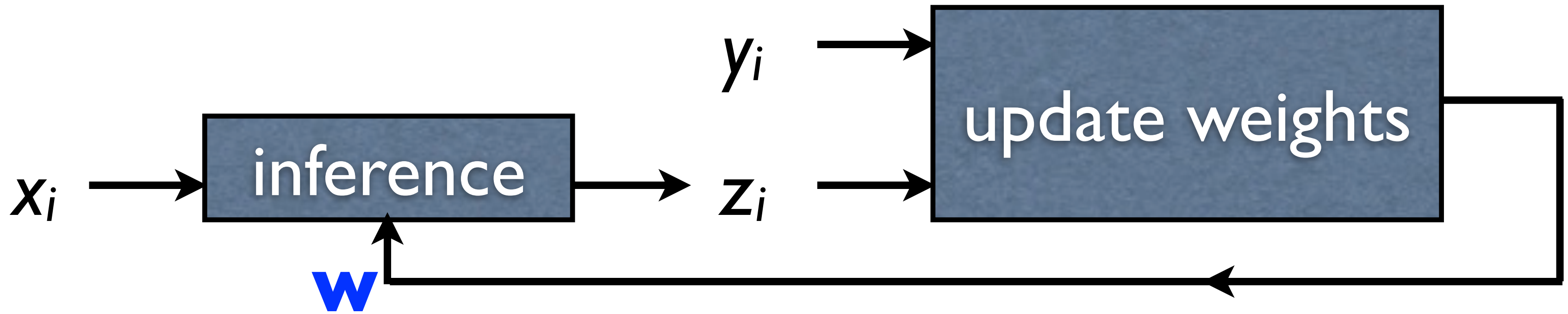
ϕ(x,y)={(<s>, DT): 1, (DT, NN):2, ..., (NN, </s>): 1, (DT, the):1, ..., (VBD, bit): 1, ..., (NN, dog): 1}

ϕ(x,z)={(<s>, DT): 1, (DT, NN):2, ..., (NN, </s>): 1, (DT, the):1, ..., (NN, bit): 1, ..., (NN, dog): 1}

- weights ++: (NN,VBD) (VBD, DT) (VBD, bit)
- weights −−: (NN, NN) (NN, DT) (NN, bit)

ϕ(x,y) - ϕ(x,z)

Structured Perceptron



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

Initialization: $\mathbf{W} = 0$

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

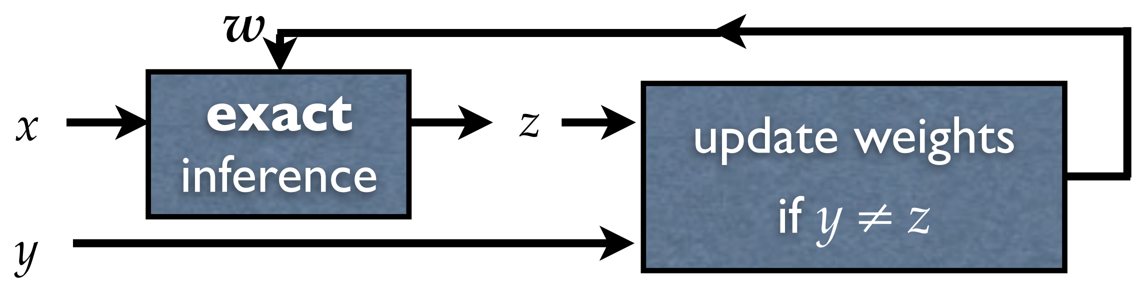
Algorithm: For $t = 1 \ldots T, i = 1 \ldots n$

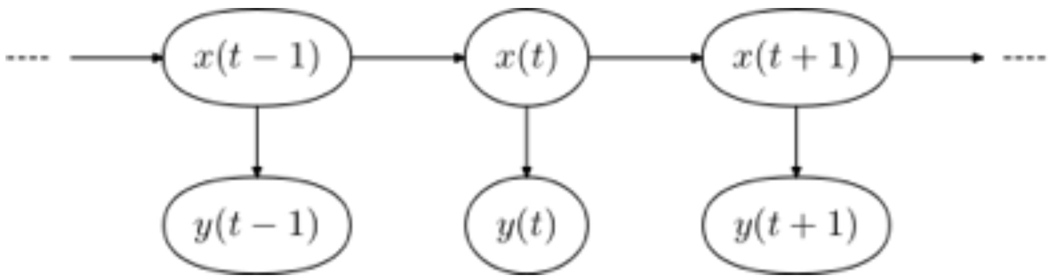
$z_i = F(x_i)$

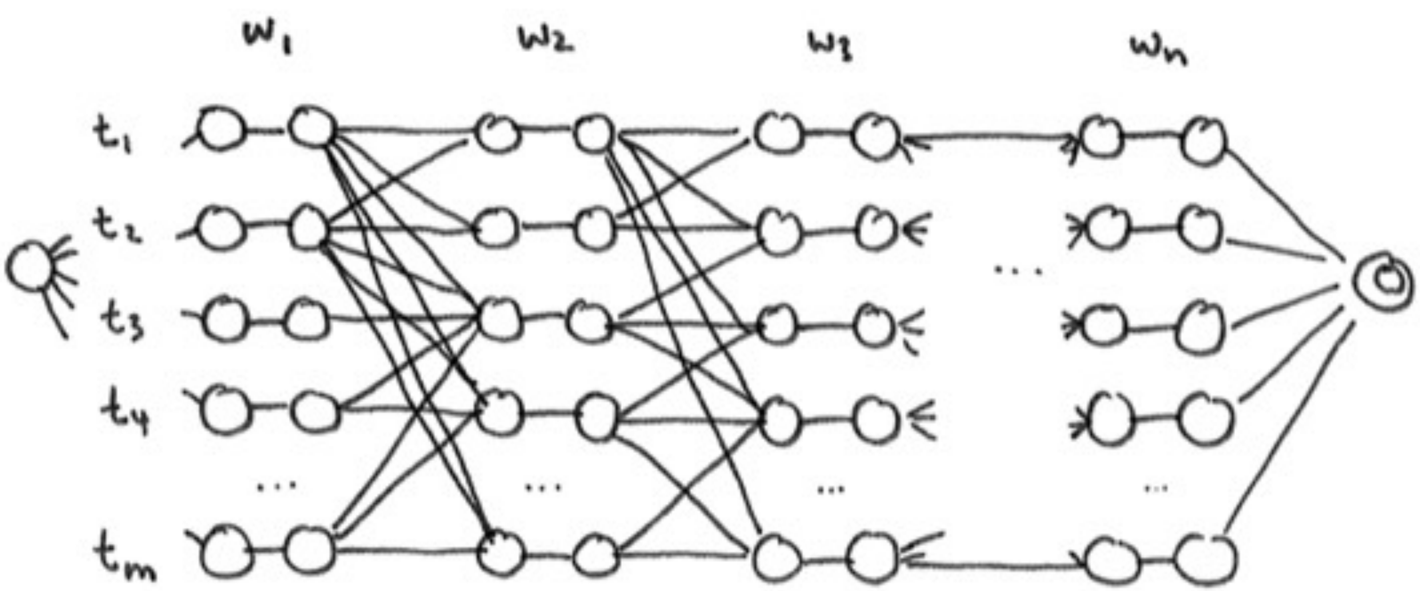
If $(z_i \neq y_i)$ $\boxed{\mathbf{W} \leftarrow \mathbf{W} + \mathbf{\Phi}(x_i, y_i) - \mathbf{\Phi}(x_i, z_i)}$

Output: Parameters $\mathbf{W}$

Inference: Dynamic Programming





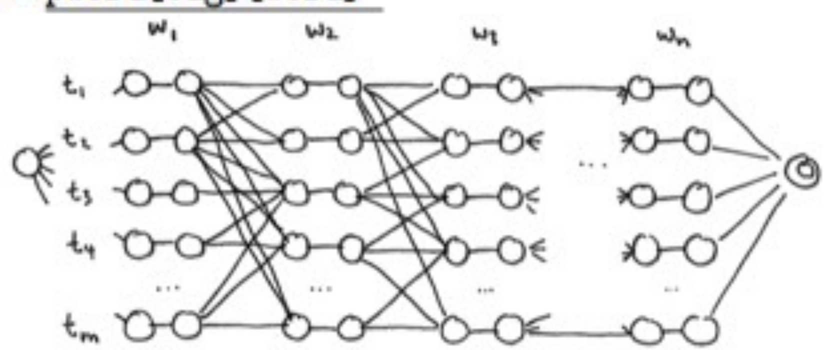


Python implementation

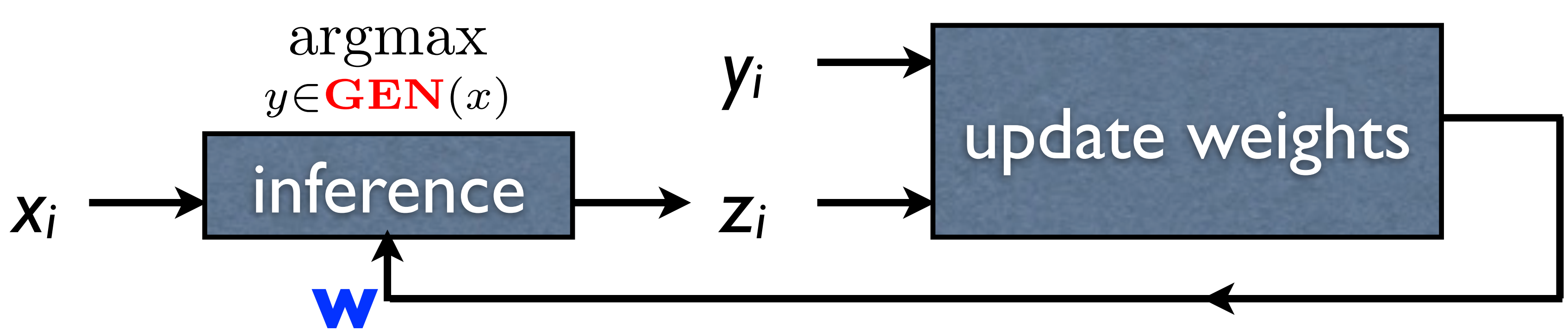
Complete this Python code implementing the Viterbi algorithm for part-of-speech tagging. It should print a list of word/tag pairs, e.g. [('a', 'D'), ('can', 'N'), ('can', 'A'), ('can', 'V'), ('a', 'D'), ('can', 'N')].

Q: what about top-down recursive + memoization?

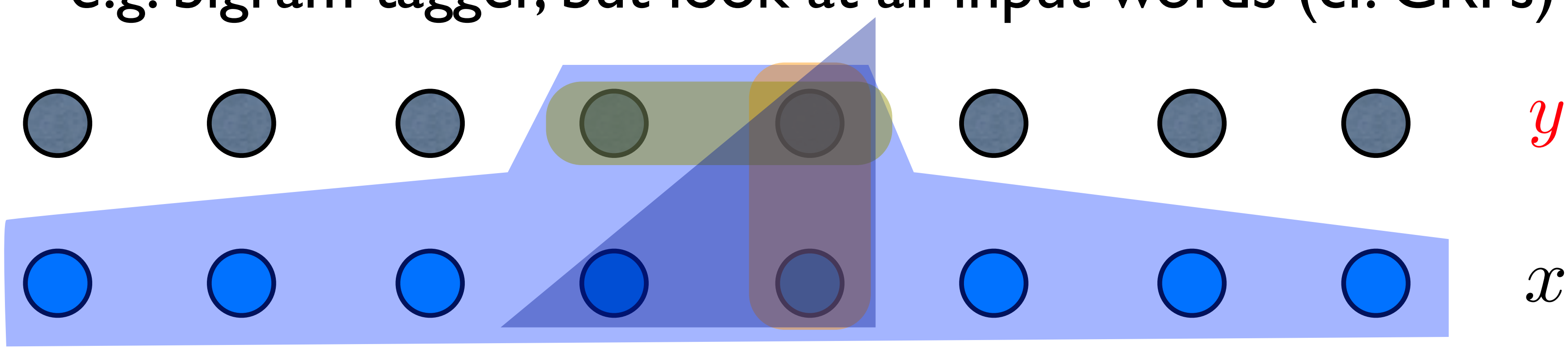
```
from collections import defaultdict

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]
```



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

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)

sparse vector: defaultdict

Algorithm 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 $\boldsymbol{w}_0 - \boldsymbol{w}_a / c$

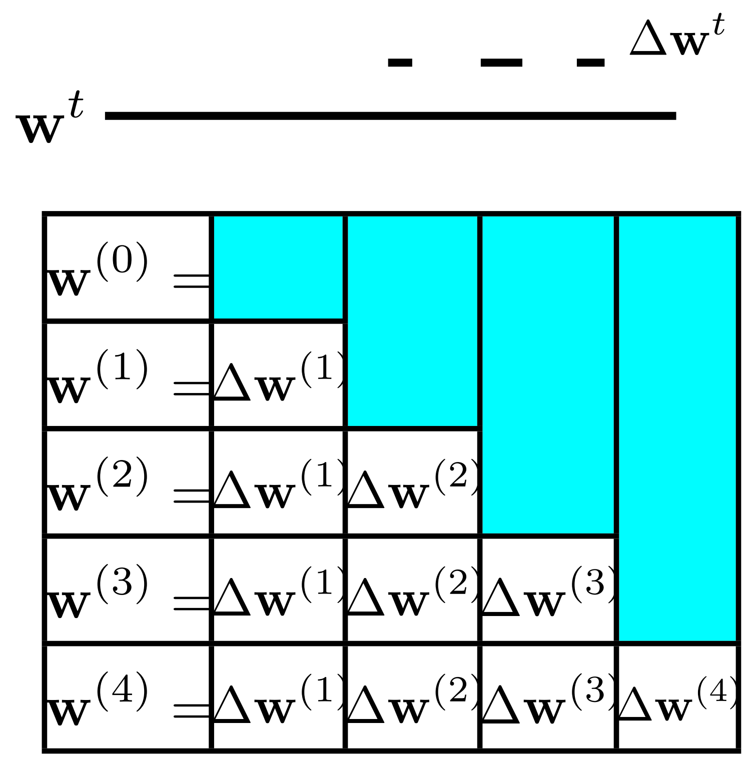
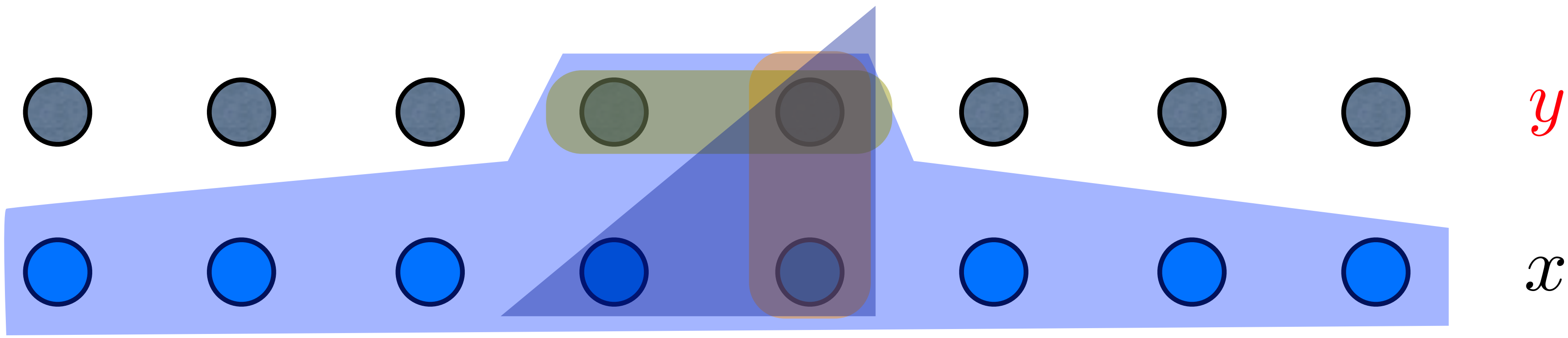


Figure 2.3: The averaged structured perceptron learning algorithm.

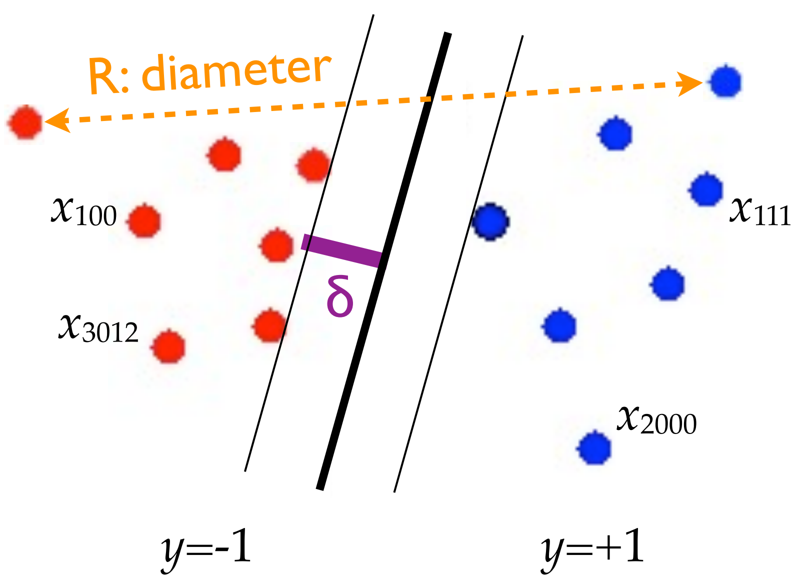
Do we need smoothing?

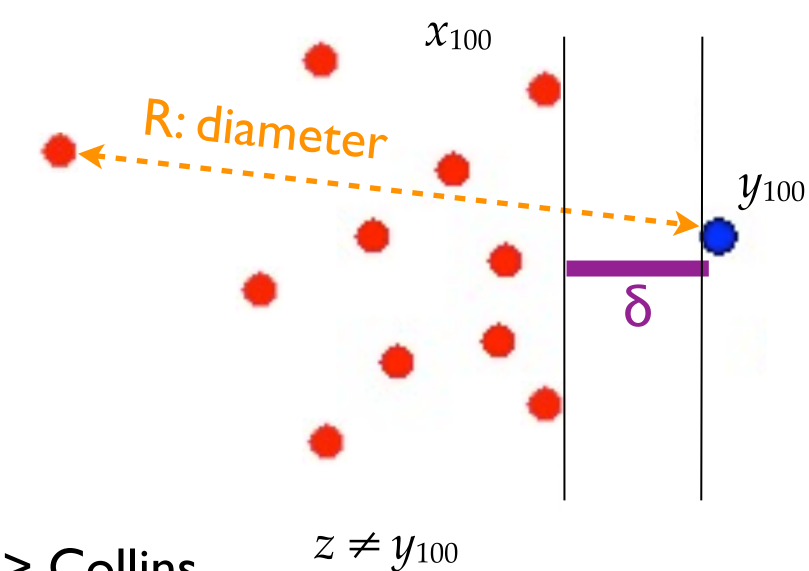


- smoothing is much easier in discriminative models
- just make sure for each feature template, its subset templates are also included
  - e.g., to include (t_0 w_0 w_{-1}) you must also include
  - (t_0 w_0) (t_0 w_{-1}) (w_0 w_{-1})
  - and maybe also (t_0 t_{-1}) because t is less sparse than w

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 ≤ R² / δ²**   R: diameter

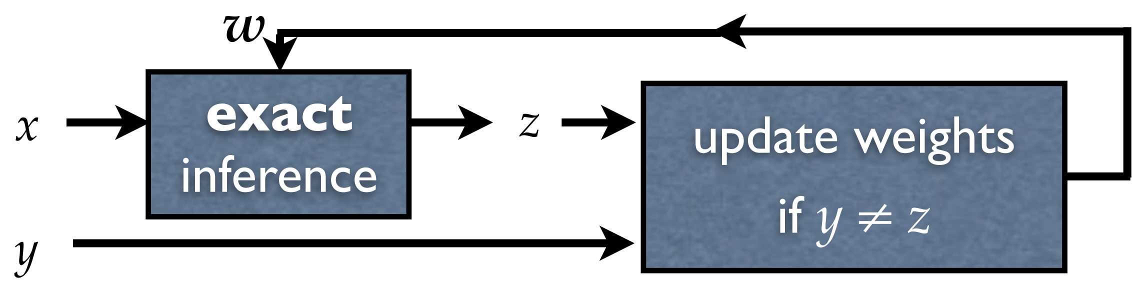


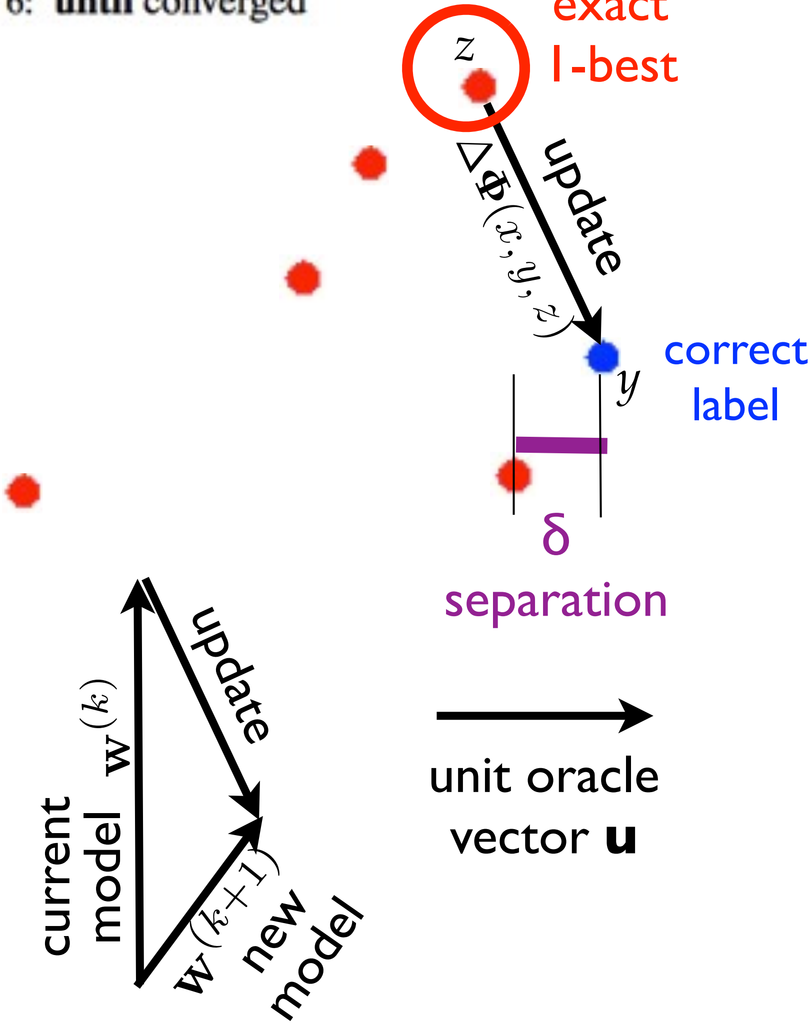


Rosenblatt => Collins
1957 2002

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:

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

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

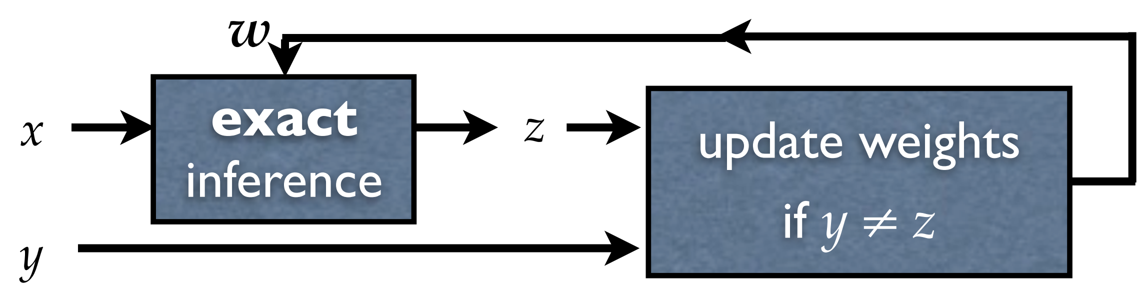
$$\mathbf{u}\cdot\mathbf{w}^{(k+1)} \geq k\delta \qquad \text{(by induction)}$$

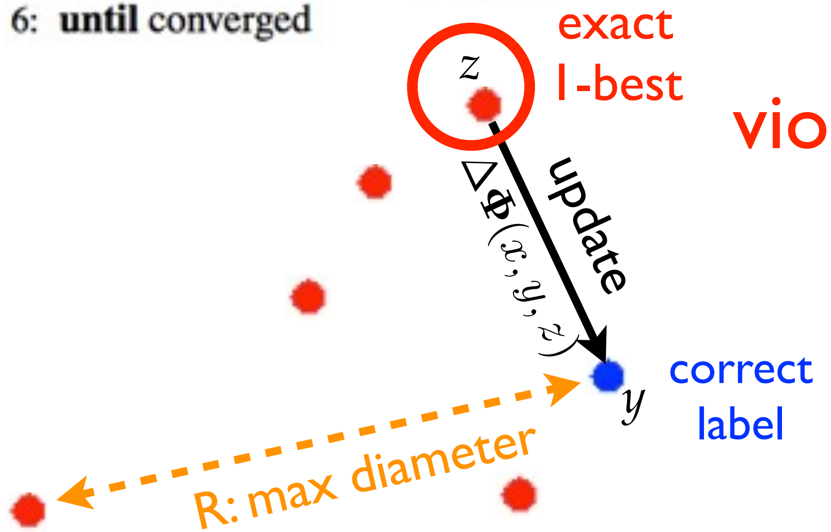
$$\|\mathbf{u}\|\|\mathbf{w}^{(k+1)}\| \geq \mathbf{u}\cdot\mathbf{w}^{(k+1)} \geq k\delta$$

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

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
```





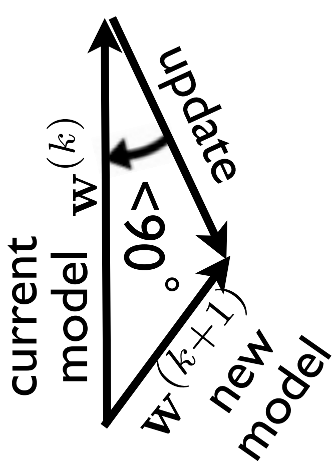
violation: incorrect label scored higher

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 SVM

- structured perceptron: $w \cdot \Delta\phi(x,y,z) > 0$
- SVM: for all (x,y), functional margin $y(w \cdot x) \geq 1$
- structured SVM version 1: simple loss
  - for all (x,y), for all $z \neq y$, margin $w \cdot \Delta\phi(x,y,z) \geq 1$
  - correct y has to score higher than any wrong z by 1
- structured SVM version 2: structured loss
  - for all (x,y), for all $z \neq y$, margin $w \cdot \Delta\phi(x,y,z) \geq \ell(y,z)$
  - correct y has to score higher than any wrong z by $\ell(y,z)$, a distance metric such as hamming loss

Loss-Augmented Decoding

- want for all z: $\quad w \cdot \phi(x,y) \geq \quad w \cdot \phi(x,z) + \ell(y,z)$
- same as: $\quad w \cdot \phi(x,y) \geq \max_z w \cdot \phi(x,z) + \ell(y,z)$
- loss-augmented decoding: $\operatorname{argmax}_z w \cdot \phi(x,z) + \ell(y,z)$
- if $\ell(y,z)$ factors in z (e.g. hamming), just modify DP

Algorithm 41 StochSubGradStructSVM($\mathbf{D}$, *MaxIter*, λ, ℓ) — CIML version

$w \leftarrow \mathbf{0}$ // initialize weights
for $iter = 1 \dots MaxIter$ **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $\hat{y} \leftarrow \operatorname{argmax}_{\hat{y} \in \mathcal{Y}(x)} w \cdot \phi(x,\hat{y}) + \ell(y,\hat{y})$ // loss-augmented prediction ←modified DP
    if $\hat{y} \neq y$ **then**
      $w \leftarrow w + \phi(x,y) - \phi(x,\hat{y})$ // update weights ←*should have learning rate!*
    end if
    $w \leftarrow w - \frac{\lambda}{N} w$ // shrink weights due to regularizer $\quad \lambda = 1/(2C)$
  end for
end for
return w // return learned weights

very similar to Pegasos; but should use Pegasos framework instead

Correct Version following Pegasos

- want for all z: $w \cdot \phi(x,y) \geq w \cdot \phi(x,z) + \ell(y,z)$
- same as: $w \cdot \phi(x,y) \geq \max_z w \cdot \phi(x,z) + \ell(y,z)$
- loss-augmented decoding: $\operatorname{argmax}_z w \cdot \phi(x,z) + \ell(y,z)$
- if $\ell(y,z)$ factors in z (e.g. hamming), just modify DP

Algorithm 41 StochSubGradStructSVM($\mathbf{D}$, *MaxIter*, λ, ℓ)

$w \leftarrow \mathbf{0}$ // initialize weights
for *iter* = 1 … *MaxIter* **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $w \leftarrow w - 1/t\ w$ // shrink weights due to regularizer
    $\hat{y} \leftarrow \operatorname{argmax}_{\hat{y} \in \mathcal{Y}(x)} w \cdot \phi(x,\hat{y}) + \ell(y,\hat{y})$ // loss-augmented prediction
    if $\hat{y} \neq y$ **then**
      $w \leftarrow w + NC/2t\ (\phi(x,y) - \phi(x,\hat{y}))$ // update weights
    end if
  end for
end for
return w // return learned weights

N=|D|, C is from SVM
t +=1 for each example

Struct. Perceptron vs Struct. SVM

- tagging, ATIS (train: 488 sent); SVM < avg perc << perc

```
perceptron
epoch 1 updates 102, |W|=291, train_err 3.90%, dev_err 9.36% avg_err 6.14%
epoch 2 updates 91, |W|=334, train_err 3.33%, dev_err 8.19% avg_err 4.97%
epoch 3 updates 78, |W|=347, train_err 2.92%, dev_err 5.85% avg_err 4.97%
epoch 4 updates 81, |W|=368, train_err 3.11%, dev_err 6.73% avg_err 5.85%
epoch 5 updates 78, |W|=378, train_err 2.70%, dev_err 6.14% avg_err 5.56%
epoch 6 updates 63, |W|=385, train_err 2.26%, dev_err 6.14% avg_err 5.56%
epoch 7 updates 69, |W|=385, train_err 2.43%, dev_err 7.02% avg_err 5.56%
epoch 8 updates 60, |W|=388, train_err 2.15%, dev_err 6.73% avg_err 5.56%
epoch 9 updates 59, |W|=390, train_err 2.04%, dev_err 6.14% avg_err 5.56%
epoch 10 updates 64, |W|=394, train_err 2.15%, dev_err 5.85% avg_err 5.26%

SVM C=1
epoch 1 updates 116, |W|=311, train_err 4.55%, dev_err 5.85%
epoch 2 updates 82, |W|=328, train_err 3.05%, dev_err 4.97%
epoch 3 updates 78, |W|=334, train_err 2.92%, dev_err 5.56%
epoch 4 updates 77, |W|=339, train_err 2.92%, dev_err 5.26%
epoch 5 updates 80, |W|=344, train_err 2.94%, dev_err 5.56%
epoch 6 updates 73, |W|=345, train_err 2.75%, dev_err 4.68%
epoch 7 updates 72, |W|=347, train_err 2.75%, dev_err 4.97%
epoch 8 updates 75, |W|=352, train_err 2.86%, dev_err 4.97%
epoch 9 updates 74, |W|=353, train_err 2.78%, dev_err 4.97%
epoch 10 updates 72, |W|=354, train_err 2.78%, dev_err 4.97%
```