CRF and Structured Perceptron

CS 585, Fall 2015 -- Oct. 6
Introduction to Natural Language Processing
http://people.cs.umass.edu/~brenocon/inlp2015/

Brendan O'Connor

UMASS CS
SCHOOL OF COMPUTER SCIENCE
50 YEARS

- Viterbi exercise solution
- CRF & Structured Perceptrons

- Thursday: project discussion + midterm review

Log-linear models (NB, LogReg, HMM, CRF...)

- x: Text Data
- y: Proposed class or sequence
- θ: Feature weights (model parameters)
- f(x,y): Feature extractor, produces feature vector

$$p(y|x) = \frac{1}{Z} \exp \underbrace{\left(\theta^{\mathsf{T}} f(x,y)\right)}_{G(y)}$$

Decision rule: $\arg \max_{y^* \in outputs(x)} G(y^*)$

How to we evaluate for HMM/CRF? Viterbi!

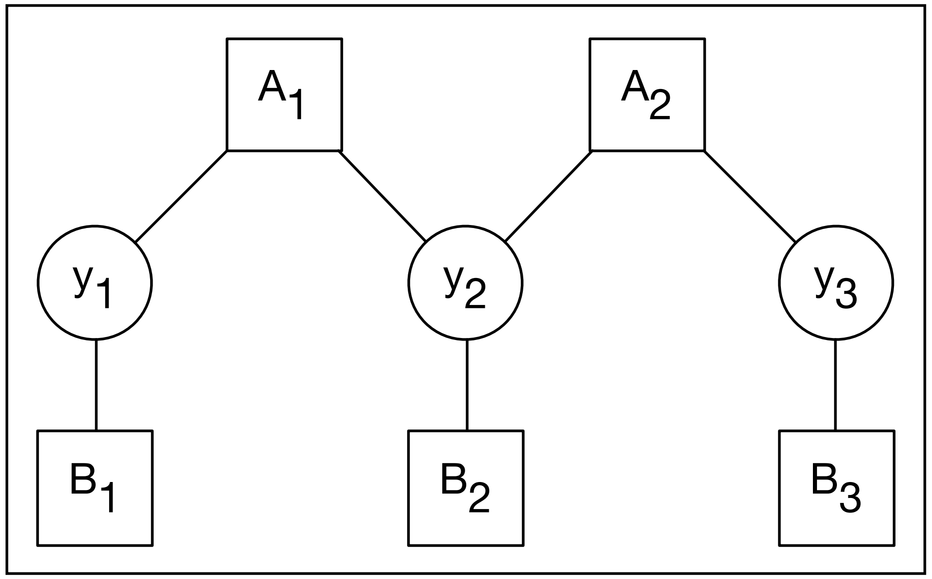
Things to do with a log-linear model

$$p(y|x) = \frac{1}{Z}\exp\Big(\underbrace{\theta^{\mathsf{T}} f(x,y)}_{G(y)}\Big)$$

	f(x,y) Feature extractor (feature vector)	**x** Text Input	**y** Output	**θ** Feature weights
decoding/prediction $\arg\max_{y^* \in outputs(x)} G(y^*)$	given	given (just one)	obtain (just one)	given
parameter learning	given	given (many pairs)	given (many pairs)	obtain
feature engineering (human-in-the-loop)	fiddle with during experiments	given (many pairs)	given (many pairs)	obtain in each experiment

[This is new slide after lecture]

HMM as factor graph



$$p(y, w) = \prod_t p(w_y|y_t)\ p(y_{t+1}|y_t)$$

$$\underset{\substack{\uparrow \\ G(y)}}{\log p(y, w)} = \sum_t \underset{\substack{\uparrow \\ B_t(y_t)}}{\log p(w_t|y_t)} + \underset{\substack{\uparrow \\ A(y_t, y_{t+1})}}{\log p(y_t|y_{t-1})}$$

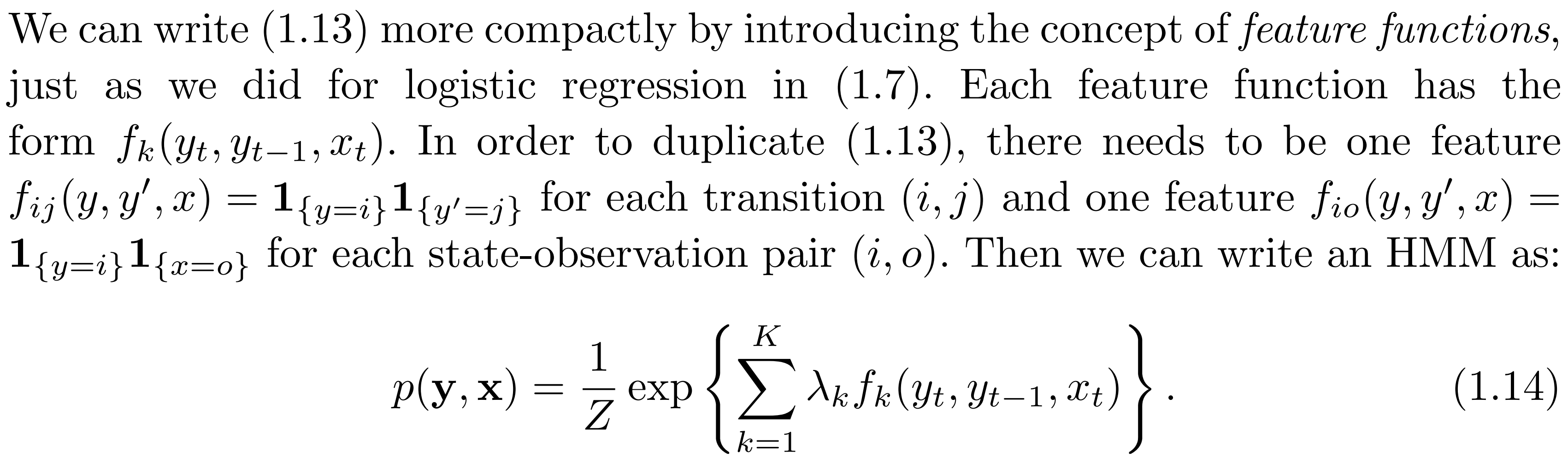
goodness

emission factor score

transition factor score

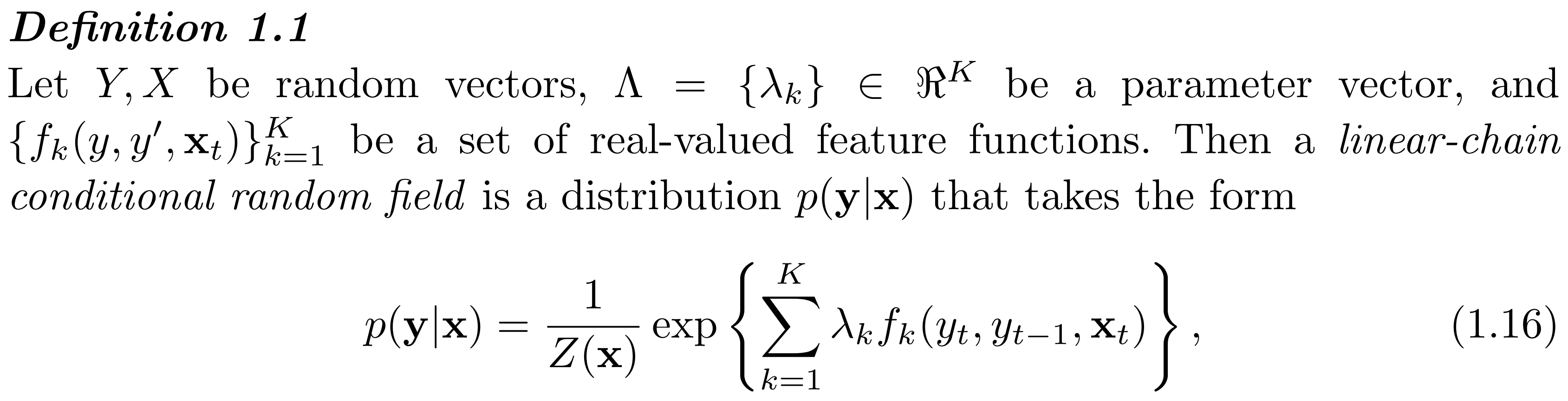
(Additive) Viterbi: $\arg \max_{y^* \in outputs(x)} G(y^*)$

- is there a terrible bug in sutton&mccallum? there's no sum over *t* in these equations!



We can write (1.13) more compactly by introducing the concept of *feature functions*, just as we did for logistic regression in (1.7). Each feature function has the form $f_k(y_t, y_{t-1}, x_t)$. In order to duplicate (1.13), there needs to be one feature $f_{ij}(y, y', x) = \mathbf{1}_{\{y=i\}}\mathbf{1}_{\{y'=j\}}$ for each transition (i, j) and one feature $f_{io}(y, y', x) = \mathbf{1}_{\{y=i\}}\mathbf{1}_{\{x=o\}}$ for each state-observation pair (i, o). Then we can write an HMM as:

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t)\right\}. \tag{1.14}$$

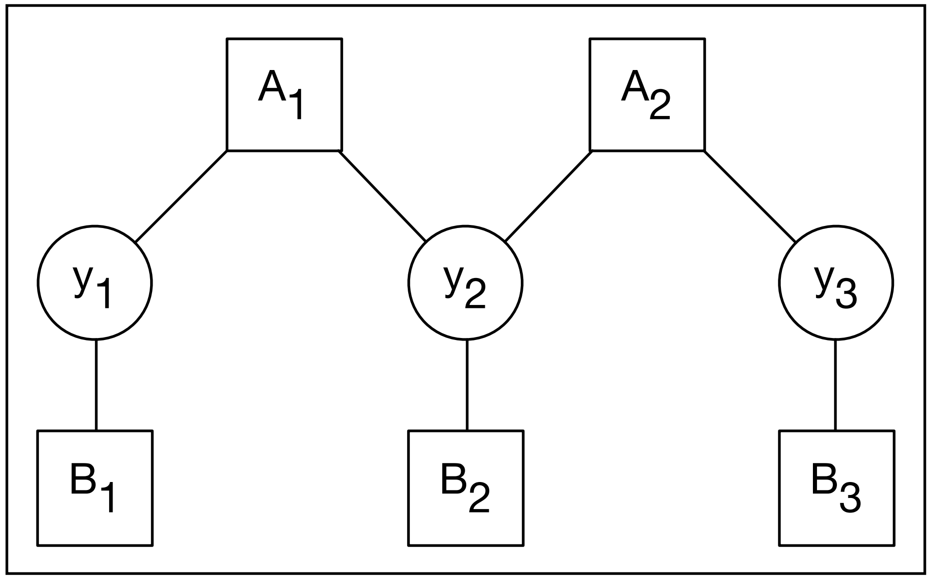


Definition 1.1

Let Y, X be random vectors, $\Lambda = \{\lambda_k\} \in \Re^K$ be a parameter vector, and $\{f_k(y, y', \mathbf{x}_t)\}_{k=1}^{K}$ be a set of real-valued feature functions. Then a *linear-chain conditional random field* is a distribution $p(\mathbf{y}|\mathbf{x})$ that takes the form

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\right\}, \tag{1.16}$$

HMM as log-linear



$$p(y, w) = \prod_t p(w_y | y_t)\ p(y_{t+1} | y_t)$$

$$\log p(y, w) = \sum_t \log p(w_t | y_t) + \log p(y_t | y_{t-1})$$

↑ $G(y)$ *goodness*

↑ $B_t(y_t)$ ***emission factor score***

↑ $A(y_t, y_{t+1})$ ***transition factor score***

$$\text{G(y)} = \sum_t \left[\sum_{k \in K} \sum_{w \in V} \mu_{w,k} 1\{y_t = k \wedge w_t = w\} + \sum_{k,j \in K} \lambda_{j,k} 1\{y_t = j \wedge y_{t+1} = k\} \right]$$

$$= \sum_t \sum_{i \in \text{allfeats}} \theta_i f_{t,i}(y_t, y_{t+1}, w_t)$$

$$= \sum_{i \in \text{allfeats}} \theta_i f_i(y_t, y_{t+1}, w_t)$$

[~ SM eq 1.13, 1.14]

CRF

$$\log p(y|x) = C + \theta^{\mathsf{T}} f(x, y)$$

Prob. dist over *whole* sequence

$$f(x, y) = \sum_t f_t(x, y_t, y_{t+1})$$

Linear-chain CRF: whole-sequence feature function decomposes into pairs

- advantages
  - 1. why just word identity features? add many more!
  - 2. can train it to optimize accuracy of sequences (discriminative learning)
- Viterbi can be used for efficient prediction

	finna	get	good
gold *y* =	V	V	A

Two simple feature templates

"Transition features"

$$f_{trans:A,B}(x,y) = \sum_t 1\{y_{t-1} = A, y_t = B\}$$

"Observation features"

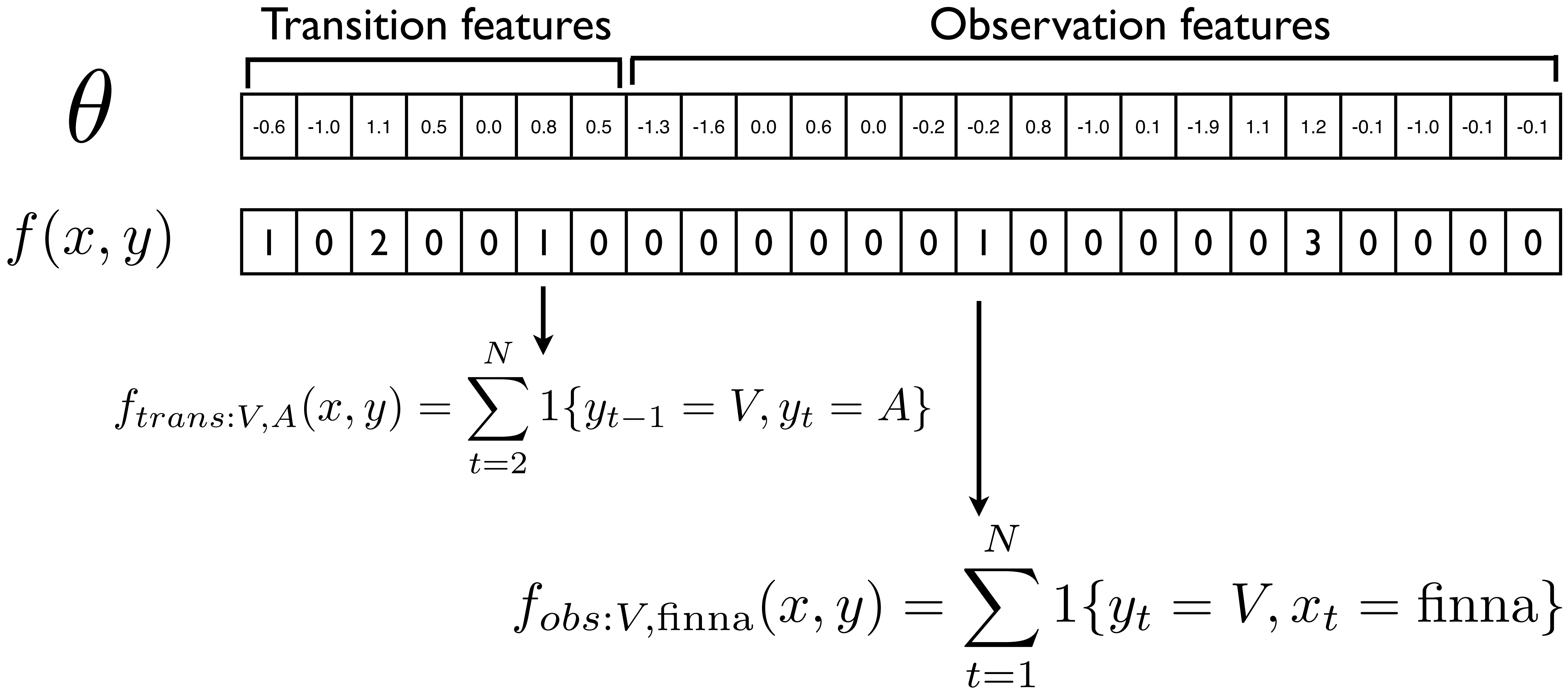
$$f_{emit:A,w}(x,y) = \sum_t 1\{y_t = A, x_t = w\}$$

f(x,y) is...

V,V: 1
V,A: 1
V,N: 0
....
V,finna: 1
V,get: 1
A,good: 1
N,good: 0
...

	finna	get	good
gold y =	V	V	A

Mathematical convention is numeric indexing, though sometimes convenient to implement as hash table.



	Transition features							Observation features																
θ	-0.6	-1.0	1.1	0.5	0.0	0.8	0.5	-1.3	-1.6	0.0	0.6	0.0	-0.2	-0.2	0.8	-1.0	0.1	-1.9	1.1	1.2	-0.1	-1.0	-0.1	-0.1
$f(x, y)$	1	0	2	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	3	0	0	0	0

$$f_{trans:V,A}(x,y) = \sum_{t=2}^{N} 1\{y_{t-1} = V, y_t = A\}$$

$$f_{obs:V,\mathrm{finna}}(x,y) = \sum_{t=1}^{N} 1\{y_t = V, x_t = \mathrm{finna}\}$$

Goodness(y) = $\theta^{\mathsf{T}} f(x, y)$

CRF: prediction with Viterbi

$$\log p(y|x) = C + \theta^{\mathsf{T}} f(x,y)$$

Prob. dist over *whole* sequence

$$f(x,y) = \sum_t f_t(x, y_t, y_{t+1})$$

Linear-chain CRF: whole-sequence feature function decomposes into pairs

- Scoring function has *local decomposition*

$$f(x,y) = \sum_t^T f^{(B)}(t,x,y) + \sum_{t=2}^T f^{(A)}(y_{t-1}, y_t)$$

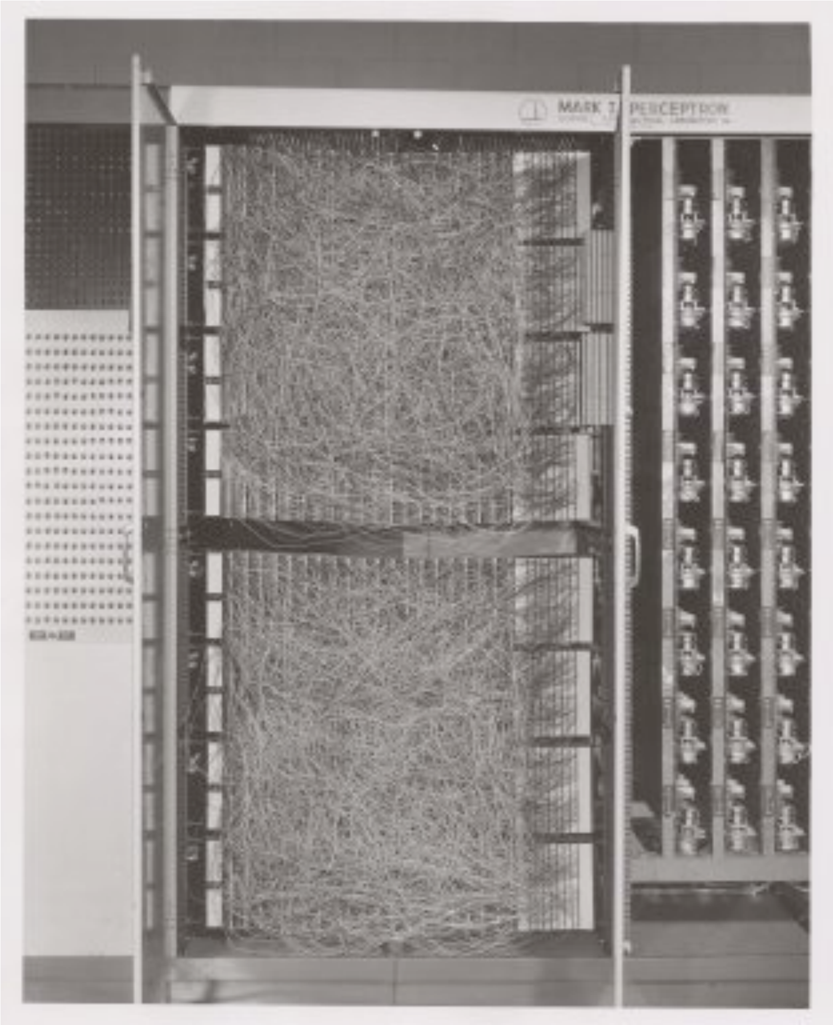
$$\theta^{\mathsf{T}} f(x,y) = \sum_t \theta^{\mathsf{T}} f^{(B)}(t,x,y) + \sum_{t=2}^T + f^{(A)}(y_{t-1}, y_t)$$

1. Motivation: we want features in our sequence model!
2. And how do we learn the parameters?

3. Outline
    1. Log-linear models
    2. Log-linear Sequence Models:
        1. Log-scale additive Viterbi
        2. Conditional Random Fields
    3. **Learning: the Perceptron**

The Perceptron Algorithm

- Perceptron is not a model:
  it is a learning algorithm
  - Rosenblatt 1957
- Insanely simple algorithm
  - Iterate through dataset.
    Predict.
    Update weights to fix prediction errors.
- Can be used for classification OR structured prediction
  - *structured perceptron*
- Discriminative learning algorithm for *any* log-linear model (our view in this course)



The Mark I Perceptron machine was the first implementation of the perceptron algorithm. The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image. The main visible feature is a patchboard that allowed experimentation with different combinations of input features. To the right of that are arrays ofpotentiometers that implemented the adaptive weights.

Binary perceptron

- For ~10 iterations
  - For each (x,y) in dataset
    - PREDICT

$$y^* = POS \text{ if } \theta^\mathsf{T} x \geq 0$$
$$= NEG \text{ if } \theta^\mathsf{T} x < 0$$

    - IF y=y*, do nothing
    - ELSE update weights

$$\theta := \theta + r\ x$$

if POS misclassified as NEG:
let's make it more positive-y next time around

$$\theta := \theta - r\ x$$

if NEG misclassified as POS:
let's make it more negative-y next time

learning rate constant
e.g. r=1

Structured/multiclass Perceptron

- For ~10 iterations
  - For each (x,y) in dataset
    - PREDICT

$$y^* = \arg\max_{y'} \theta^\mathsf{T} f(x, y')$$

    - IF y=y*, do nothing
    - ELSE update weights

$$\theta := \theta + r[f(x,y) - f(x,y^*)]$$

learning rate constant
e.g. r=1

Features for
TRUE label

Features for
PREDICTED label

Update rule

y=POS
x="this awesome movie ..."
Make mistake: y*=NEG

learning rate
e.g. r=1

Features for
TRUE label

Features for
PREDICTED label

$$\theta := \theta + r[f(x,y) - f(x,y^*)]$$

	POS_awesome	POS_this	POS_oof	....	NEG_awesome	NEG_this	NEG_oof	....
real f(x, POS) =	1	1	0	....	0	0	0	....
pred f(x, NEG) =	0	0	0	....	1	1	0	....
f(x, POS) – f(x, NEG) =	+1	+1	0	....	-1	-1	0	....

Update rule

learning rate
e.g. r=1

Features for
TRUE label

Features for
PREDICTED label

$$\theta := \theta + r[f(x, y) - f(x, y^*)]$$

For each feature j in true y but not predicted y*:

$$\theta_j := \theta_j + (r) f_j(x, y)$$

For each feature j not in true y, but in predicted y*:

$$\theta_j := \theta_j - (r) f_j(x, y)$$

finna get good

gold y = V V A

Two simple feature templates

"Transition features"

$$f_{trans:A,B}(x,y) = \sum_t 1\{y_{t-1} = A, y_t = B\}$$

"Observation features"

$$f_{emit:A,w}(x,y) = \sum_t 1\{y_t = A, x_t = w\}$$

f(x,y) is...

V,V: 1
V,A: 1
V,N: 0
....
V,finna: 1
V,get: 1
A,good: 1
N,good: 0
...

	finna	get	good
gold y =	V	V	A

Mathematical convention is numeric indexing, though sometimes convenient to implement as hash table.

Transition features

Observation features

θ	-0.6	-1.0	1.1	0.5	0.0	0.8	0.5	-1.3	-1.6	0.0	0.6	0.0	-0.2	-0.2	0.8	-1.0	0.1	-1.9	1.1	1.2	-0.1	-1.0	-0.1	-0.1
$f(x,y)$	1	0	2	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	3	0	0	0	0

$$f_{trans:V,A}(x,y) = \sum_{t=2}^{N} 1\{y_{t-1} = V, y_t = A\}$$

$$f_{obs:V,\text{finna}}(x,y) = \sum_{t=1}^{N} 1\{y_t = V, x_t = \text{finna}\}$$

Goodness(y) = $\theta^{\mathsf{T}} f(x,y)$

	finna	get	good
gold y =	V	V	A
pred y^* =	N	V	A

Learning idea: want gold y to have high scores.
Update weights so y would have a higher score, and y* would be lower, next time.

$f(x, y)$

V,V: 1
V,A: 1
V,finna: 1
V,get: 1
A,good: 1

$f(x, y^*)$

N,V: 1
V,A: 1
N,finna: 1
V,get: 1
A,good: 1

$f(x,y) - f(x, y^*)$

V,V: +1
N,V: -1
V,finna: +1
N,finna: -1

Perceptron update rule:

$$\theta := \theta + r[f(x,y) - f(x,y^*)]$$

$$\theta := \theta + r[f(x,y) - f(x,y^*)]$$

	Transition features							Observation features															
θ	-0.6	-1.0	1.1	0.5	0.0	0.8	0.5	-1.3	-1.6	0.0	0.6	0.0	-0.2	-0.2	0.8	-1.0	0.1	-1.9	1.1	1.2	-0.1	-1.0	-0.1
$f(x,y)$	1	0	**2**	0	0	1	0	0	0	0	0	0	0	1	0	0	0	**0**	0	3	0	0	0
$f(x,y^*)$	1	0	**1**	0	0	1	0	0	0	0	0	0	0	1	0	0	0	**1**	0	3	0	0	0

The update vector:

+ r (			**+1**															**-1**						)

Perceptron notes/issues

- Issue: does it converge? (generally no)
  - Solution: the *averaged* perceptron
- Can you regularize it? No... just averaging...

- By the way, there's also *likelihood* training out there (gradient ascent on the log-likelihood function: the traditional way to train a CRF)
  - structperc is easier to implement/conceptualize and performs similarly in practice

Averaged perceptron

- To get stability for the perceptron: *Voted* perc or *Averaged* perc
- See HW2 writeup
- Averaging: For *t*'th example... average together vectors from every timestep

$$\bar{\theta}_t = \frac{1}{t} \sum_{t'=1}^{t} \theta_{t'}$$

- Efficiency?
  - Lazy update algorithm in HW