Easy Quiz Question!

- Suppose we have a 1 feature maxent model built over observed data as shown.
- What is the constructed model's probability distribution over the four possible outcomes?

Empirical

	A	a
B	2	1
b	2	1

Features

	A	a
B		
b	🟥	🟥

Expectations

Probabilities

	A	a
B		
b		

Smoothing: Issues of Scale

- Lots of features:
  - NLP maxent models can have over 1M features.
  - Even storing a single array of parameter values can have a substantial memory cost.
- Lots of sparsity:
  - Overfitting very easy – need smoothing!
  - Many features seen in training will never occur again at test time.
- Optimization problems:
  - Feature weights can be infinite, and iterative solvers can take a long time to get to those infinities.

Smoothing: Issues

- Assume the following empirical distribution:

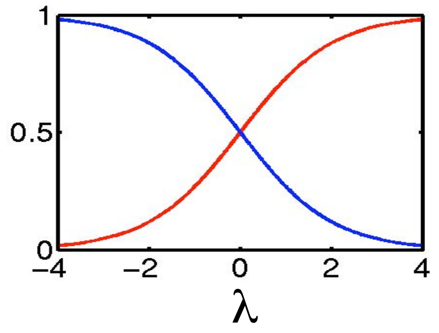
Heads	Tails
h	t

- Features: {Heads}, {Tails}
- We'll have the following model distribution:

$$p_{\text{HEADS}} = \frac{e^{\lambda_{\text{H}}}}{e^{\lambda_{\text{H}}} + e^{\lambda_{\text{T}}}} \qquad p_{\text{TAILS}} = \frac{e^{\lambda_{\text{T}}}}{e^{\lambda_{\text{H}}} + e^{\lambda_{\text{T}}}}$$

- Really, only one degree of freedom ($\lambda = \lambda_H - \lambda_T$)

$$p_{\text{HEADS}} = \frac{e^{\lambda_{\text{H}}} e^{-\lambda_{\text{T}}}}{e^{\lambda_{\text{H}}} e^{-\lambda_{\text{T}}} + e^{\lambda_{\text{T}}} e^{-\lambda_{\text{T}}}} = \frac{e^{\lambda}}{e^{\lambda} + e^{0}} \qquad p_{\text{TAILS}} = \frac{e^{0}}{e^{\lambda} + e^{0}}$$

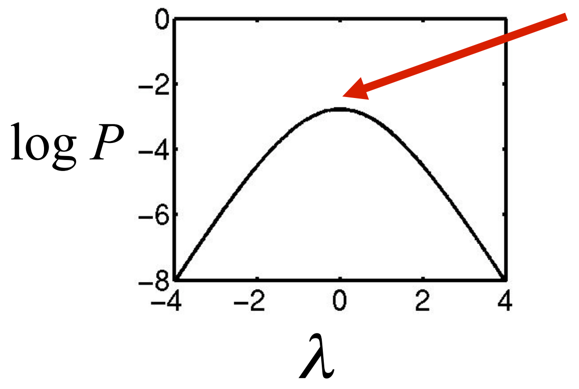


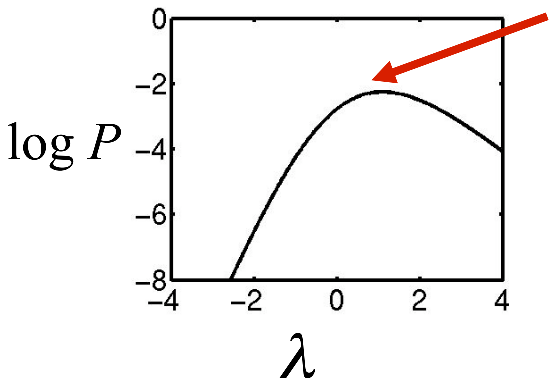
Smoothing: Issues

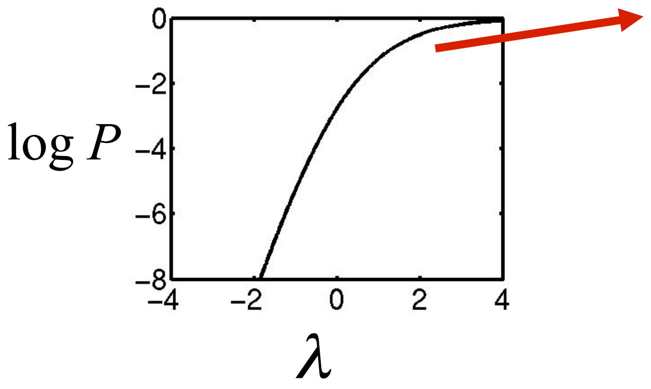
- The data likelihood in this model is:

$$\log P(h,t \mid \lambda) = h \log p_{\text{HEADS}} + t \log p_{\text{TAILS}}$$

$$\log P(h,t \mid \lambda) = h\lambda - (t+h)\log(1+e^{\lambda})$$







Heads	Tails
2	2

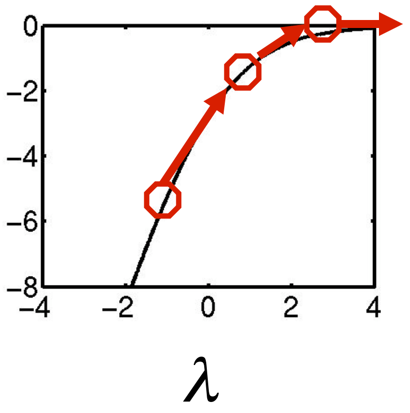
Heads	Tails
3	1

Heads	Tails
4	0

Smoothing: Early Stopping

- In the 4/0 case, there were two problems:
  - The optimal value of λ was ∞, which is a long trip for an optimization procedure.
  - The learned distribution is just as spiked as the empirical one – no smoothing.

- One way to solve both issues is to just stop the optimization early, after a few iterations.
  - The value of λ will be finite (but presumably big).
  - The optimization won't take forever (clearly).
  - Commonly used in early maxent work.



Heads	Tails
4	0

Input

Heads	Tails
1	0

Output

Smoothing: Priors (MAP)

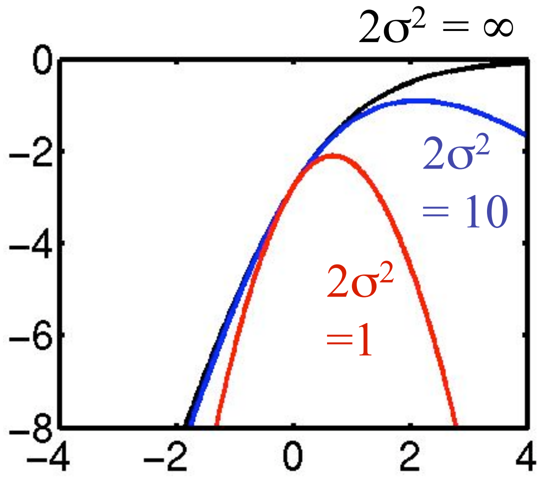
- What if we had a prior expectation that parameter values wouldn't be very large?
- We could then balance evidence suggesting large parameters (or infinite) against our prior.
- The evidence would never totally defeat the prior, and parameters would be smoothed (and kept finite!).
- We can do this explicitly by changing the optimization objective to maximum posterior likelihood:

$$\log P(C,\lambda \mid D) = \log P(\lambda) + \log P(C \mid D,\lambda)$$

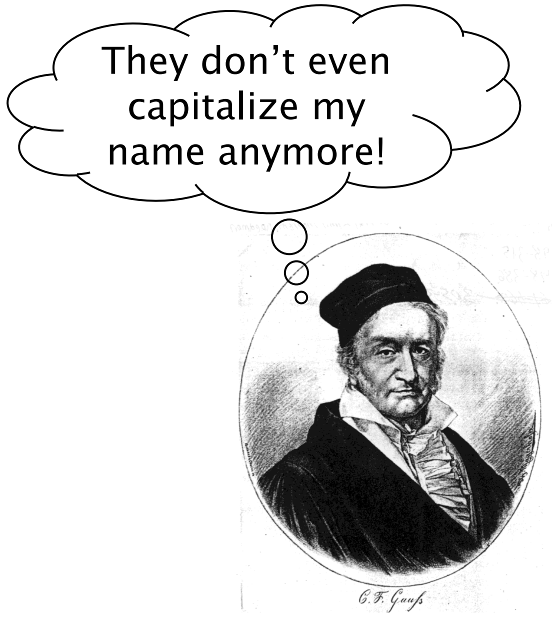
Posterior Prior Evidence

Smoothing: Priors

- Gaussian, or quadratic, or L2 priors:
  - Intuition: parameters shouldn't be large.
  - Formalization: prior expectation that each parameter will be distributed according to a gaussian with mean μ and variance σ^2.



$$P(\lambda_i) = \frac{1}{\sigma_i\sqrt{2\pi}}\exp\left(-\frac{(\lambda_i-\mu_i)^2}{2\sigma_i^2}\right)$$



  - Penalizes parameters for drifting to far from their mean prior value (usually $\mu=0$).
  - $2\sigma^2=1$ works surprisingly well.

Example: NER Smoothing

Because of smoothing, the more common prefix and single-tag features have larger weights even though entire-word and tag-pair features are more specific.

Local Context

	Prev	Cur	Next
State	Other	???	???
Word	at	Grace	Road
Tag	IN	NNP	NNP
Sig	x	Xx	Xx

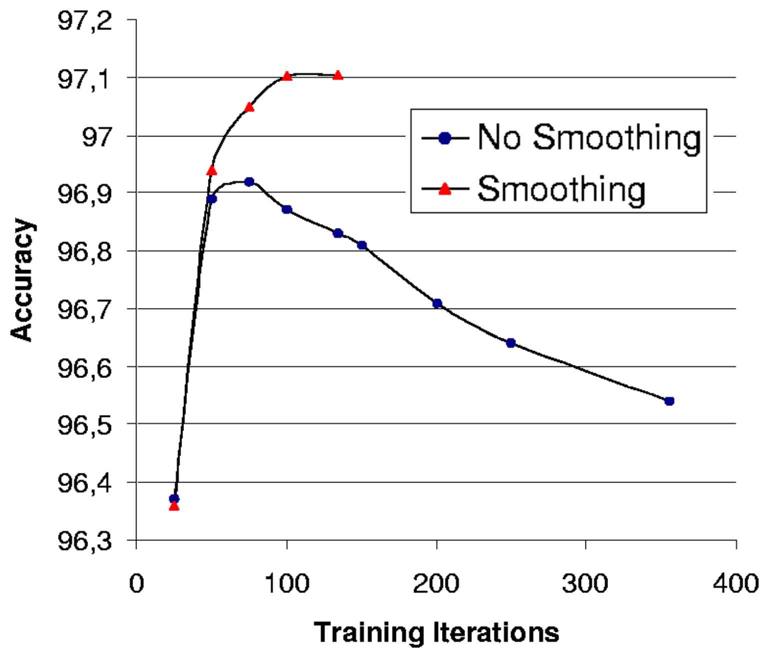
Feature Weights

Feature Type	Feature	PERS	LOC
Previous word	*at*	-0.73	0.94
Current word	*Grace*	0.03	0.00
Beginning bigram	*<G*	0.45	-0.04
Current POS tag	NNP	0.47	0.45
Prev and cur tags	IN NNP	-0.10	0.14
Previous state	Other	-0.70	-0.92
Current signature	Xx	0.80	0.46
Prev state, cur sig	O-Xx	0.68	0.37
Prev-cur-next sig	x-Xx-Xx	-0.69	0.37
P. state - p-cur sig	O-x-Xx	-0.20	0.82
…			
Total:		**-0.58**	**2.68**

Example: POS Tagging

- From (Toutanova et al., 2003):

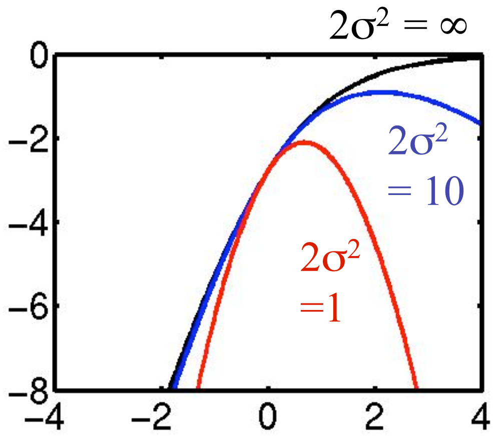
	Overall Accuracy	Unknown Word Acc
Without Smoothing	96.54	85.20
With Smoothing	97.10	88.20



- Smoothing helps:
  - Softens distributions.
  - Pushes weight onto more explanatory features.
  - Allows many features to be dumped safely into the mix.
  - Speeds up convergence (if both are allowed to converge)!

Smoothing: Priors

- If we use gaussian priors:
  - Trade off some expectation-matching for smaller parameters.
  - When multiple features can be recruited to explain a data point, the more common ones generally receive more weight.
  - Accuracy generally goes up!
- Change the objective:



$$\log P(C,\lambda \mid D) = \log P(C \mid D,\lambda) - \log P(\lambda)$$

$$\log P(C,\lambda \mid D) = \sum_{(c,d)\in(C,D)} P(c \mid d,\lambda) - \sum_i \frac{(\lambda_i - \mu_i)^2}{2\sigma_i^2} + k$$

- Change the derivative:

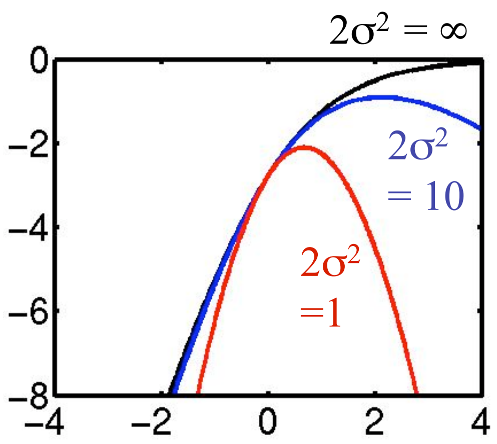
$$\partial \log P(C,\lambda \mid D) / \partial\lambda_i = \text{actual}(f_i, C) - \text{predicted}(f_i,\lambda) - (\lambda_i - \mu_i)/\sigma^2$$

Smoothing: Priors

- If we use gaussian priors:
  - Trade off some expectation-matching for smaller parameters.
  - When multiple features can be recruited to explain a data point, the more common ones generally receive more weight.
  - Accuracy generally goes up!
- Change the objective:

$$\log P(C,\lambda \mid D) = \log P(C \mid D,\lambda) \; - \log P(\lambda)$$

$$\log P(C,\lambda \mid D) = \sum_{(c,d)\in(C,D)} P(c \mid d,\lambda) \quad - \sum_i \frac{\lambda_i^2}{2\sigma_i^2} + k$$

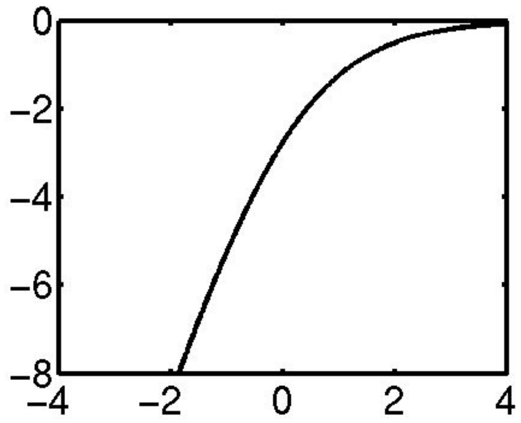


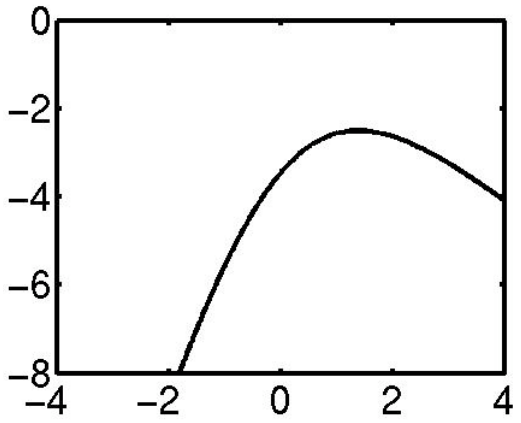
- Change the derivative:

$$\partial \log P(C,\lambda \mid D) / \partial \lambda_i = \text{actual}(f_i, C) - \text{predicted}(f_i, \lambda) \quad - \lambda_i / \sigma^2$$

Smoothing: Virtual Data

- Another option: smooth the data, not the parameters.
- Example:





Heads	Tails
4	0

$\rightarrow$

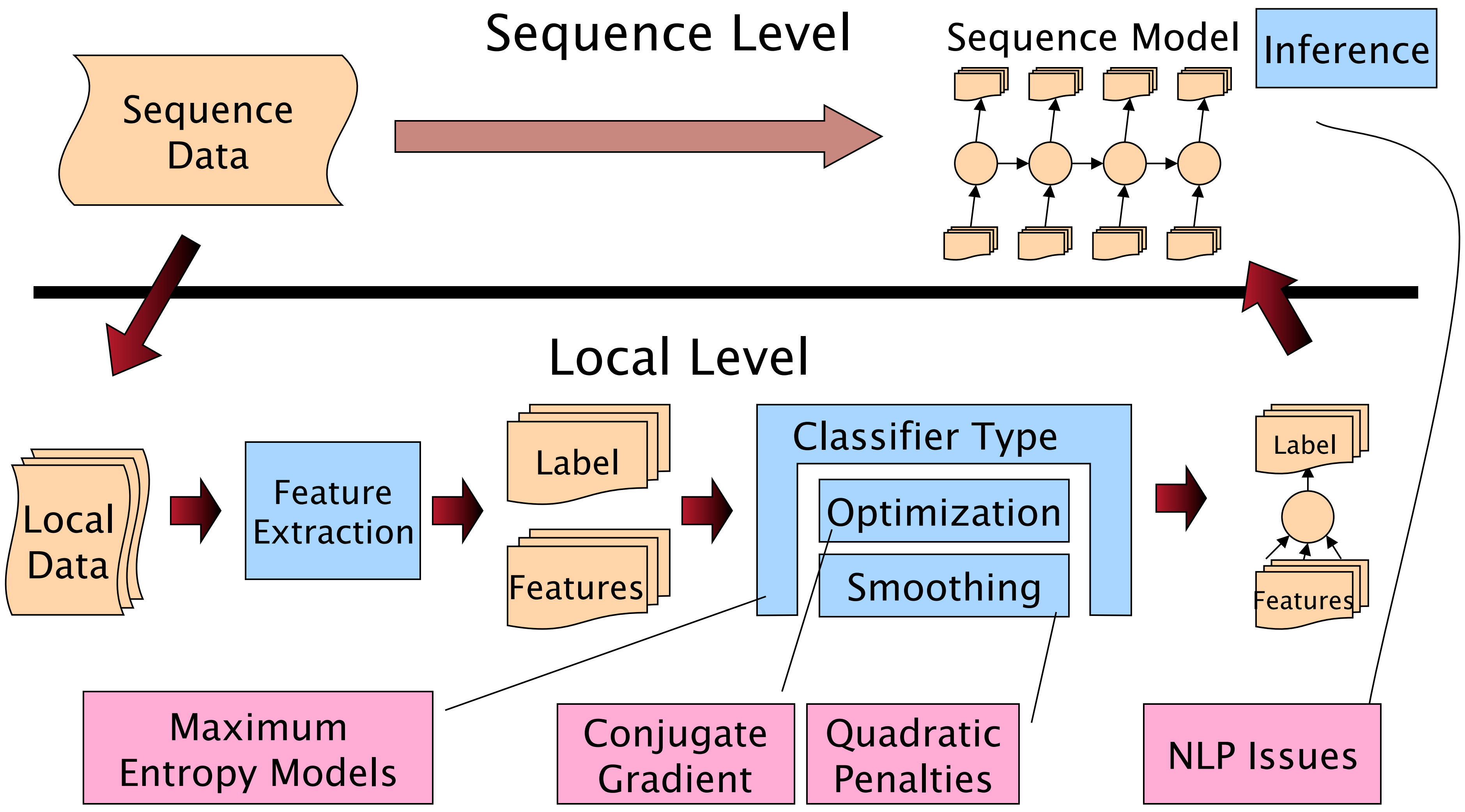
Heads	Tails
5	1

  - Equivalent to adding two extra data points.
  - Similar to add-one smoothing for generative models.
- Hard to know what artificial data to create!

Smoothing: Count Cutoffs

- In NLP, features with low empirical counts were usually dropped.
  - Very weak and indirect smoothing method.
  - Equivalent to locking their weight to be zero.
  - Equivalent to assigning them gaussian priors with mean zero and variance zero.
  - Dropping low counts does remove the features which were most in need of smoothing…
  - … and speeds up the estimation by reducing model size …
  - … but count cutoffs generally hurt accuracy in the presence of proper smoothing.
- We recommend: don't use count cutoffs unless absolutely necessary.

Inference in Systems



MEMM inference in systems

- For a Conditional Markov Model (CMM) a.k.a. a Maximum Entropy Markov Model (MEMM), the classifier makes a single decision at a time, conditioned on evidence from observations and previous decisions.
- A larger space of sequences is explored via search

Decision Point

Local Context

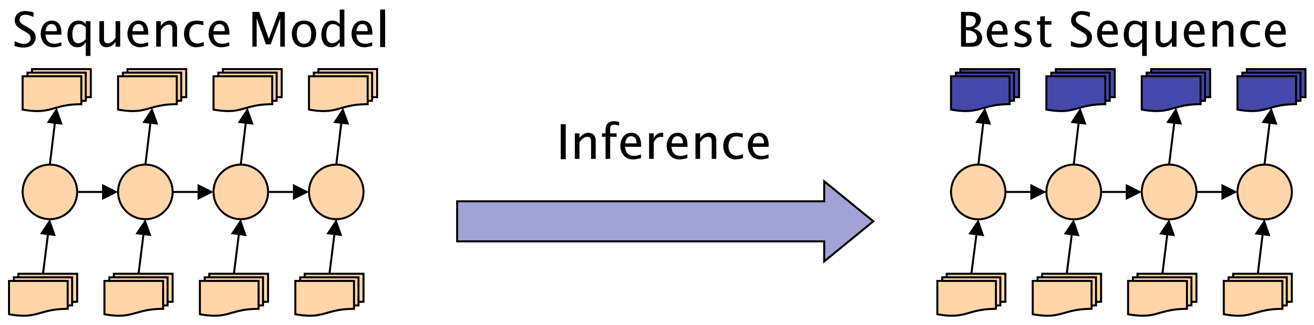
-3	-2	-1	0	+1
DT	NNP	VBD	???	???
The	Dow	fell	22.6	%

Features

W_0	22.6
W_{+1}	%
W_{-1}	fell
T_{-1}	VBD
T_{-1}-T_{-2}	NNP-VBD
hasDigit?	true
…	…

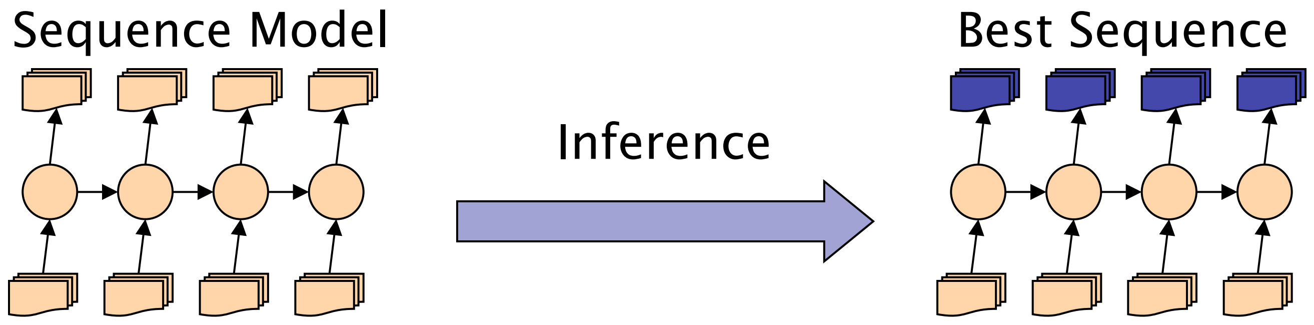
(Ratnaparkhi 1996; Toutanova et al. 2003, etc.)

Beam Inference



- Beam inference:
  - At each position keep the top k complete sequences.
  - Extend each sequence in each local way.
  - The extensions compete for the k slots at the next position.
- Advantages:
  - Fast; and beam sizes of 3–5 are as good or almost as good as exact inference in many cases.
  - Easy to implement (no dynamic programming required).
- Disadvantage:
  - Inexact: the globally best sequence can fall off the beam.

Viterbi Inference



- Viterbi inference:
  - Dynamic programming or memoization.
  - Requires small window of state influence (e.g., past two states are relevant).
- Advantage:
  - Exact: the global best sequence is returned.
- Disadvantage:
  - Harder to implement long-distance state-state interactions (but beam inference tends not to allow long-distance resurrection of sequences anyway).

Viterbi Inference: J&M Ch. 6

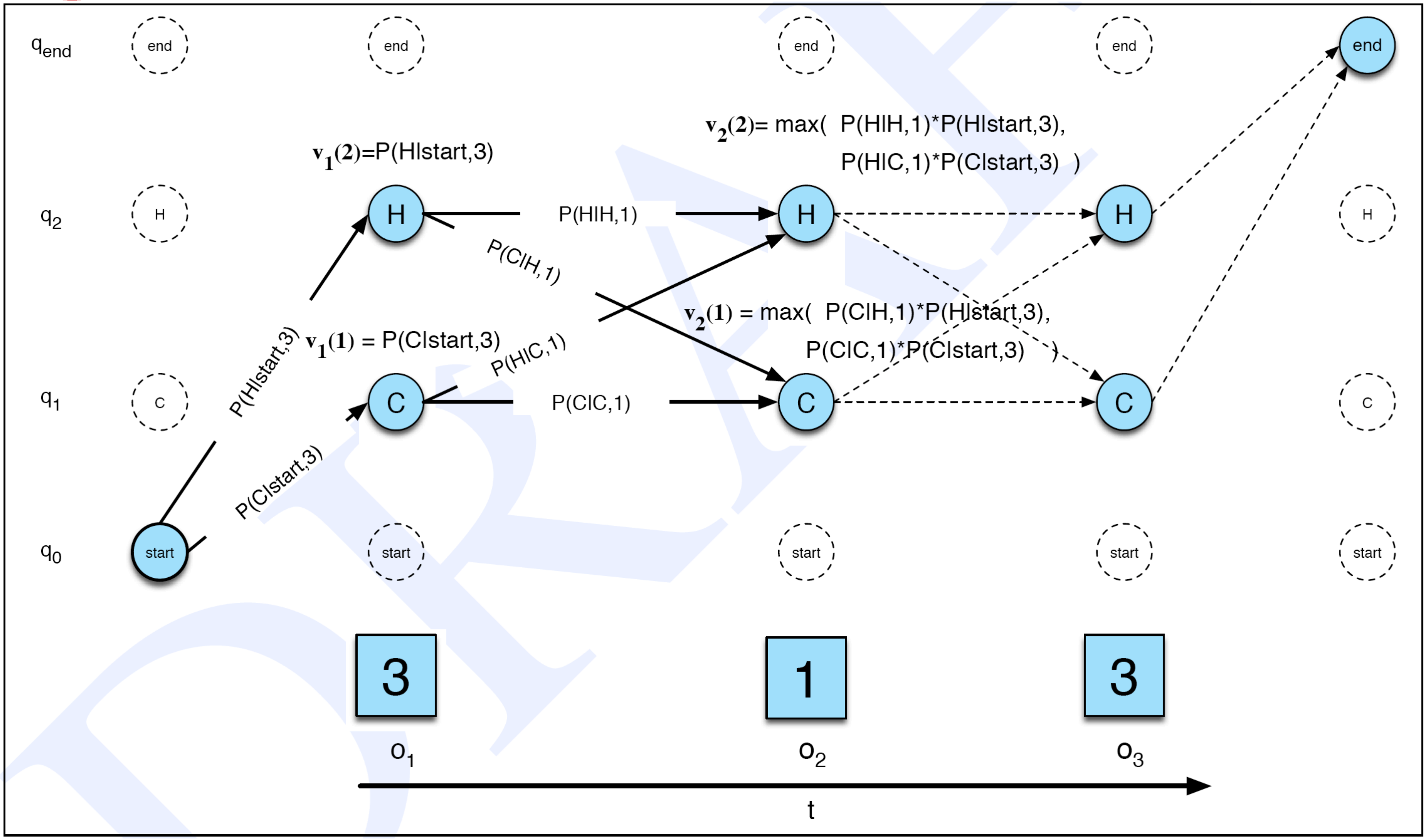
- I'm basically punting on this … read Ch. 6.
  - I'll do dynamic programming for parsing
- It's a small change from HMM Viterbi
  - From:

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, P(s_j|s_i)\, P(o_t|s_j) \quad 1 \leq j \leq N, 1 < t \leq T$$

  - To:

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, P(s_j|s_i, o_t) \quad 1 \leq j \leq N, 1 < t \leq T$$

Viterbi Inference: J&M Ch. 6



CRFs [Lafferty, Pereira, and McCallum 2001]

- Another sequence model: Conditional Random Fields (CRFs)
- A whole-sequence conditional model rather than a chaining of local models.

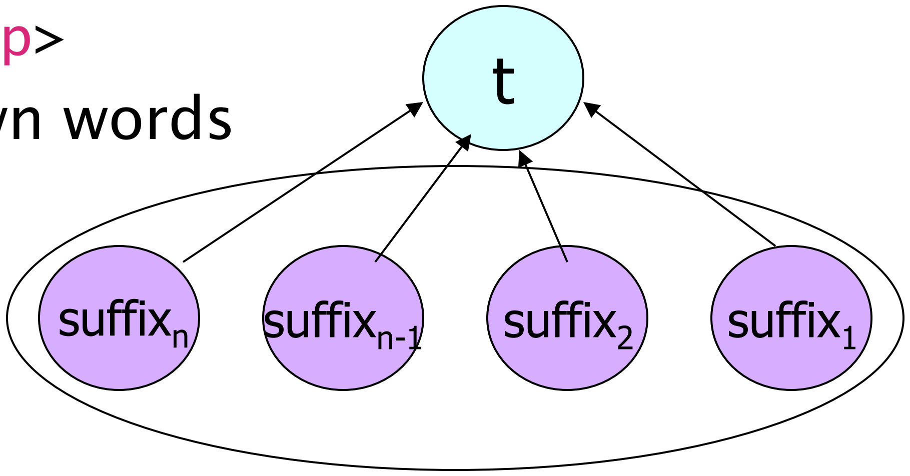
$$P(c \mid d, \lambda) = \frac{\exp \sum_i \lambda_i f_i(c, d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c', d)}$$

- The space of c's is now the space of sequences
  - But if the features f_i remain local, the conditional sequence likelihood can be calculated exactly using dynamic programming
- Training is slow, but CRFs avoid causal-competition biases
- These (or a variant using a max margin criterion) are seen as the state-of-the-art these days

HMM Tagging Models - Brants 2000

- Highly competitive with other state-of-the art models
- Trigram HMM with smoothed transition probabilities
- Capitalization feature becomes part of the state – each tag state is split into two e.g.

  NN → <NN,cap>,<NN,not cap>
- Suffix features for unknown words



$$P(w \mid tag) = P(suffix \mid tag)(w \mid suffix)$$
$$\approx \hat{P}(suffix)\tilde{P}(tag \mid suffix) / \hat{P}(tag)$$

$$\tilde{P}(tag \mid suffix_n) = \lambda_1 \hat{P}(tag \mid suffix_n) + \lambda_2 \hat{P}(tag \mid suffix_{n-1}) + \ldots + \lambda_n \hat{P}(tag)$$

MEMM Tagging Models -II

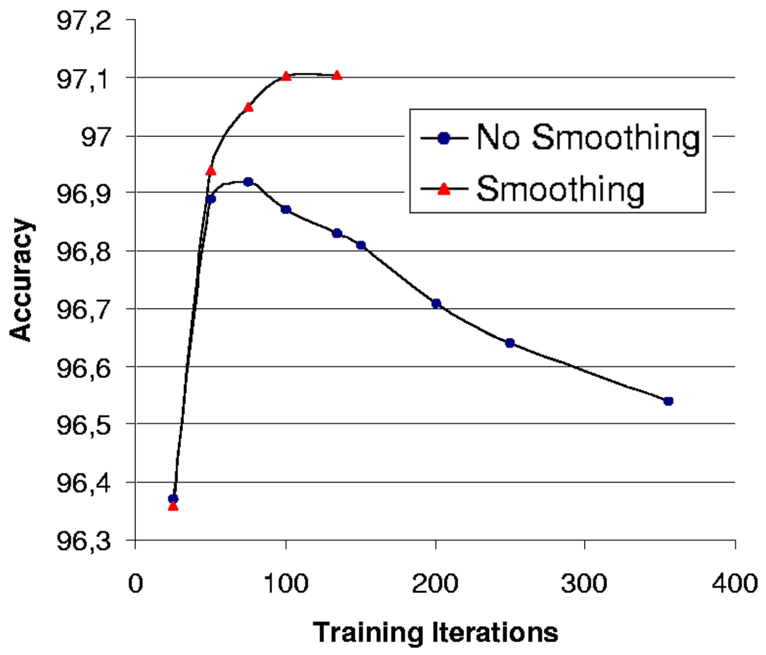
- Ratnaparkhi (1996): local distributions are estimated using maximum entropy models
  - Previous two tags, current word, previous two words, next two words, suffix, prefix, hyphenation, and capitalization features for unknown words
- Toutanova et al. (2003)
  - Richer features, bidirectional inference, better smoothing, better unknown word handling

Model	Overall Accuracy	Unknown Words
HMM (Brants 2000)	96.7	85.5
MEMM (Ratn. 1996)	96.63	85.56
MEMM (T. et al 2003)	97.24	89.04

Smoothing: POS Tagging

- From (Toutanova et al., 2003):

	Overall Accuracy	Unknown Word Acc
Without Smoothing	96.54	85.20
With Smoothing	97.10	88.20



- Smoothing helps:
  - Softens distributions.
  - Pushes weight onto more explanatory features.
  - Allows many features to be dumped safely into the mix.
  - Speeds up convergence (if both are allowed to converge)!

Summary of Tagging

For tagging, the change from generative to discriminative model **does not by itself** result in great improvement

One profits from discriminative models for specifying dependence on **overlapping features of the observation** such as spelling, suffix analysis,etc

A CMM allows integration of rich features of the observations, but can suffer strongly from assuming independence from following observations; this effect can be relieved by adding dependence on following words

This additional power (of the CMM ,CRF, Perceptron models) has been shown to result in improvements in accuracy

The **higher accuracy** of discriminative models comes at the price of **much slower training**