Sequence-to-Sequence

Consider the problem of jointly modeling a pair of strings

- E.g.: part of speech tagging

Y:	DT	NNP	NN	VBD	VBN	RP	NN	NNS
X:	The	Georgia	branch	had	taken	on	loan	commitments …
	DT	NN	IN	NN		VBD	NNS	VBD
X':	The	average	of	interbank		offered	rates	plummeted …

Q: How do we map each word in the input sentence onto the appropriate label?

A: We can learn a joint distribution:

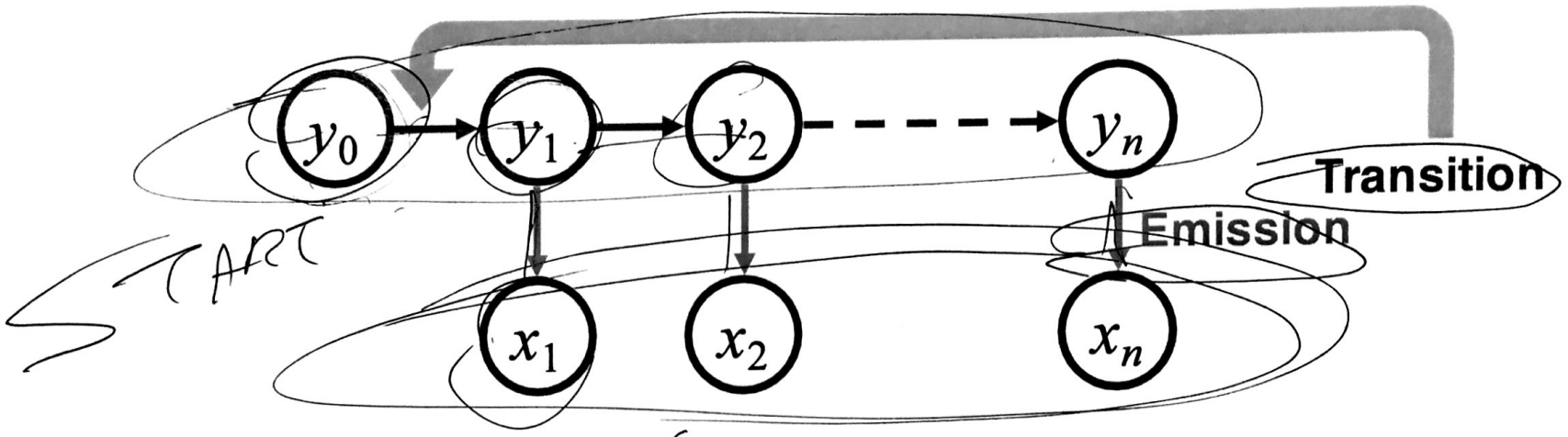
$$p(x_1 \dots x_n, y_1 \dots y_n)$$

And then compute the most likely assignment:

$$\arg\max_{y_1 \dots y_n} p(x_1 \dots x_n, y_1 \dots y_n)$$

Classic Solution: HMMs

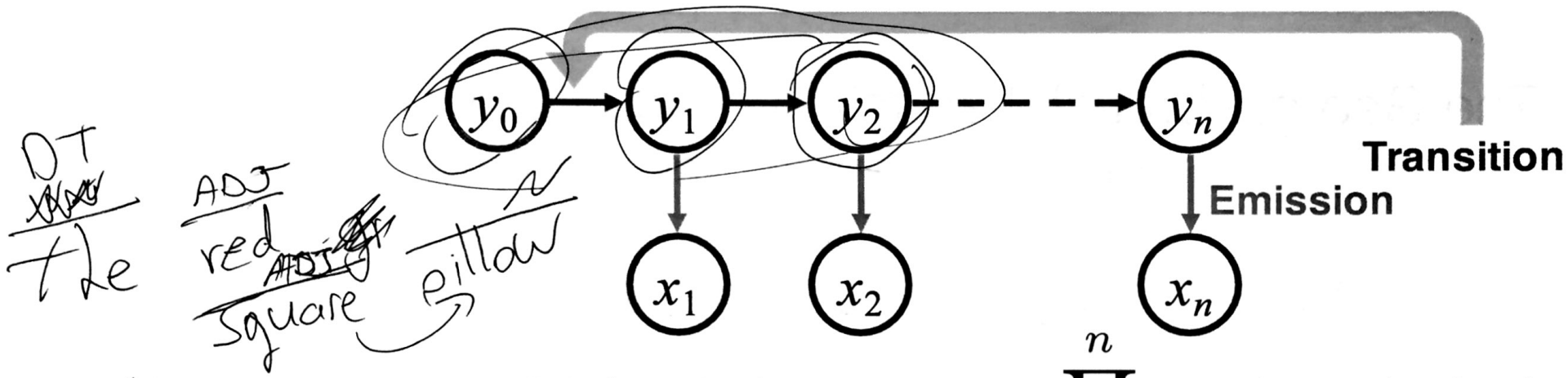
We want a model of sequences y and observations x



$$p(x_1 \ldots x_n, y_1 \ldots y_n) = q(y_0) \sum_i q(y_i | y_{i-1}) e(x_i | y_i)$$

where y_0=*START* and we call $q(y_i|y_{i-1})$ the **transition** distribution and $e(x_i|y_i)$ the **emission** (or observation) distribution.

Model Assumptions



$$p(x_1 \dots x_n, y_1 \dots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

- Tag/state sequence is generated by a Markov model
- Words are chosen independently, conditioned only on the tag/state
- These are totally broken assumptions: why? Examples? Think prepositions …

HMM for POS Tagging

The Georgia branch had taken on loan commitments …

DT NNP NN VBD VBN RP NN NNS

- HMM Model:
  - States Y = Tags DT, NNP, . . .
  - Observations X = V
  - Transition dist'n q(yi |yi -1) models → tags to follow one another
  - Emission dist'n e(xi |yi) models → show words are assigned tags

HMM Inference and Learning

- Learning
  - Maximum likelihood: transitions q and emissions e

$$p(x_1 \dots x_n, y_1 \dots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

- Inference
  - Viterbi

$$y^* = \arg \max_{y_1 \dots y_n} p(x_1 \dots x_n, y_1 \dots y_n)$$

  - Forward backward (for posterior marginals)

$$p(x_1 \dots x_n, y_i) = \sum_{y_1 \dots y_{i-1}} \sum_{y_{i+1} \dots y_n} p(x_1 \dots x_n, y_1 \dots y_n)$$

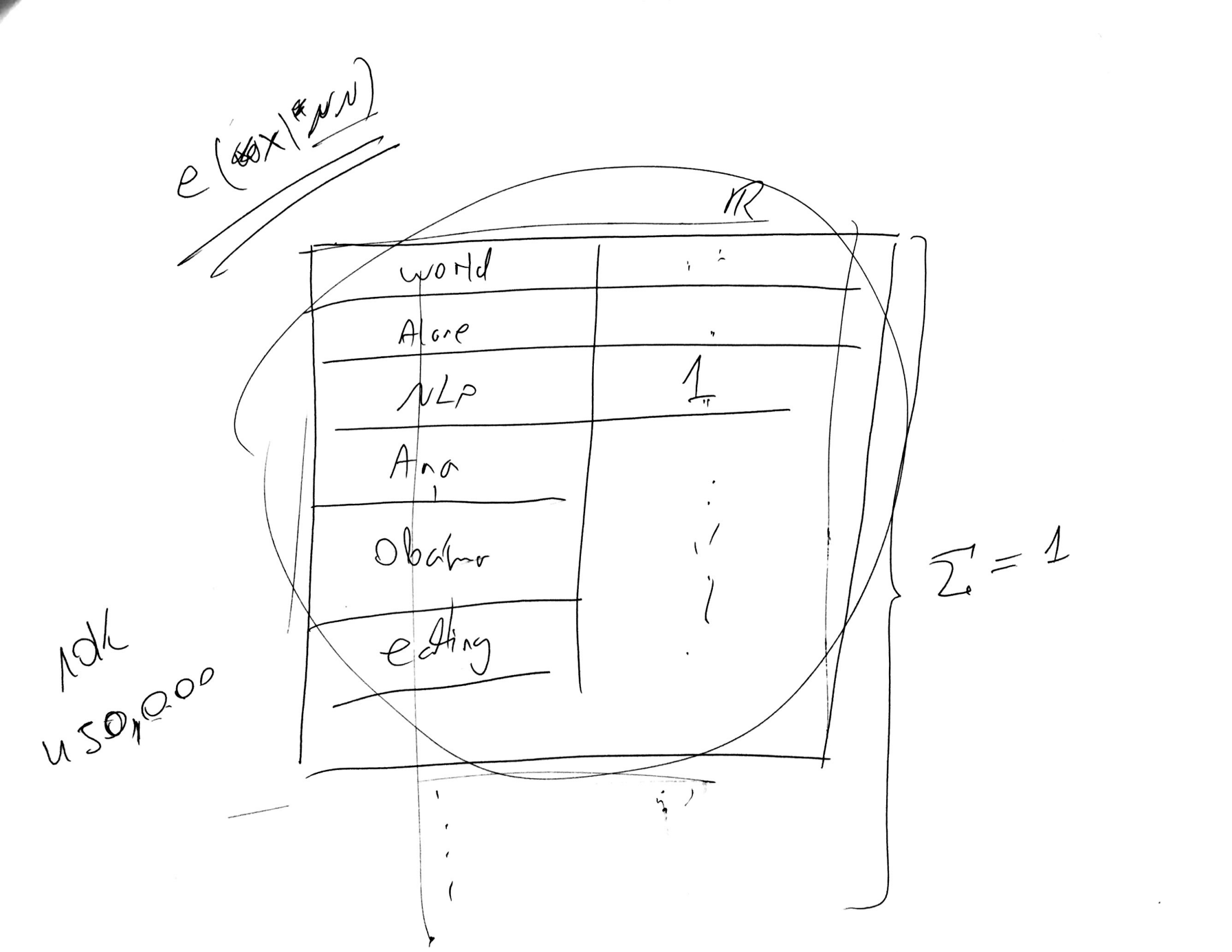
Learning: Maximum Likelihood

$$p(x_1 \dots x_n, y_1 \dots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

- Maximum likelihood methods for estimating transitions q and emissions e

$$q_{ML}(y_i|y_{i-1}) = \frac{c(y_{i-1}, y_i)}{c(y_{i-1})} \qquad e_{ML}(x|y) = \frac{c(y, x)}{c(y)}$$

- Will these estimates be high quality?
  - Which is likely to be more sparse, q or e?
- Smoothing?


e(
R
world
Alone
NLP
1
Ang
Obama
editing
Σ = 1
10k
US$1,000

Learning: Low Frequency Words

$$p(x_1 \ldots x_n, y_1 \ldots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

- Typically, for transitions:
  - Linear Interpolation

much smaller table

$$q(y_i|y_{i-1}) = \lambda_1 q_{ML}(y_i|y_{i-1}) + \lambda_2 q_{ML}(y_i)$$

valid prob. dist

Learning: Low Frequency Words

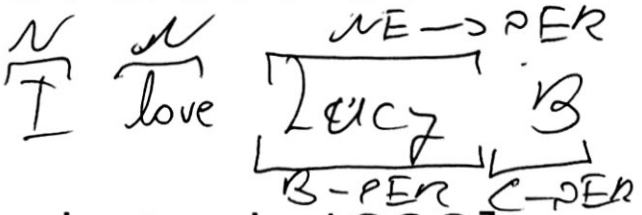
$$p(x_1 \dots x_n, y_1 \dots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

- Typically, for transitions:
  - Linear Interpolation

$$q(y_i|y_{i-1}) = \lambda_1 q_{ML}(y_i|y_{i-1}) + \lambda_2 q_{ML}(y_i)$$

- However, other approaches used for emissions
  - **Step 1:** Split the vocabulary
    - Frequent words: appear more than M (often 5) times
    - Low frequency: everything else
  - **Step 2:** Map each low frequency word to one of a small, finite set of possibilities
    - For example, based on prefixes, suffixes, etc.
  - **Step 3:** Learn model for this new space of possible word sequences

Low Frequency Words: An Example



- Named Entity Recognition [Bickel et. al, 1999]
  - Used the following word classes for infrequent words:

Word class	Example	Intuition
twoDigitNum	90	Two digit year
fourDigitNum	1990	Four digit year
containsDigitAndAlpha	A8956-67	Product code
containsDigitAndDash	09-96	Date
containsDigitAndSlash	11/9/89	Date
containsDigitAndComma	23,000.00	Monetary amount
containsDigitAndPeriod	1.00	Monetary amount,percentage
othernum	456789	Other number
allCaps	BBN	Organization
capPeriod	M.	Person name initial
firstWord	first word of sentence	no useful capitalization information
initCap	Sally	Capitalized word
lowercase	can	Uncapitalized word
other	,	Punctuation marks, all other words

Low Frequency Words: An Example

Profits/NA soared/NA at/NA Boeing/SO Co./CO ,/NA easily/NA topping/NA forecasts/NA on/NA Wall/SL Street/CL ,/NA as/NA their/NA CEO/NA Alan/SP Mulally/CP announced/NA first/NA quarter/NA results/NA ./NA

firstword/NA soared/NA at/NA initCap/SC Co./CC ,/NA easily/NA lowercase/NA forecasts/NA on/NA initCap/SL Street/CL ,/NA as/NA their/NA CEO/NA Alan/SP initCap/CP announced/NA first/NA quarter/NA results/NA ./NA

- NA = No entity
- SO = Start Organization
- CO = Continue Organization
- SL = Start Location
- CL = Continue Location
- …

Inference (Decoding)

- Problem: find the most likely (Viterbi) sequence under the model

$$y^* = \arg\max_{y_1 \ldots y_n} p(x_1 \ldots x_n, y_1 \ldots y_n)$$

- Given model parameters, we can score any sequence pair

NNP	VBZ	NN	NNS	CD	NN	.
Fed	raises	interest	rates	0.5	percent	.

q(NNP|♦) e(Fed|NNP) q(VBZ|NNP) e(raises|VBZ) q(NN|VBZ).....

- In principle, we're done – list all possible tag sequences, score each one, pick the best one (the Viterbi state sequence)

NNP VBZ NN NNS CD NN . ➡ *logP* = -23

NNP NNS NN NNS CD NN . ➡ *logP* = -29

NNP VBZ VB NNS CD NN . ➡ *logP* = -27

Any issue?

HMM Inference and Learning

- Learning + Smoothing
  - Maximum likelihood: transitions q and emissions e

$$p(x_1 \dots x_n, y_1 \dots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

- Inference
  - Viterbi

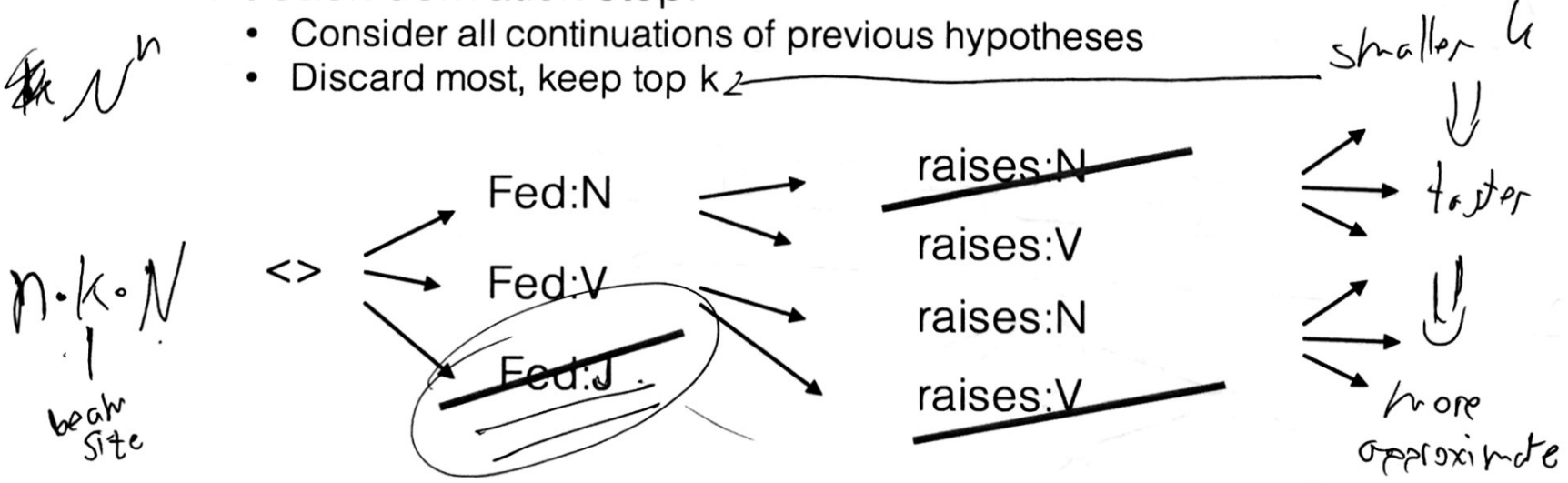
$$y^* = \arg \max_{y_1 \dots y_n} p(x_1 \dots x_n, y_1 \dots y_n)$$

  - Forward backward

$$p(x_1 \dots x_n, y_i) = \sum_{y_1 \dots y_{i-1}} \sum_{y_{i+1} \dots y_n} p(x_1 \dots x_n, y_1 \dots y_n)$$

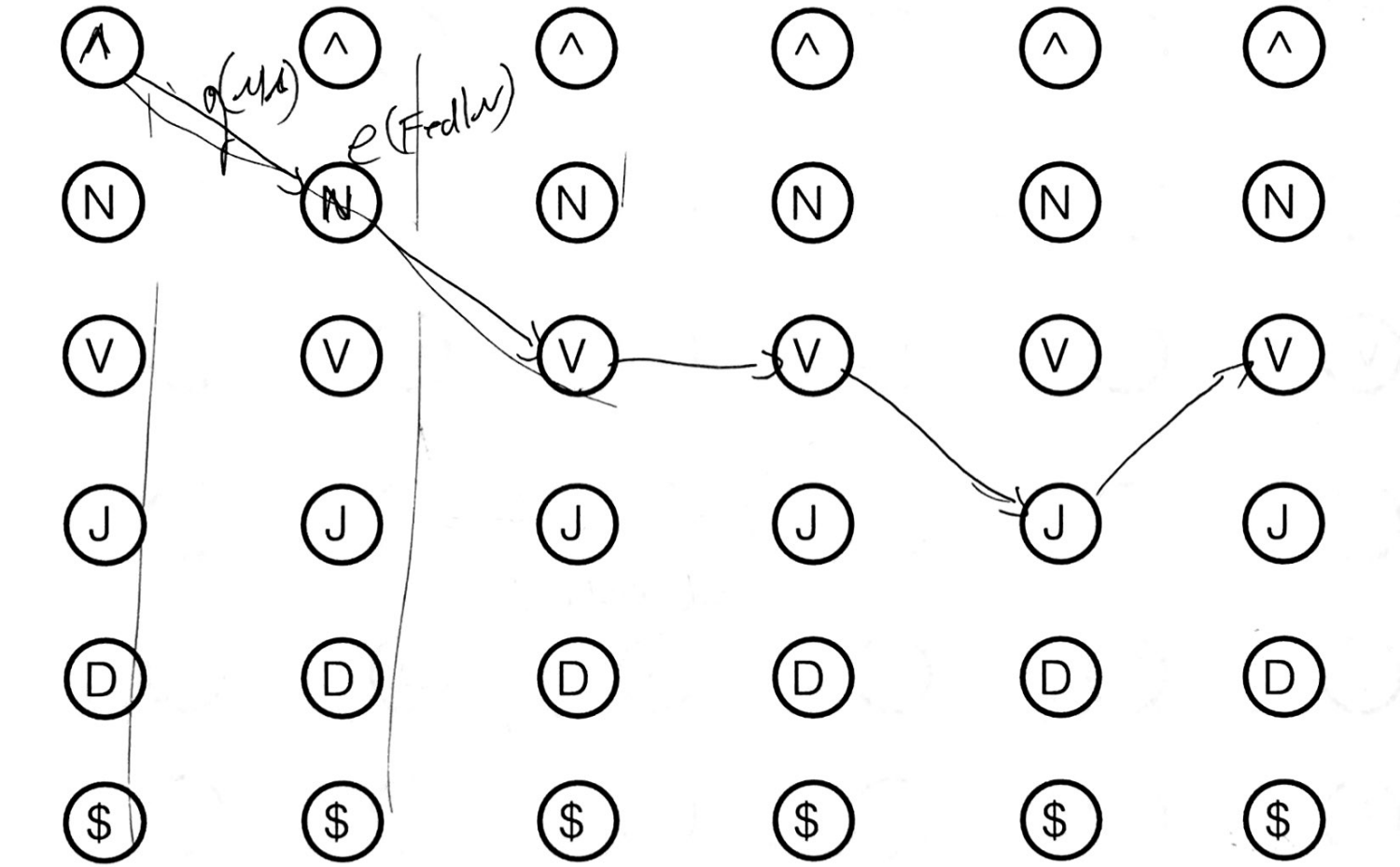
Finding the Best Trajectory

- Too many trajectories (state sequences) to list
- Option 1: Beam Search
  - A beam is a set of partial hypotheses
  - Start with just the single empty trajectory
  - At each derivation step:
    - Consider all continuations of previous hypotheses
    - Discard most, keep top k

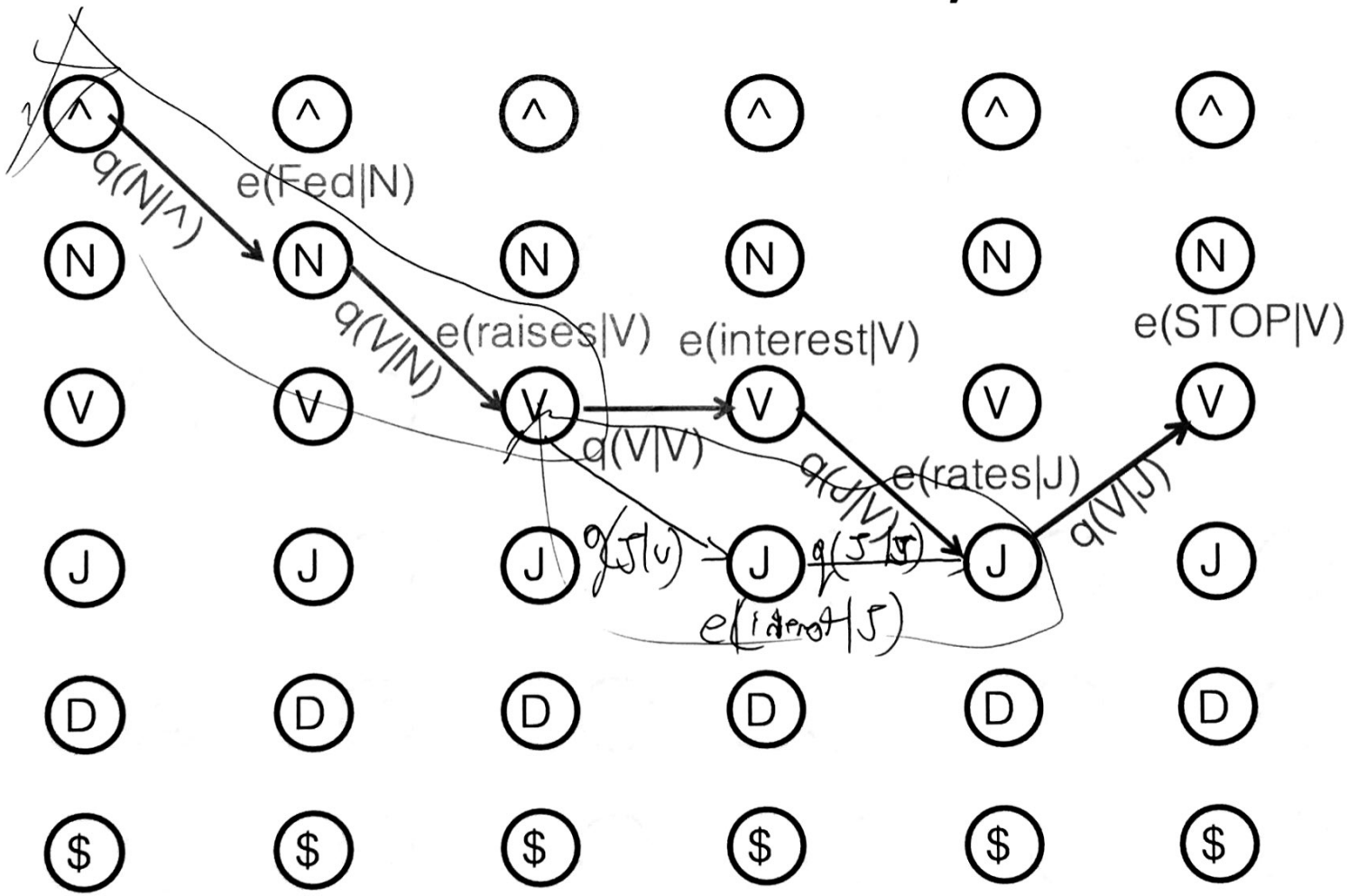


- Beam search often works OK in practice, but …
  - … but sometimes you want the optimal answer
  - … and there's usually a better option than naïve beams

The State Lattice / Trellis



The State Lattice / Trellis



START	Fed	raises	interest	rates	STOP
^	N	V	V->J	J	V

Scoring a Sequence

$$y^* = \arg\max_{y_1 \dots y_n} p(x_1 \dots x_n, y_1 \dots y_n)$$

$$p(x_1 \dots x_n, y_1 \dots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

- Define $\pi(i,y_i)$ to be the max score of a sequence of length i ending in tag y_i

$$\pi(i, y_i) = \max_{y_1 \dots y_{i-1}} p(x_1 \dots x_i, y_1 \dots y_i)$$

$$= \max_{y_{i-1}} e(x_i|y_i) q(y_i|y_{i-1}) \max_{y_1 \dots y_{i-2}} p(x_1 \dots x_{i-1}, y_1 \dots y_{i-1})$$

$$= \max_{y_{i-1}} e(x_i|y_i) q(y_i|y_{i-1}) \pi(i-1, y_{i-1})$$

- We can now design an efficient algorithm.
  - How?

The Viterbi Algorithm

Dynamic program for computing (for all i)

$$\pi(i, y_i) = \max_{y_1 \dots y_{i-1}} p(x_1 \dots x_i, y_1 \dots y_i)$$

Iterative computation:

$$\pi(0, y_0) = \begin{cases} 1 \text{ if } y_0 == START \\ 0 \text{ otherwise} \end{cases}$$

For i = 1 … n:

// Store score

$$\pi(i, y_i) = \max_{y_{i-1}} e(x_i|y_i) q(y_i|y_{i-1}) \pi(i-1, y_{i-1})$$

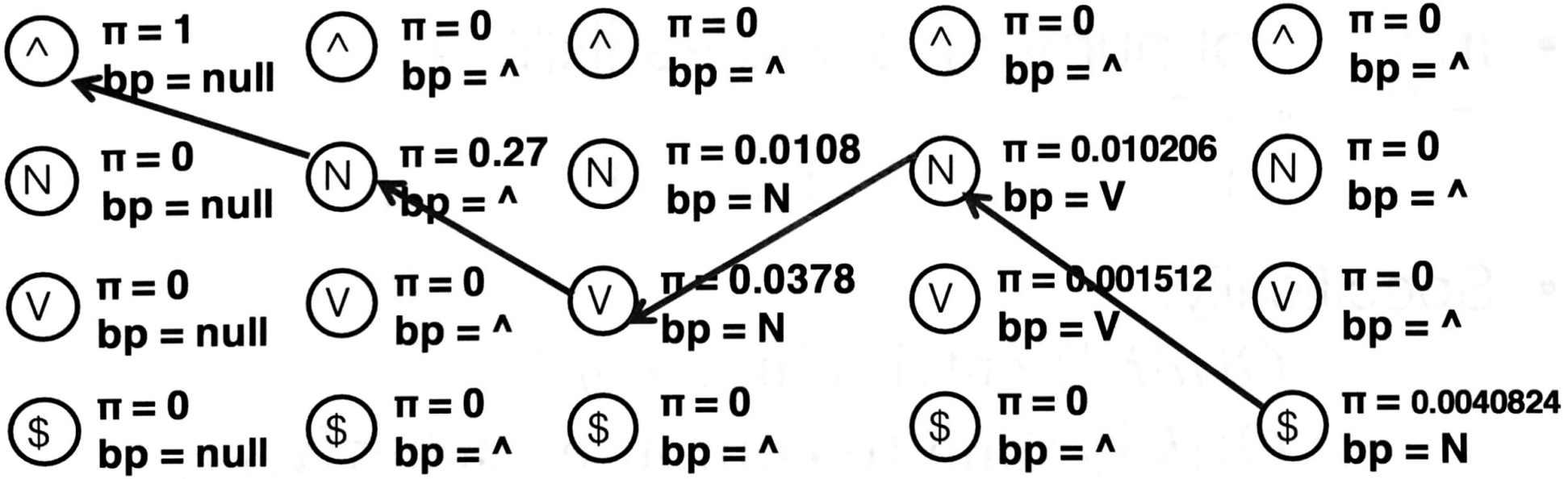
What for?

// Store back-pointer

$$bp(i, y_i) = \arg\max_{y_{i-1}} e(x_i|y_i) q(y_i|y_{i-1}) \pi(i-1, y_{i-1})$$

Tie breaking:
Prefer first

The State Lattice / Trellis



START Fed raises interest STOP

from \ to	^	N	V	$
^	0.0	0.6	0.4	0.0
N	0.0	0.4	0.2	0.4
V	0.0	0.6	0.1	0.3
$	0.0	0.0	0.0	1.0

emissions	START	Fed	raises	interest	STOP
^	1.0	0.0	0.0	0.0	0.0
N	0.0	0.45	0.1	0.45	0.0
V	0.0	0.0	0.7	0.4	0.0
$	0.0	0.0	0.0	0.0	1.0

The Viterbi Algorithm: Runtime

- In term of sentence length n?
  - Linear
- In term of number of states |K|?
  - Polynomial

$$\pi(i, y_i) = \max_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\pi(i-1, y_{i-1})$$

- Specifically:

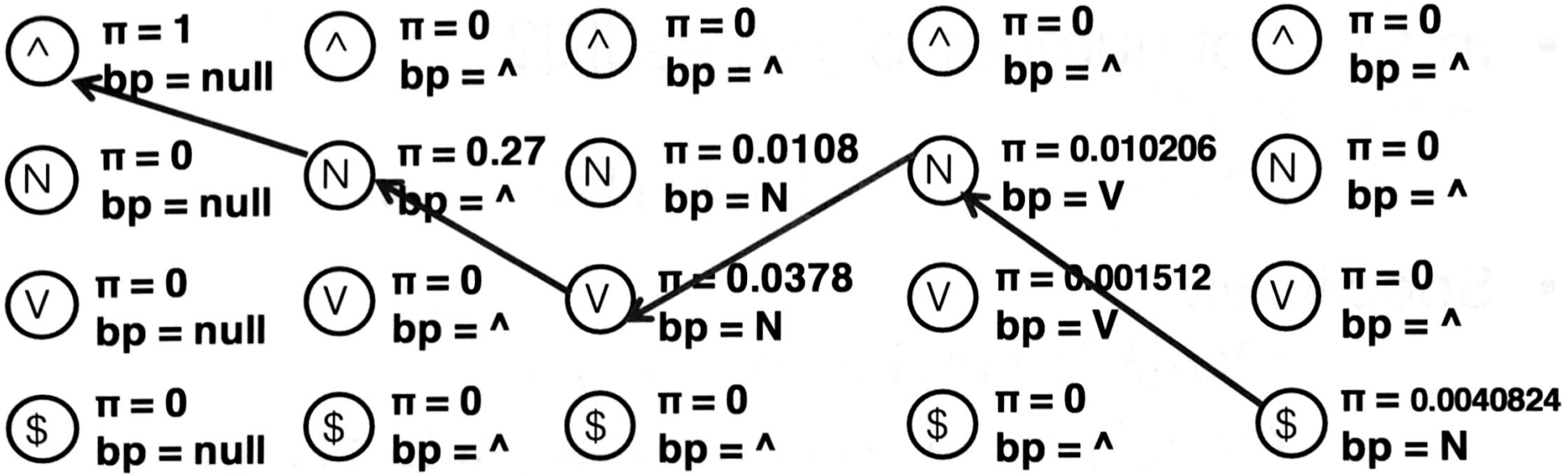
$$O(n|\mathcal{K}|) \text{ entries in } \pi(i, y_i)$$

$$O(|\mathcal{K}|) \text{ time to compute each } \pi(i, y_i)$$

- Total runtime: $O(n|\mathcal{K}|^2)$
- Q: Is this a practical algorithm?
- A: depends on |K|….

Tie breaking:
Prefer first

The State Lattice / Trellis



START Fed raises interest STOP

from \ to	^	N	V	$
^	0.0	0.6	0.4	0.0
N	0.0	0.4	0.2	0.4
V	0.0	0.6	0.1	0.3
$	0.0	0.0	0.0	1.0

emissions	START	Fed	raises	interest	STOP
^	1.0	0.0	0.0	0.0	0.0
N	0.0	0.45	0.1	0.45	0.0
V	0.0	0.0	0.7	0.4	0.0
$	0.0	0.0	0.0	0.0	1.0

HMM Inference and Learning

- Learning
  - Maximum likelihood: transitions q and emissions e

$$p(x_1 \dots x_n, y_1 \dots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

- Inference
  - Viterbi

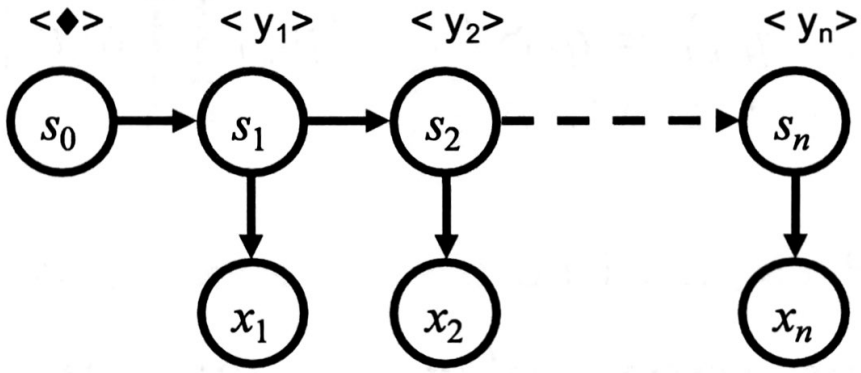
$$y^* = \arg \max_{y_1 \dots y_n} p(x_1 \dots x_n, y_1 \dots y_n)$$

  - Forward backward

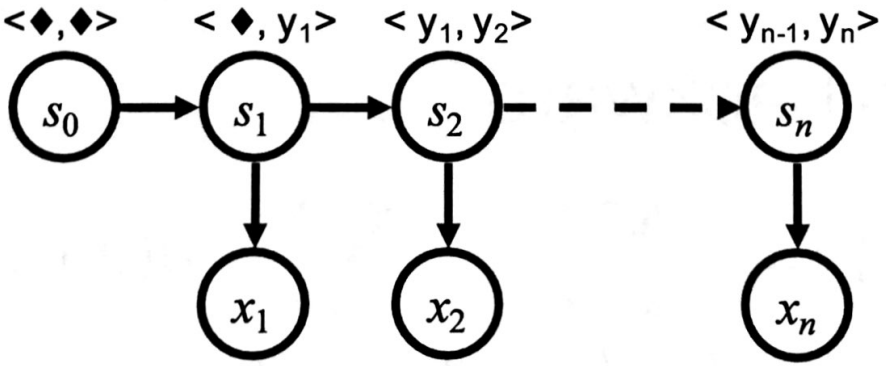
$$p(x_1 \dots x_n, y_i) = \sum_{y_1 \dots y_{i-1}} \sum_{y_{i+1} \dots y_n} p(x_1 \dots x_n, y_1 \dots y_n)$$

What about n-gram Taggers?

- States encode what is relevant about the past
- Transitions $P(s_i \mid s_{i-1})$ encode well-formed tag sequences
  - In a bigram tagger, states = tags



  - In a trigram tagger, states = tag pairs



MEMM Taggers

One step up: also condition on previous tags:

$$p(y_1 \ldots y_m | x_1 \ldots x_m) = \prod_{i=1}^{m} p(y_i | y_1 \ldots y_{i-1}, x_1 \ldots x_m)$$

$$= \prod_{i=1}^{m} p(y_i | y_{i-1}, x_1 \ldots x_m)$$

- Training:
  - Train $p(y_i | y_{i-1}, x_1 \ldots x_m)$ as a discrete log-linear (MaxEnt) model
- Scoring:

$$p(y_i | y_{i-1}, x_1 \ldots x_m) = \frac{e^{w \cdot \phi(x_1 \ldots x_m, i, y_{i-1}, y_i)}}{\sum_{y'} e^{w \cdot \phi(x_1 \ldots x_m, i, y_{i-1}, y')}}$$

- This is referred to as an MEMM tagger [Ratnaparkhi 96]

HMM vs. MEMM

- HMM models joint distribution:

$$p(x_1 \dots x_n, y_1 \dots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

- MEMM models conditioned distribution:

$$p(y_1 \dots y_m|x_1 \dots x_m) = \prod_{i=1}^{m} p(y_i|y_1 \dots y_{i-1}, x_1 \dots x_m)$$

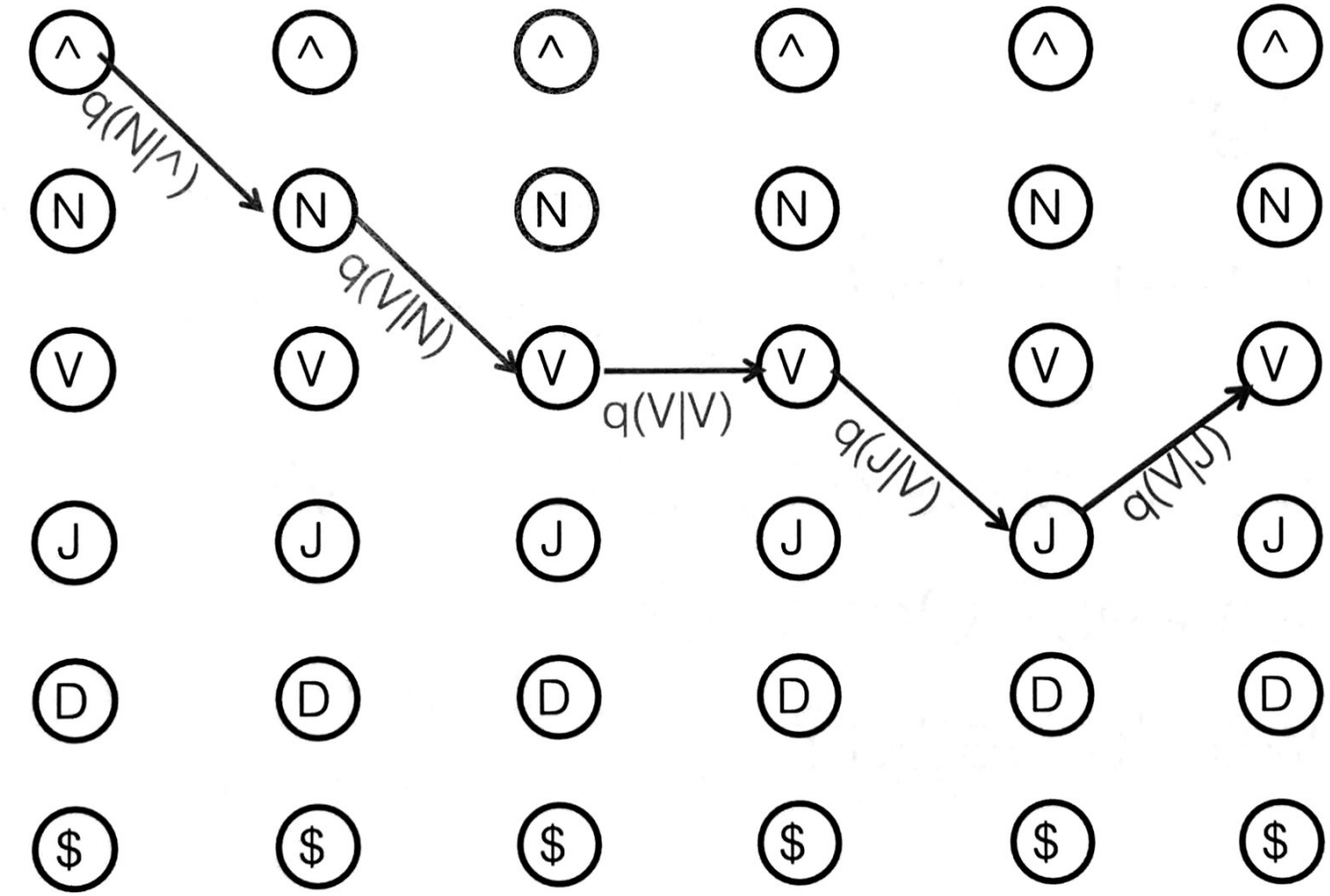
Decoding MEMM Taggers

- Scoring:

$$p(y_i|y_{i-1}, x_1 \dots x_m) = \frac{e^{w \cdot \phi(x_1 \dots x_m, i, y_{i-1}, y_i)}}{\sum_{y'} e^{w \cdot \phi(x_1 \dots x_m, i, y_{i-1}, y')}}$$

- Beam search is effective – why?
- Guarantees? Optimal?
- Can we do better?

The MEMM State Lattice / Trellis



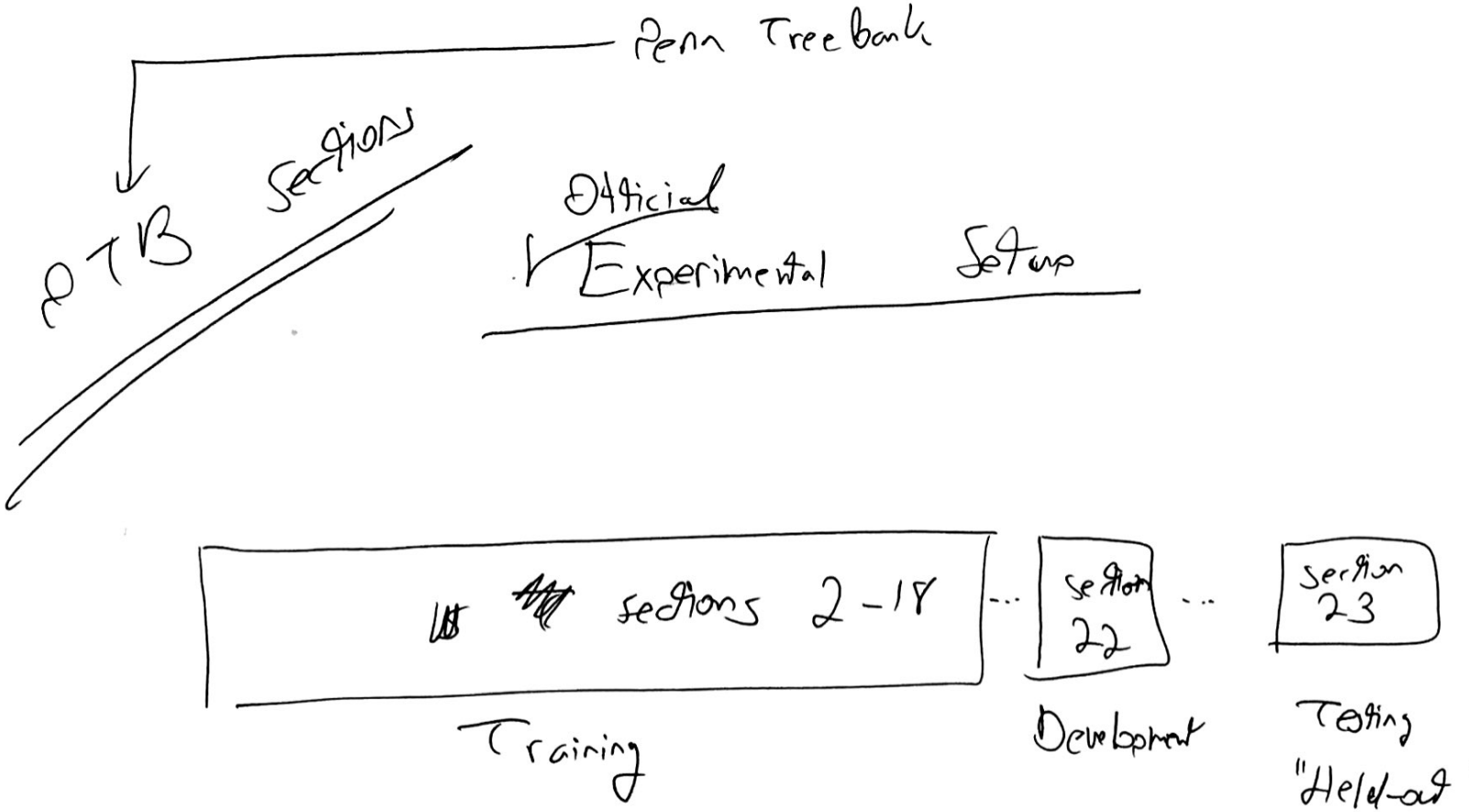
Decoding MEMM Taggers

- Decoding MaxEnt taggers:
  - Just like decoding HMMs
  - Viterbi, beam search
- Viterbi algorithm (HMMs):
  - Define $\pi(i,y_i)$ to be the max score of a sequence of length i ending in tag y_i

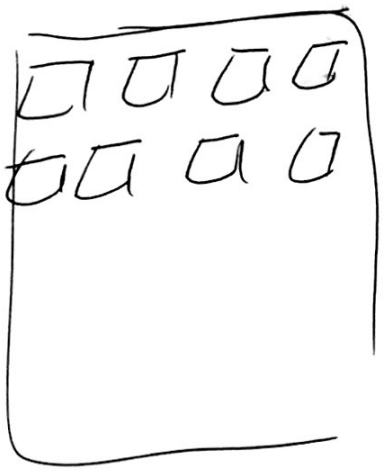
$$\pi(i, y_i) = \max_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\pi(i-1, y_{i-1})$$

- Viterbi algorithm (MaxEnt):
  - Can use same algorithm for MEMMs, just need to redefine $\pi(i,y_i)$!

$$\pi(i, y_i) = \max_{y_{i-1}} p(y_i|y_{i-1}, x_1 \dots x_m)\pi(i-1, y_{i-1})$$



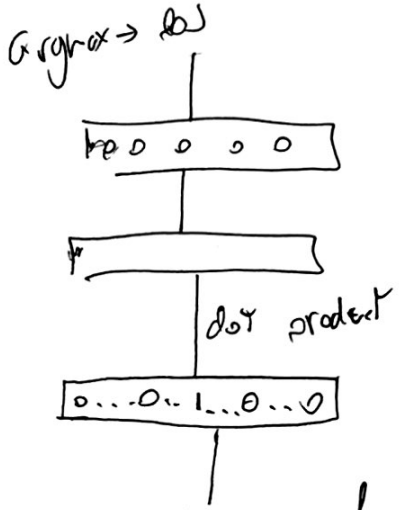
All other sections can be used for development too.

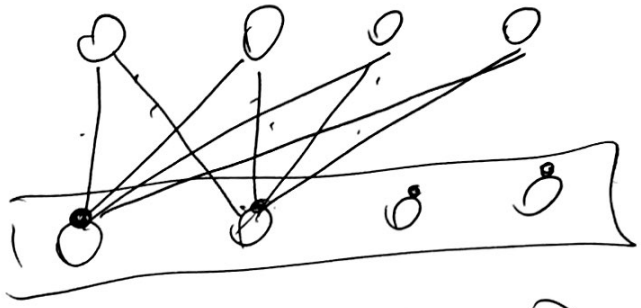


1

~~Primitive~~ Simple Feed Forward Architecture

Fully connected NN

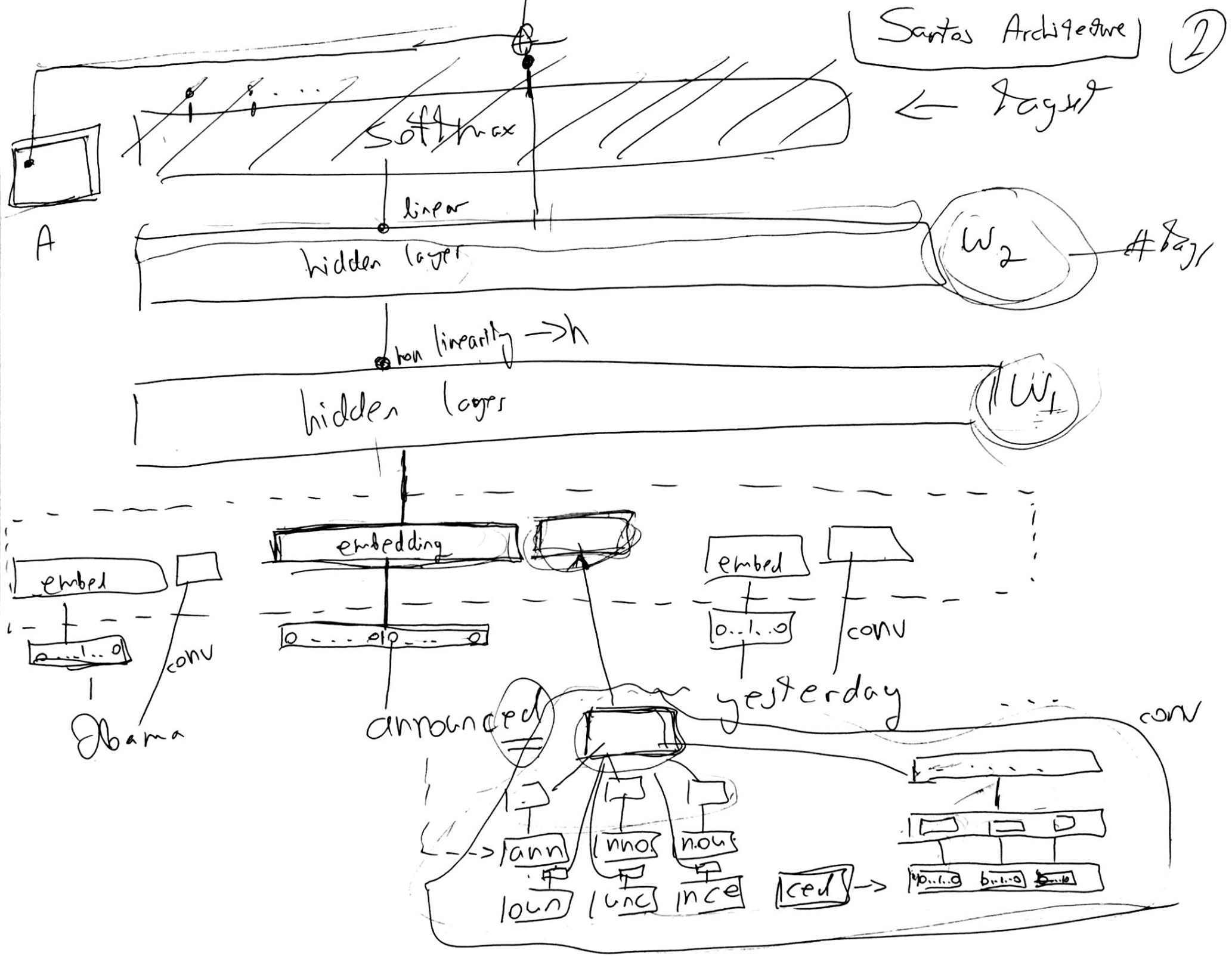




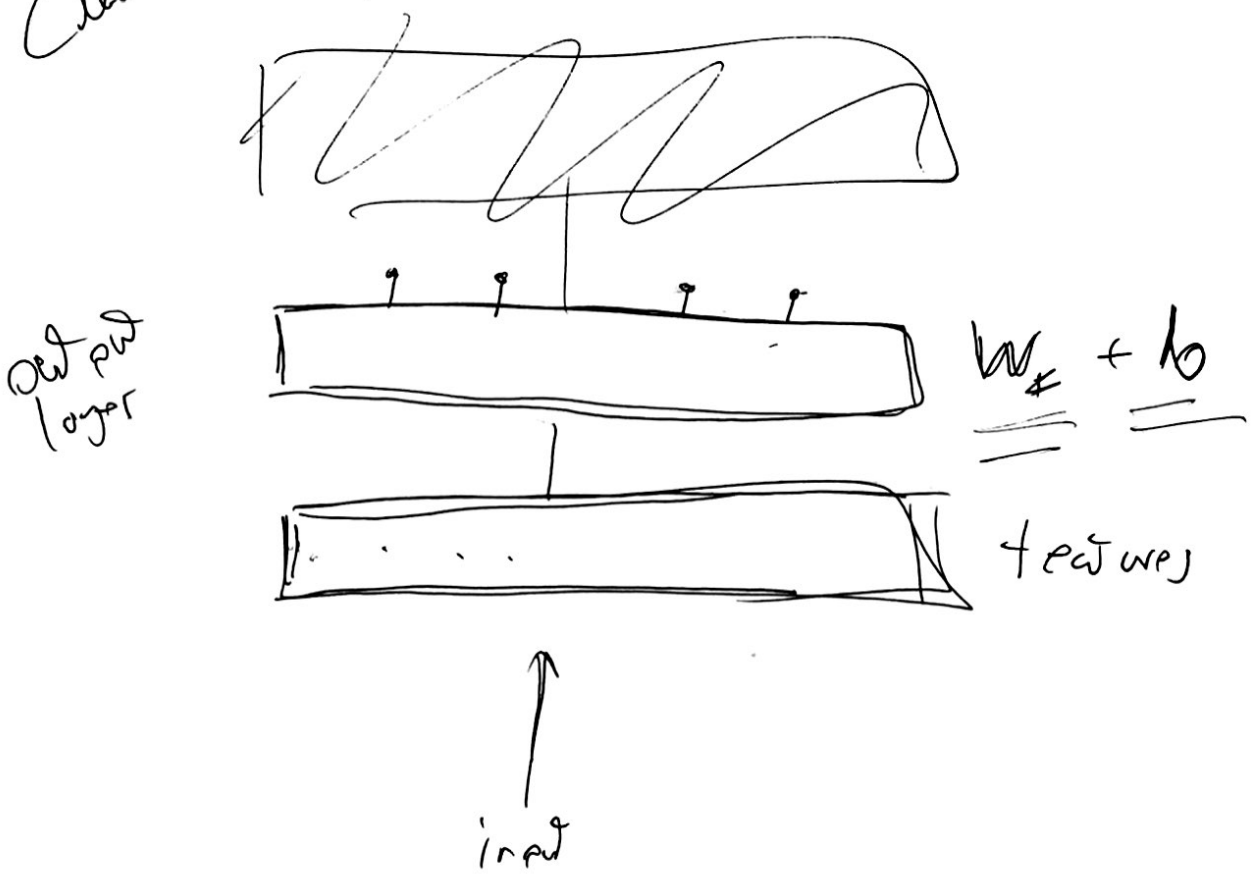
(activations) (edge weights)

Obama announced yesterday that ...

Santos Architecture (2)



Simple Classification



4

Recurrent Architecture