CS 533: Natural Language Processing

Sequence Labeling (Tagging)

Karl Stratos

Rutgers University

Decoding: the Search Problem

- Goal: find highest scoring output

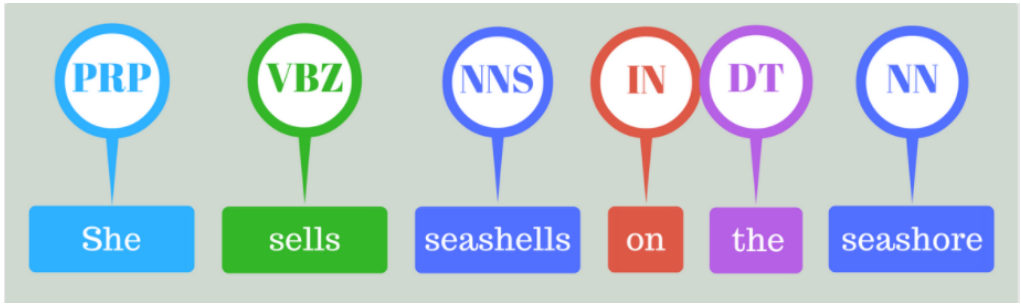
$$\mathbf{y}^* = \arg\max_{\mathbf{y} \in \mathcal{S}} \text{score}(\mathbf{y})$$

- Cannot eneumerate all elements if the search space $\mathcal{S}$ is large

- If $\mathcal{S}$ is structured, by making conditional independence assumptions the search problem can be solved **exactly** and **efficiently**
  - Today: $\mathcal{S}$ = sequences ("tagging")

Tagging is Extremely Practical

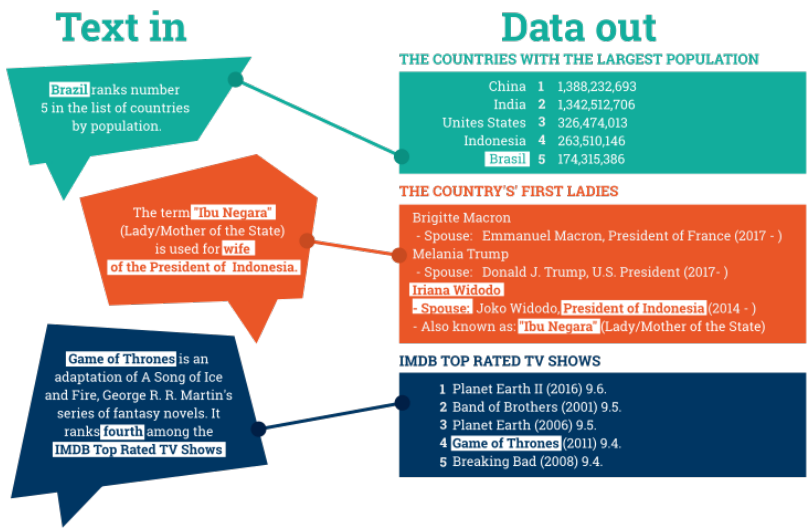
part-of-speech tagging



named entity recognition

Figure 1: An example of NER application on an example text

information extraction



Part-Of-Speech (POS) Tagging

Task. Given a sentence, output a sequence of POS tags.

Ambiguity. A word can have many possible POS tags.

```
the/DT man/NN saw/VBD the/DT cut/NN
the/DT saw/NN cut/VBD the/DT man/NN
```

Evaluation. Per-position accuracy (can consider others, like sentence-level accuracy)

Penn Treebank Tagset (for English)

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coordinating conjunction	*and, but, or*	PDT	predeterminer	*all, both*	VBP	verb non-3sg present	*eat*
CD	cardinal number	*one, two*	POS	possessive ending	*'s*	VBZ	verb 3sg pres	*eats*
DT	determiner	*a, the*	PRP	personal pronoun	*I, you, he*	WDT	wh-determ.	*which, that*
EX	existential 'there'	*there*	PRP$	possess. pronoun	*your, one's*	WP	wh-pronoun	*what, who*
FW	foreign word	*mea culpa*	RB	adverb	*quickly*	WP$	wh-possess.	*whose*
IN	preposition/ subordin-conj	*of, in, by*	RBR	comparative adverb	*faster*	WRB	wh-adverb	*how, where*
JJ	adjective	*yellow*	RBS	superlatv. adverb	*fastest*	$	dollar sign	*$*
JJR	comparative adj	*bigger*	RP	particle	*up, off*	#	pound sign	*#*
JJS	superlative adj	*wildest*	SYM	symbol	*+,%, &*	“	left quote	‘ or “
LS	list item marker	*1, 2, One*	TO	“to”	*to*	”	right quote	’ or ”
MD	modal	*can, should*	UH	interjection	*ah, oops*	(	left paren	[, (, {, <
NN	sing or mass noun	*llama*	VB	verb base form	*eat*	)	right paren	],), }, >
NNS	noun, plural	*llamas*	VBD	verb past tense	*ate*	,	comma	,
NNP	proper noun, sing.	*IBM*	VBG	verb gerund	*eating*	.	sent-end punc	. ! ?
NNPS	proper noun, plu.	*Carolinas*	VBN	verb past part.	*eaten*	:	sent-mid punc	: ; ... – -

Figure 8.1 Penn Treebank part-of-speech tags (including punctuation).

[45 tags]

(Marcus et al., 1993)

Other definitions: universal tagset (12 tags, language agnostic)
http://www.lrec-conf.org/proceedings/lrec2012/pdf/274_Paper.pdf

Problem: Disambiguation

- Use a statistical approach to disambiguate POS tags.

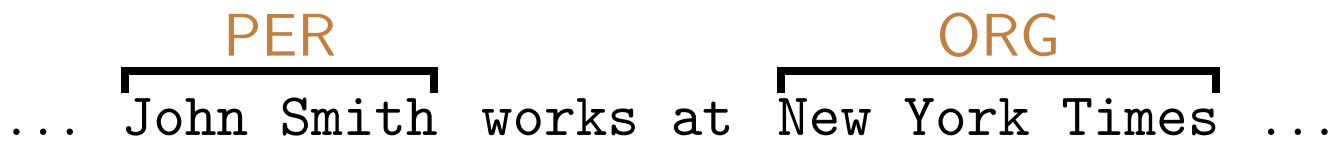
Types:		WSJ	Brown
Unambiguous	(1 tag)	44,432 (**86%**)	45,799 (**85%**)
Ambiguous	(2+ tags)	7,025 (**14%**)	8,050 (**15%**)
Tokens:			
Unambiguous	(1 tag)	577,421 (**45%**)	384,349 (**33%**)
Ambiguous	(2+ tags)	711,780 (**55%**)	786,646 (**67%**)

Figure 8.2 Tag ambiguity for word types in Brown and WSJ, using Treebank-3 (45-tag) tagging. Punctuation were treated as words, and words were kept in their original case.

- http://nlp.stanford.edu:8080/parser/index.jsp
  - Try "`the old man the boat`"
- POS tagging is still a relatively easy problem
  - Many words unambiguous or barely ambiguous (e.g., "the")
  - Majority class assignment gets $> 92\%$ on Penn Treebank
  - State-of-the-art $> 97\%$
- However, not solved
  - https://nlp.stanford.edu/pubs/CICLing2011-manning-tagging.pdf

Named Entity Recognition (NER)

Task. Given a sentence, identify and label all spans that are "entities"



Reduction to tagging. "Linearize" labeled spans into a label sequence using "BIO" scheme

John/B-PER Smith/I-PER works/O at/O New/B-ORG York/I-ORG Times/I-ORG

Number of labels. number of entity types $\times 3 + 1$

CoNLL 2003 Dataset

```
West    B-MISC
Indian  I-MISC
all-rounder     O
Phil    B-PER
Simmons I-PER
took    O
four    O
for     O
38      O
on      O
Friday  O
as      O
Leicestershire  B-ORG
beat    O
Somerset        B-ORG
by      O
```

4 entity types (PER, ORG, LOC, MISC)

NER Evaluation

- Per-entity F1 score (harmonic mean of precision and recall)

$$F_1(e) = \frac{2p(e)r(e)}{p(e) + r(e)}$$

$$p(e) = \frac{tp(e)}{tp(e) + fp(e)} \times 100 \qquad r(e) = \frac{tp(e)}{tp(e) + fn(e)} \times 100$$

- Global F1 score

$$F_1 = \frac{2pr}{p + r}$$

$$p = \frac{tp}{tp + fp} \times 100 \qquad r = \frac{tp}{tp + fn} \times 100$$

- State-of-the-art: over 93 F1 on CoNLL 2003!

  https://paperswithcode.com/sota/named-entity-recognition-ner-on-conll-2003

Agenda

1. HMM: A Probabilistic Generative Model of Sequence
2. CRF: A Probabilistic Discriminative Model of Sequence

Sequence Labeling with a Probabilistic Model

Vocabulary $\mathcal{X}$, set of POS tags $\mathcal{Y}$

$$\mathcal{X} = \{\texttt{prim}, \texttt{that}, \texttt{Arya}, \texttt{fastidiously}, \texttt{1988}, \ldots\}$$
$$\mathcal{Y} = \{\texttt{DT}, \texttt{NN}, \texttt{VBD}, \texttt{JJ}, \ldots\}$$

Want to define a **joint** distribution $p(x_1 \ldots x_n,\ y_1 \ldots y_n)$ over

1. Any sentence $x_1 \ldots x_n \in \mathcal{X}^n$
2. A corresponding sequence of POS tags $y_1 \ldots y_n \in \mathcal{Y}^n$

Why? Then we can infer for any given $x_1 \ldots x_n$

$$\begin{aligned} y_1^* \ldots y_n^* &= \arg\max_{y_1 \ldots y_n \in \mathcal{Y}^n} p(y_1 \ldots y_n | x_1 \ldots x_n) \\ &= \arg\max_{y_1 \ldots y_n \in \mathcal{Y}^n} p(x_1 \ldots x_n,\ y_1 \ldots y_n) \end{aligned}$$

A Left-to-Right Generative Process

By the chain rule, we may assume that

$$
\begin{aligned}
&p(x_1 \dots x_n,\ y_1 \dots y_n) \\
&\quad = p(y_1) \times p(x_1|y_1) \times p(y_2|x_1, y_1) \times p(x_2|x_1, y_1, y_2) \cdots \\
&\qquad \times p(y_n|\left\{x_i, y_i\right\}_{i=1}^{n-1}) \times p(x_n|\left\{x_i, y_i\right\}_{i=1}^{n-1}, y_n) \\
&\qquad \times p(\texttt{STOP}|\left\{x_i, y_i\right\}_{i=1}^{n})
\end{aligned}
$$

Design a tractable model by making **independence assumptions**.

- What kind of assumption is reasonable for POS tagging?

First-Order HMM Assumptions

1. At any position i, the word depends on the current tag only.

$$p(x_i | \{x_j, y_j\}_{j=1}^{i-1}, y_i) = p(x_i | y_i)$$

2. At any position i, the tag depends on the previous tag only.

$$p(y_i | \{x_j, y_j\}_{j=1}^{i-1}) = p(y_i | y_{i-1})$$

Model Parameters

- $|\mathcal{X}| \times |\mathcal{Y}|$ "emission" probabilities

$$o(x|y) = \text{probability of emitting word } x \text{ given tag } y$$

- $|\mathcal{Y}|^2 + 2\,|\mathcal{Y}|$ "transition" probabilities

$$t(y'|y) = \text{probability of transitioning from tag } y \text{ to } y'$$
$$t(y|*) = \text{probability of starting with tag } y$$
$$t(\texttt{STOP}|y) = \text{probability of ending with tag } y$$

Used to calculate ($y_0 = *$ and $y_{n+1} = \texttt{STOP}$ special symbols)

$$p(x_1 \dots x_n,\ y_1 \dots y_n) = \prod_{i=1}^{n+1} t(y_i|y_{i-1}) \times \prod_{i=1}^{n} o(x_i|y_i)$$

Labeled Data

- Consists of N annotated sentences $(x^{(1)}, y^{(1)}) \dots (x^{(N)}, y^{(N)})$ where $l_i = \left|x^{(i)}\right| = \left|y^{(i)}\right|$ and $y_0^{(i)} = *$, $y_{l_i+1}^{(i)} = \texttt{STOP}$.

- Define $\textbf{count}(y, y')$ for $y, y' \in \mathcal{Y} \cup \{*, \texttt{STOP}\}$:

$$\textbf{count}(y, y') = \sum_{i=1}^{N} \sum_{\substack{j=1: \\ y_{j-1}^{(i)}=y \\ y_j^{(i)}=y'}}^{l_i+1} 1$$

- Define $\textbf{count}(x, y)$ for $x \in \mathcal{X},\ y \in \mathcal{Y}$:

$$\textbf{count}(x, y) = \sum_{i=1}^{N} \sum_{\substack{j=1: \\ x_j^{(i)}=x \\ y_j^{(i)}=y}}^{l_i} 1$$

Parameter Estimation

- For all y, y' with $\textbf{count}(y, y') > 0$, set

$$t(y'|y) = \frac{\textbf{count}(y, y')}{\sum_{y' \in \mathcal{Y}} \textbf{count}(y, y')}$$

Otherwise $t(y'|y) = 0$.

- For all x, y with $\textbf{count}(x, y) > 0$, set

$$o(x|y) = \frac{\textbf{count}(x, y)}{\sum_{x \in \mathcal{X}} \textbf{count}(x, y)}$$

Otherwise $o(x|y) = 0$.

Justification

Claim. The solution of

$$o^*, t^* = \underset{\substack{o,\, t:\ o(x|y),\, t(y'|y) \geq 0 \\ \sum_{y'} t(y'|y) = \sum_x o(x|y) = 1}}{\arg\max} \sum_{i=1}^{N} \log p(x^{(i)}, y^{(i)})$$

where $p(x, y)$ is the distribution of an HMM is given by

$$o^*(x|y) = \frac{\textbf{count}(x, y)}{\sum_x \textbf{count}(x, y)} \qquad t^*(y'|y) = \frac{\textbf{count}(y, y')}{\sum_{y'} \textbf{count}(y, y')}$$

Setting

- We now assume that we have parameters $o(x|y)$ and $t(y'|y)$.
- They define the joint probability distribution

$$p(x_1 \dots x_n,\ y_1 \dots y_n) = \prod_{i=1}^{n+1} t(y_i|y_{i-1}) \times \prod_{i=1}^{n} o(x_i|y_i)$$

  over any words $x_1 \dots x_n \in \mathcal{X}^n$ and POS tags $y_1 \dots y_n \in \mathcal{Y}^n$.
- **Given a fixed sentence** $x_1 \dots x_n \in \mathcal{X}^n$, we often wish to perform two critical calculations (next slide).

Marginalization and Inference

1. What is the probability of $x_1 \dots x_n$ under the HMM?

$$\sum_{y_1 \dots y_n \in \mathcal{Y}^n} p(x_1 \dots x_n, \ y_1 \dots y_n)$$

2. What is the most probable $y_1 \dots y_n \in \mathcal{Y}^n$ under the HMM?

$$\arg\max_{y_1 \dots y_n \in \mathcal{Y}^n} p(x_1 \dots x_n, \ y_1 \dots y_n)$$

Number of Possible Tag Sequences

- **Exponential in the length of the sentence**
- Enumerating all $|\mathcal{Y}|^n$ candidates is clearly not practical.
- We will exploit the HMM assumptions to perform marginalization/inference **exactly** and with **polynomial complexity**.

Left-to-Right Incremental Marginalization

- **Idea.** No need to consider all $|\mathcal{Y}|^n$ candidates because of the left-to-right generative process and independence assumptions under the HMM

- **Forward algorithm.** For $i = 1 \ldots n$, for all $y \in \mathcal{Y}$,

$$\pi(i, y) := \sum_{y_1 \ldots y_i \in \mathcal{Y}^i :\, y_i = y} p(x_1 \ldots x_i, y_1 \ldots y_i)$$

  We will see that computing each $\pi(i, y)$ takes $O(|\mathcal{Y}|)$ time using **dynamic programming**.

- Total runtime?

Base Case ($i = 1$)

$$\pi(1, y) := \sum_{y_1 \in \mathcal{Y}:\, y_1 = y} p(x_1, y_1)$$
$$= t(y|*) \times o(x_1|y)$$

Main Body ($i > 1$)

$$\pi(i, y') := \sum_{y_1 \ldots y_i :\, y_i = y'} p(x_1 \ldots x_i, y_1 \ldots y_i)$$

Main Body ($i > 1$)

$$\pi(i, y') := \sum_{y_1 \ldots y_i :\, y_i = y'} p(x_1 \ldots x_i, y_1 \ldots y_i)$$

$$= \sum_{y_1} \cdots \sum_{y_{i-1}} p(x_1 \ldots x_i, y_1 \ldots y_{i-1}\ y')$$

Main Body ($i > 1$)

$$\pi(i, y') := \sum_{y_1 \ldots y_i :\, y_i = y'} p(x_1 \ldots x_i, y_1 \ldots y_i)$$

$$= \sum_{y_1} \cdots \sum_{y_{i-1}} p(x_1 \ldots x_i, y_1 \ldots y_{i-1}\ y')$$

$$= \sum_{y_1} \cdots \sum_{y_{i-1}} p(x_1 \ldots x_{i-1}, y_1 \ldots y_{i-1}) \times t(y' | y_{i-1}) \times o(x_i | y')$$

Main Body ($i > 1$)

$$\pi(i, y') := \sum_{y_1 \ldots y_i :\, y_i = y'} p(x_1 \ldots x_i, y_1 \ldots y_i)$$

$$= \sum_{y_1} \cdots \sum_{y_{i-1}} p(x_1 \ldots x_i, y_1 \ldots y_{i-1}\ y')$$

$$= \sum_{y_1} \cdots \sum_{y_{i-1}} p(x_1 \ldots x_{i-1}, y_1 \ldots y_{i-1}) \times t(y'|y_{i-1}) \times o(x_i|y')$$

$$= \sum_{y} \sum_{y_1 \ldots y_{i-2}} p(x_1 \ldots x_{i-1}, y_1 \ldots y_{i-2}\ y) \times t(y'|y) \times o(x_i|y')$$

Main Body ($i > 1$)

$$\pi(i, y') := \sum_{y_1 \ldots y_i :\, y_i = y'} p(x_1 \ldots x_i, y_1 \ldots y_i)$$

$$= \sum_{y_1} \cdots \sum_{y_{i-1}} p(x_1 \ldots x_i, y_1 \ldots y_{i-1}\ y')$$

$$= \sum_{y_1} \cdots \sum_{y_{i-1}} p(x_1 \ldots x_{i-1}, y_1 \ldots y_{i-1}) \times t(y'|y_{i-1}) \times o(x_i|y')$$

$$= \sum_{y} \sum_{y_1 \ldots y_{i-2}} p(x_1 \ldots x_{i-1}, y_1 \ldots y_{i-2}\ y) \times t(y'|y) \times o(x_i|y')$$

$$= \sum_{y} \pi(i-1, y) \times t(y'|y) \times o(x_i|y')$$

Final Marginalization

Obtain the probability of $x_1 \dots x_n$ under the HMM by

$$\sum_{y_1 \dots y_n} p(x_1 \dots x_n,\ y_1 \dots y_n) = \sum_{y \in \mathcal{Y}} \pi(n, y)$$

Left-to-Right Incremental Maximization

- **Same Idea.** Use the properties of the HMM.

- **Viterbi algorithm.** For $i = 1 \ldots n$, for all $y \in \mathcal{Y}$,

$$\pi(i, y) := \max_{y_1 \ldots y_i \in \mathcal{Y}^i:\, y_i = y} \; p(x_1 \ldots x_i, y_1 \ldots y_i)$$

- The *only* difference from the forward alg: "$\sum$" $\mapsto$ "max"

$$\pi(1, y) = t(y|*) \times o(x_1|y)$$
$$\pi(i, y') = \max_{y \in \mathcal{Y}} \; \pi(i-1, y) \times t(y'|y) \times o(x_i|y')$$

- But how do we extract the actual **tag sequence**?

$$y_1^* \ldots y_n^* = \arg\max_{y_1 \ldots y_n \in \mathcal{Y}^n} \; p(x_1 \ldots x_n,\; y_1 \ldots y_n)$$

Backtracking

- Keep an *additional* chart to record the **path**:

$$\beta(i, y') = \arg\max_{y \in \mathcal{Y}} \ \pi(i-1, y) \times t(y'|y) \times o(x_i|y')$$

for $i = 2 \ldots n$.

- After running Viterbi, we can "backtrack"

$$y_n^* = \arg\max_{y \in \mathcal{Y}} \ \pi(n, y)$$
$$y_{n-1}^* = \beta(n, y_n^*)$$
$$\vdots$$
$$y_1^* = \beta(2, y_2^*)$$

and return $y_1^* \ldots y_n^*$.

Log Space

- For numerical stability, always operate in **log space**.
- For Viterbi, it's a simple change:

$$\pi(1, y) = \log t(y|*) + \log o(x_1|y)$$
$$\pi(i, y') = \max_{y \in \mathcal{Y}} \ \pi(i-1, y) + \log t(y'|y) + \log o(x_i|y')$$

- For the forward algorithm, we need a helper function:

$$\text{logsumexp}(\log(c_1) \dots \log(c_n))$$

returns $\log(c_1 + \cdots + c_n)$ **without exponentiating** $\log(c_i)$!

Log Space: Forward Algorithm

- Original:

$$\pi(1, y) = t(y|*) \times o(x_1|y)$$
$$\pi(i, y') = \sum_{y \in \mathcal{Y}} \pi(i-1, y) \times t(y'|y) \times o(x_i|y')$$

- Log space:

$$\pi(1, y) = \log t(y|*) + \log o(x_1|y)$$
$$\pi(i, y') = \underset{y \in \mathcal{Y}}{\text{logsumexp}} \ \pi(i-1, y) + \log t(y'|y) + \log o(x_i|y')$$

Trick to Sum Logs

Input: $\log a \geq \log b$
Output: $\log(a + b)$

- If $\log a < -\infty$: return $-\infty$.
- If $\log b - \log a < -20$: return $\log a$.
- If $\log b - \log a \geq -20$: return

$$\log a + \log(1 + \exp(\log b - \log a))$$

Justification of the Trick

$$\log(a+b) = \log\left(a\left(1+\frac{b}{a}\right)\right)$$
$$= \log(a) + \log(1+\exp(\log b - \log a))$$

- Even if $\exp(\log a)$ and $\exp(\log b)$ underflow to zero, $\exp(\log b - \log a)$ does not.

$$\log a = -99999$$
$$\log b = -100000$$
$$\log b - \log a = -1$$

Remember LogSumExp!

$$\text{logsumexp}(a_1 \dots a_m) = \log \sum_{i=1}^{m} \exp(a_i)$$

- Your go-to function pretty much whenever you implement a probabilistic model for numerical stability
- Standard function in many software libraries (e.g., PyTorch)

```
>>> import torch
>>> a = torch.randn(3)
>>> a.exp().sum().log().item()
1.4297373294830322
>>> a.logsumexp(dim=0).item()
1.4297374486923218
```

Debugging

- How do you debug the forward/Viterbi algorithm?

- The (only) surest check:
  1. Generate a small synthetic HMM, say with $|\mathcal{X}| = 10, |\mathcal{Y}| = 5$.
  2. Generate a short random sentence, say length 7.
  3. **Brute-force**: enumerate all 5^7 possible sequences for exact marginalization and inference.
  4. Run your forward/Viterbi.
  5. Make sure 4 is precisely the same as 3.
  6. Repeat 2–5 many times.

Score Function Under an HMM

- Given a fixed input sequence $x = (x_1 \dots x_n)$, an HMM defines the "score" of a candidate sequence $y = (y_1 \dots y_n)$ as

$$\text{score}_x(y) = \prod_{i=1}^{n} \text{score}_x(y_i | y_1 \dots y_{i-1})$$

where each local score is **restricted** to only depend on the previous label y_{i-1} and current input x_i.

$$\text{score}_x(y_i | y_1 \dots y_{i-1}) := t(y_i | y_{i-1}) \times o(x_i | y_i)$$

- **With this restriction**, we can efficiently and exactly compute

$$\arg\max_{y_1 \dots y_n} \text{score}(y_1 \dots y_n) \qquad \text{(Viterbi)}$$

$$\sum_{y_1 \dots y_n} \text{score}(y_1 \dots y_n) \qquad \text{(forward)}$$

General Score Function

- Now suppose we have a local score that can depend arbitrarily on **all previous labels** $y_1 \dots y_{i-1}$:

$$\text{score}_x(y_i | y_1 \dots y_{i-1}) = f(x_1 \dots x_n, y_1 \dots y_{i-1})$$

- Without any Markov assumption, we can't hope to do inference/marginalization efficiently and exactly.

- But we can **approximate** it.

Beam Search

- A hack to approximate a **set** of top-K candidate sequences

$$\mathcal{B} \approx \underset{y_1 \dots y_n}{\text{K-argmax}} \ \ \text{score}_x(y_1 \dots y_n)$$

for **any** score function of the form

$$\text{score}_x(y) = \prod_{i=1}^{n} \text{score}_x(y_i | y_1 \dots y_{i-1})$$

Uses of the Beam Search

- The best sequence can be approximated as

$$\underset{(y_1 \ldots y_n) \in \mathcal{B}}{\arg\max} \quad \text{score}(y_1 \ldots y_n)$$

- The total score of all sequences can be approximated as

$$\sum_{(y_1 \ldots y_n) \in \mathcal{B}} \quad \text{score}(y_1 \ldots y_n)$$

Idea

- Maintain a "beam" $\mathcal{B}_i$ at each time step $i = 1 \dots n$ where

$$\mathcal{B}_i \approx \underset{y_1 \dots y_i}{\text{K-argmax}} \quad \textsf{score}_x(y_1 \dots y_i)$$

Beam Search Algorithm

- Base case ($i = 1$):

$$\mathcal{B}_1 = \underset{y \in \mathcal{Y}}{\text{K-argmax}} \quad \textsf{score}_x(y)$$

- Main body ($i > 1$):

$$\mathcal{B}_i = \underset{\substack{(y_1 \ldots y_{i-1}) \in \mathcal{B}_{i-1} \\ y_i \in \mathcal{Y}}}{\text{K-argmax}} \quad \textsf{score}_x(y_1 \ldots y_{i-1}) \times \textsf{score}_x(y_i | y_1 \ldots y_{i-1})$$

Leaky Priority Queue

- A "leaky" priority queue q with capacity K
- Accepts a stream of elements $[\texttt{thing}, \texttt{score}]$ but maintains only K elements with the highest scores seen so far.
- Both push and pop: $O(\log K)$ worst-case time complexity
- Assume a $O(K \log K)$ operation `dump`:

$$q.\texttt{dump}() = [q.\texttt{pop}() \text{ for } K \text{ times}]$$

- Exercise: try implementing it with a standard priority queue.

Implementation

- $q \leftarrow \texttt{leaky\_priority\_queue}(K)$
- $q.\texttt{push}([y_1, \text{score}_x(y_1)]) \qquad \forall y_1 \in \mathcal{Y}$
- For $i = 2 \ldots n$:
  - $\mathcal{B}_{i-1} \leftarrow q.\texttt{dump}()$
  - For $(y, s) \in \mathcal{B}_{i-1}$:

$$q.\texttt{push}([y.\text{append}(y_i), s \times \text{score}_x(y_i|y)]) \qquad \forall y_i \in \mathcal{Y}$$

- Return $q.\texttt{dump}()$.

Runtime complexity: $O(|\mathcal{Y}|\, K \log K n)$

Compare with first-order HMM's forward/Viterbi: $O(|\mathcal{Y}|^2\, n)$

Generative vs Discriminative

	Generative	Discriminative
Model	$p(\mathbf{x}, \mathbf{y})$	$p(\mathbf{y}\|\mathbf{x})$
Example	HMM	CRF, structured SVMs

- If all we want is just to infer $\mathbf{y}$ for given $\mathbf{x}$, we many not want/need to model $\mathbf{x}$

Conditional Random Fields (CRFs): Motivation

- Observations: an entire token sequence $\mathbf{x} = (x_1 \dots x_n) \in \mathcal{X}^n$
- Latents: an entire label sequence $\mathbf{y} = (y_1 \dots y_n) \in \mathcal{Y}^n$ for $\mathbf{x}$
  - Let $y_0 = *$ denote a special start symbol
- Goal: score an entire observation sequence and an entire label sequence with $s(\mathbf{x}, \mathbf{y}) \in \mathbb{R}$
- Questions:
  1. **Learning**: how can we learn such a scoring function?
  2. **Decoding**: how can we infer best label sequence for given $\mathbf{x}$?

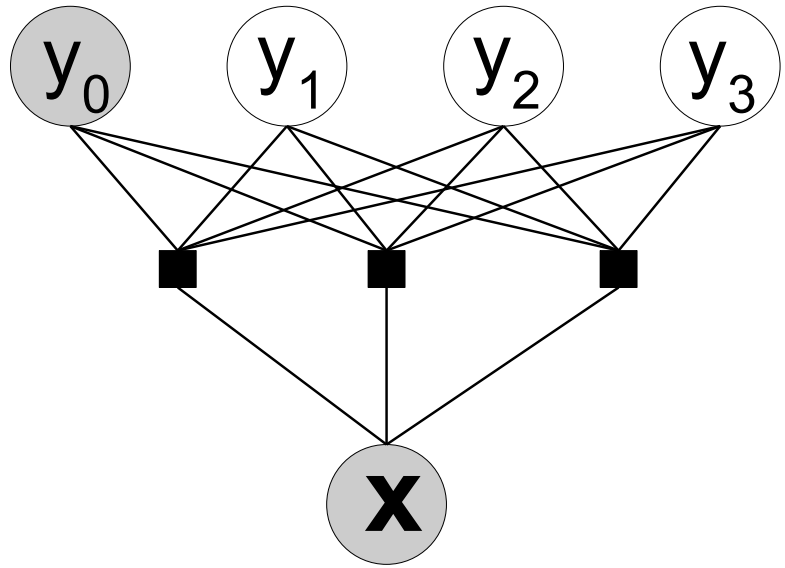
$$\arg\max_{\mathbf{y} \in \mathcal{Y}^n} \; s(\mathbf{x}, \mathbf{y})$$

- CRFs provide a framework that answers both questions.
  - General and applicable to other structured prediction tasks beyond sequence labeling

CRF: MRF (i.e., UGM) + Observed Nodes

https://en.wikipedia.org/wiki/Markov_random_field

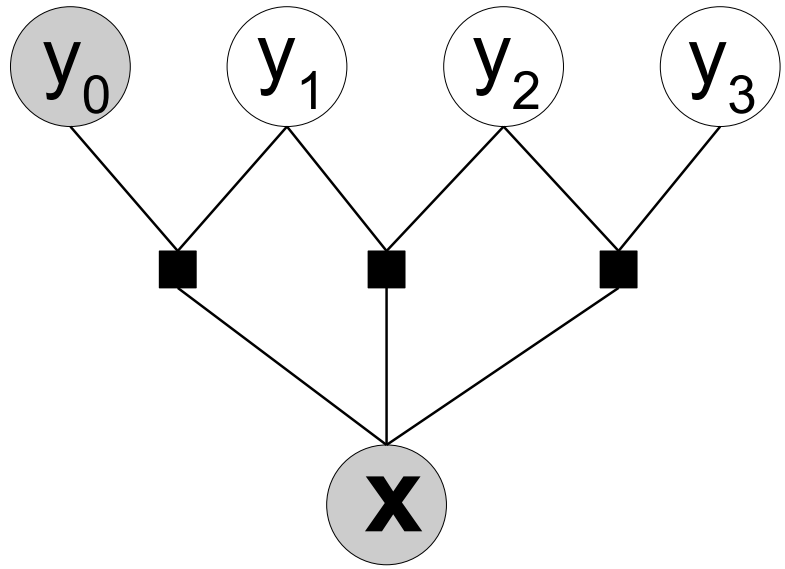
$$p(\mathbf{y}|\mathbf{x})$$



- Each label $y_i \in \mathcal{Y}$ depends on all other labels (and $\mathbf{x}$)
- All $\mathcal{Y}^n$ configurations must be considered for marginalization or inference

First-Order CRF: Markov Assumption

$$p(\mathbf{y}|\mathbf{x})$$



- Each label $y_i \in \mathcal{Y}$ depends only on $y_{i-1} \in \mathcal{Y}$ (and $\mathbf{x}$)
- We will see how marginalization/inference is now possible in $O(|\mathcal{Y}|^2 n)$ complexity

Conditional Distribution

- Hammersley-Clifford theorem

$$p(\mathbf{y}|\mathbf{x}) \propto \prod_{i=1}^{n} \underbrace{\exp(s(\mathbf{x}, y_{i-1}, y_i, i))}_{\text{nonnegative potential for } i\text{-th maximal clique}}$$

- Equivalently a giant softmax

$$p(\mathbf{y}|\mathbf{x}) = \frac{\exp(s(\mathbf{x}, \mathbf{y}))}{\sum_{\mathbf{y}' \in \mathcal{Y}^n} \exp(s(\mathbf{x}, \mathbf{y}'))}$$

with a factorized score function

$$s(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} s(\mathbf{x}, y_{i-1}, y_i, i)$$

Learning

- Your model defines a (differentiable) score function $s^\theta(\mathbf{x}, y_{i-1}, y_i, i)$ with some parameters θ
- MLE: single sample $(\mathbf{x}, \mathbf{y})$, maximize $\log p^\theta(\mathbf{y}|\mathbf{x})$ wrt. θ

$$\max_\theta \; s^\theta(\mathbf{x}, \mathbf{y}) - \underbrace{\log \left(\sum_{\mathbf{y}' \in \mathcal{Y}^n} \exp(s^\theta(\mathbf{x}, \mathbf{y}')) \right)}_{Z_\mathbf{x}}$$

- This is differentiable: as long as we can calculate this, we can use backpropagation and take gradient steps

Computing the Conditional Normalization Term $Z_{\mathbf{x}}$

Definition.

$$\pi(i, y) = \log \left(\sum_{y_1 \ldots y_{i-1} \in \mathcal{Y}^{i-1}} \exp(s^{\theta}(\mathbf{x}, y_1 \ldots y_{i-1} y)) \right)$$

We will calculate $\pi(i, y)$ for all $i = 1 \ldots n$ and $y \in \mathcal{Y}$. Then return

$$Z_{\mathbf{x}} = \log \left(\sum_{y \in \mathcal{Y}} \exp(\pi(n, y)) \right)$$

Why?

Forward Algorithm $O(|\mathcal{Y}|^2 n)$

Base case. For all $y \in \mathcal{Y}$: $\pi(1, y) = \log\left(\exp(s^\theta(\mathbf{x}, *, y, 1))\right)$

Body For all $i = 2 \ldots n$, for all $y \in \mathcal{Y}$:

$$\pi(i, y) = \log\left(\sum_{y_1 \cdots y_{i-1}} \exp(s^\theta(\mathbf{x}, y_1 \ldots y_{i-1} y))\right)$$

$$= \log\left(\sum_{y_1 \cdots y_{i-1}} \prod_{j=1}^{i-1} \exp(s^\theta(\mathbf{x}, y_{j-1}, y_j, j)) \exp(s^\theta(\mathbf{x}, y_{i-1}, y, i))\right)$$

$$= \log\left(\sum_{y'} \underbrace{\sum_{y_1 \cdots y_{i-2}} \exp(s^\theta(\mathbf{x}, y_1 \ldots y_{i-2} y'))}_{\exp(\pi(i-1, y'))} \exp(s^\theta(\mathbf{x}, y', y, i))\right)$$

$$= \log\left(\sum_{y'} \exp(\pi(i-1, y') + s^\theta(\mathbf{x}, y', y, i))\right)$$

Decoding

- Most likely sequence = highest scoring sequence

$$\arg\max_{\mathbf{y} \in \mathcal{Y}^n} \log p^\theta(\mathbf{y}|\mathbf{x}) = \arg\max_{\mathbf{y} \in \mathcal{Y}^n} s^\theta(\mathbf{x}, \mathbf{y})$$

- Viterbi algorithm $O(|\mathcal{Y}|^2 n)$

$$\pi(i, y) = \max_{y_1 \dots y_{i-1}} s^\theta(\mathbf{x}, y_1 \dots y_{i-1} y)$$

$$= \begin{cases} s^\theta(\mathbf{x}, *, y, 1) & \text{for } i = 1 \\ \max_{y'} \ \pi(i-1, y') + s^\theta(\mathbf{x}, y', y, i) & \text{for } i > 1 \end{cases}$$

- Retrieve optimal sequence by backtracking

Decoding: Backtracking

- Keep an additional chart to record the path: for $i = 2 \dots n$,

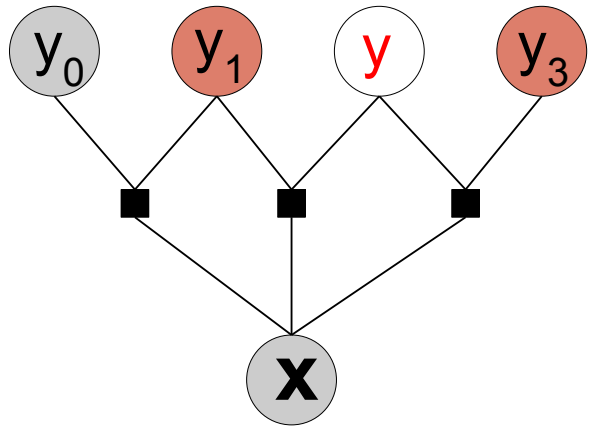
$$\beta(i, y) = \arg\max_{y'} \ \pi(i-1, y') + s^{\theta}(\mathbf{x}, y', y, i)$$

- After running Viterbi, return $\mathbf{y}^* = \arg\max_{\mathbf{y} \in \mathcal{Y}^n} \ s^{\theta}(\mathbf{x}, \mathbf{y})$ by

$$y_n^* = \arg\max_{y \in \mathcal{Y}} \ \pi(n, y)$$
$$y_{n-1}^* = \beta(n, y_n^*)$$
$$\vdots$$
$$y_1^* = \beta(2, y_2^*)$$

Alternative to Viterbi: Marginal Decoding

$$\mu(i, y) = \sum_{\mathbf{y}:\, y_i = y} p(\mathbf{y}|\mathbf{x})$$



- For each $i = 1 \ldots n$, return $y_i^* = \arg\max_y \mu(i, y)$
- Better for per-position accuracy (POS tagging), worse for structure modeling (NER)
- Need to run "backward" algorithm and combine with forward
  - Special case of belief propagation (for linear chains)
  - Assignment 4

Using CRF as an Inference Layer in Neural Networks

1. **Input transformation.** Get continuous representations $h_1 \dots h_n \in \mathbb{R}^d$ of input tokens $x_1 \dots x_n \in \mathcal{V}$
   - Each h_i can be a function of the entire $\mathbf{x}$
   - Popular choices: bidirectional LSTM, transformer

2. **CRF Layer.** $s^\theta(\mathbf{x}, y, y', i)$ parameterized by θ using $h_1 \dots h_n$. All parameters trained by minimizing the "CRF loss"

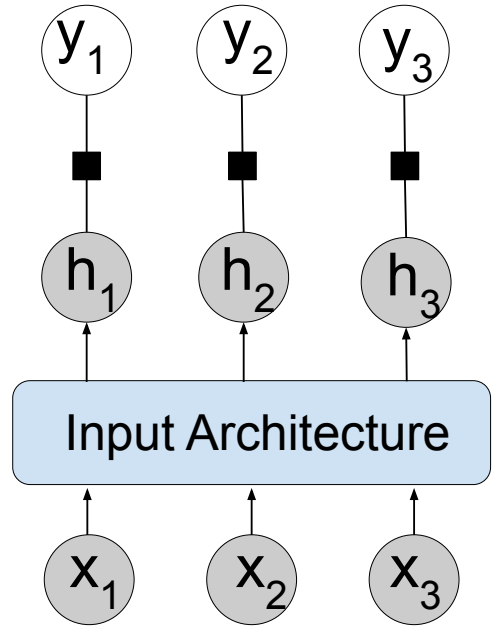
$$\underbrace{\log \left(\sum_{\mathbf{y}' \in \mathcal{Y}^n} \exp(s^\theta(\mathbf{x}, \mathbf{y}')) \right)}_{\text{Compute by the forward algorithm}} - s^\theta(\mathbf{x}, \mathbf{y})$$

3. **Decoding.** At test time, given $\mathbf{x}$ use Viterbi (or marginal decoding) to return an optimal label sequence $\mathbf{y}^*$

Greedy Decoding: 0-th Order CRF

Parameters $U \in \mathbb{R}^{|\mathcal{Y}| \times d}$

$$s^\theta(\mathbf{x}, y, y', i) = [Uh_i]_{y'}$$



Exercise: check that CRF reduces to (next slide is useful)

$$p(\mathbf{y}|\mathbf{x}) = \prod_i \text{softmax}_{y_i}(Uh_i)$$

$$y_i^* = \arg\max_{y \in \mathcal{Y}} [Uh_i]_y \qquad (\mathbf{y}^* = \arg\max_{\mathbf{y} \in \mathcal{Y}^n} s^\theta(\mathbf{x}, \mathbf{y}))$$

Sum-Product Interchangeability

Lemma. Let A be a set of m items. Let $n \in \mathbb{N}$ and assume $f_i : A \to \mathbb{R}$ for $i = 1 \ldots n$. Then

$$\underbrace{\sum_{a \in A^n} \prod_{i=1}^{n} f_i(a_i)}_{①} = \underbrace{\prod_{i=1}^{n} \sum_{\alpha \in A} f_i(\alpha)}_{②}$$

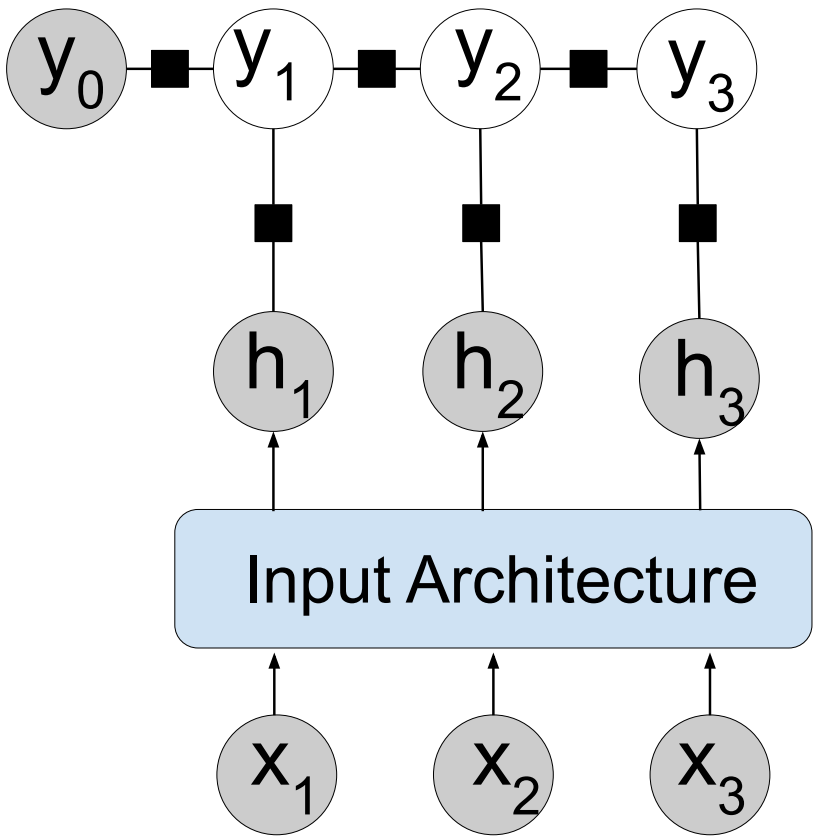
Proof. ② when expanded is just a different way of writing the sum of m^n products $\prod_i f_i(\alpha_i)$, $\alpha_i \in A$. E.g., $A = \{\alpha, \beta\}$, $n = 2$,

$$\begin{aligned} ② &= (f_1(\alpha) + f_1(\beta))(f_2(\alpha) + f_2(\beta)) \\ &= f_1(\alpha)f_2(\alpha) + f_1(\alpha)f_2(\beta) + f_1(\beta)f_2(\alpha) + f_1(\beta)f_2(\beta) = ① \end{aligned}$$

Typical First-Order CRF

Parameters $U \in \mathbb{R}^{|\mathcal{Y}| \times d}, T \in \mathbb{R}^{|\mathcal{Y}| \times (|\mathcal{Y}|+1)}$

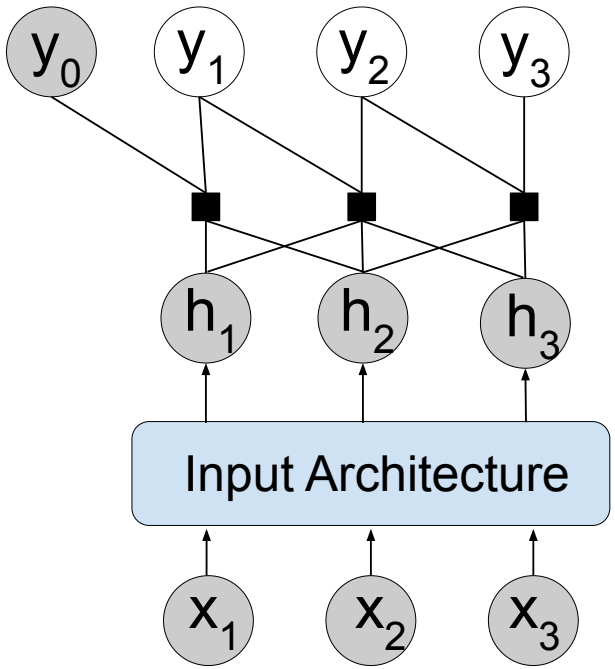
$$s^\theta(\mathbf{x}, y, y', i) = [Uh_i]_{y'} + [T]_{y',y}$$



Other Possibilities

Parameters $U \in \mathbb{R}^{|\mathcal{Y}| \times (3d+2d')}$, $E \in \mathbb{R}^{d' \times (|\mathcal{Y}|+1)}$

$$s^\theta(\mathbf{x}, y, y', i) = \tanh\left(U \begin{bmatrix} h_i \\ h_{i-1} \\ h_{i+1} \\ e_y \\ e_{y'} \end{bmatrix}\right)$$



Inference/marginalization does not change! As long as there is only pairwise dependence (y_{i-1}, y_i)