DSC291: Advanced Statistical Natural Language Processing

Classification
Sequence Tagging

Zhiting Hu

Lecture 10, April 28, 2022

UC San Diego
HALICIOĞLU DATA SCIENCE INSTITUTE

Outline

- Classification
  - Weakly supervised learning
- Sequence Tagging/Labeling

Classification

Classification with few labels

- Data augmentation
- Zero-/few-shot learning via prompting
- Weak supervision

The difficulty with supervised learning

- Annotated data is expensive and costs increase when...
  - *A task requires specialized expertise*

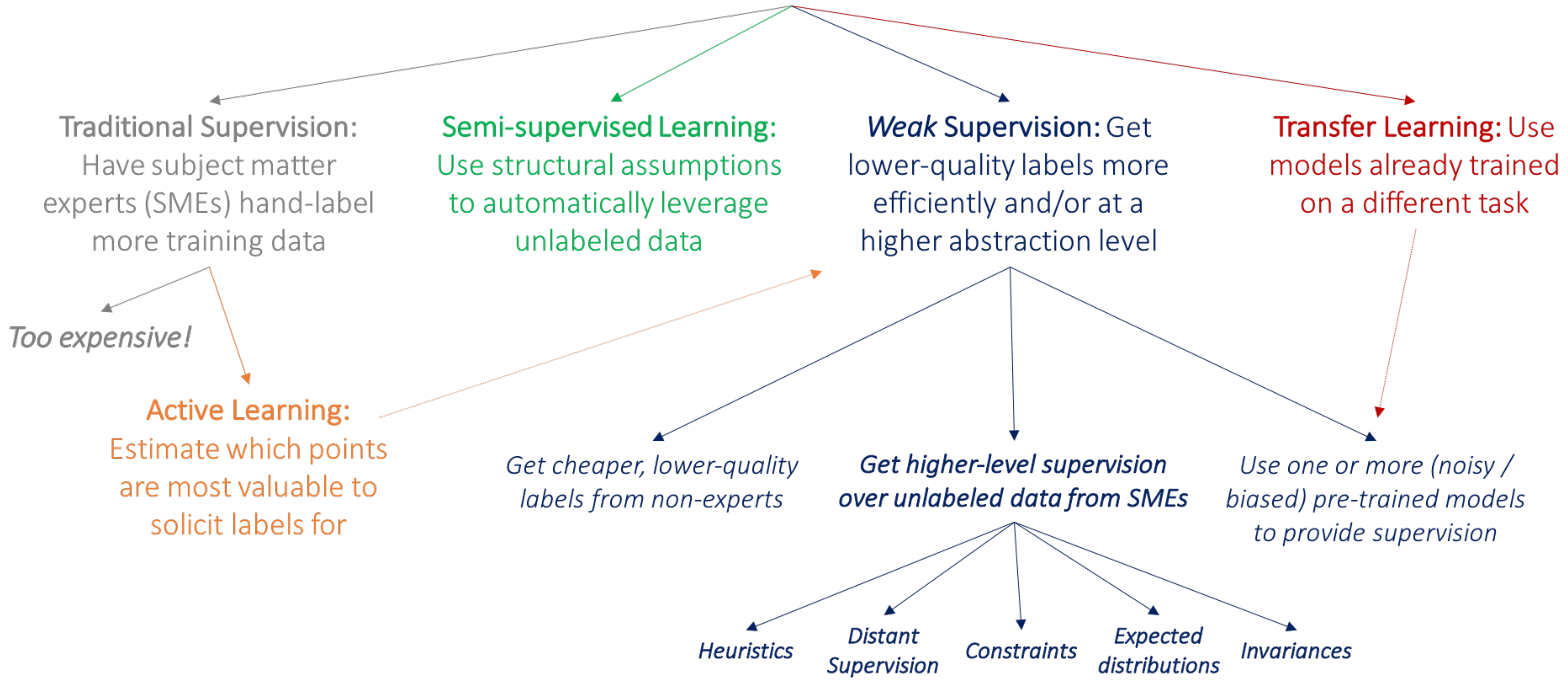
    E.g. "Only a trained linguist or a board certified radiologist can label my data"
  - *Labeling examples involves making multiple decisions*

    E.g. "Annotate this sentence with a parse tree"

    (instead of a single binary decision)

Credit: https://svivek.com/teaching/lectures/slides/weak-supervision/weak-supervision.pdf

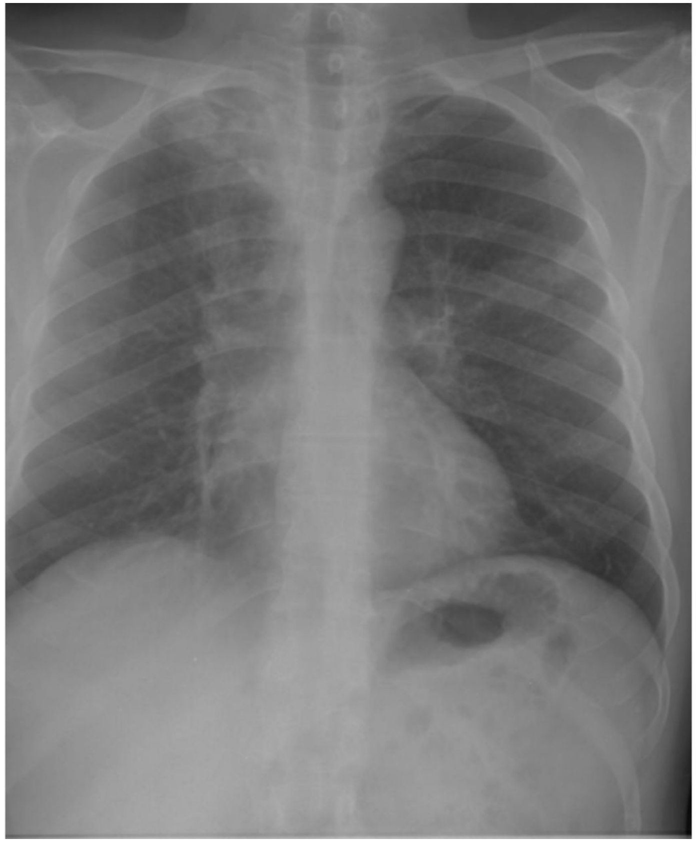
How to get more labeled training data?



Credit: https://dawn.cs.stanford.edu/2017/07/16/weak-supervision/

Example (I): labeling with heuristics

Task: Build a chest x-ray classifier
(normal/abnormal)

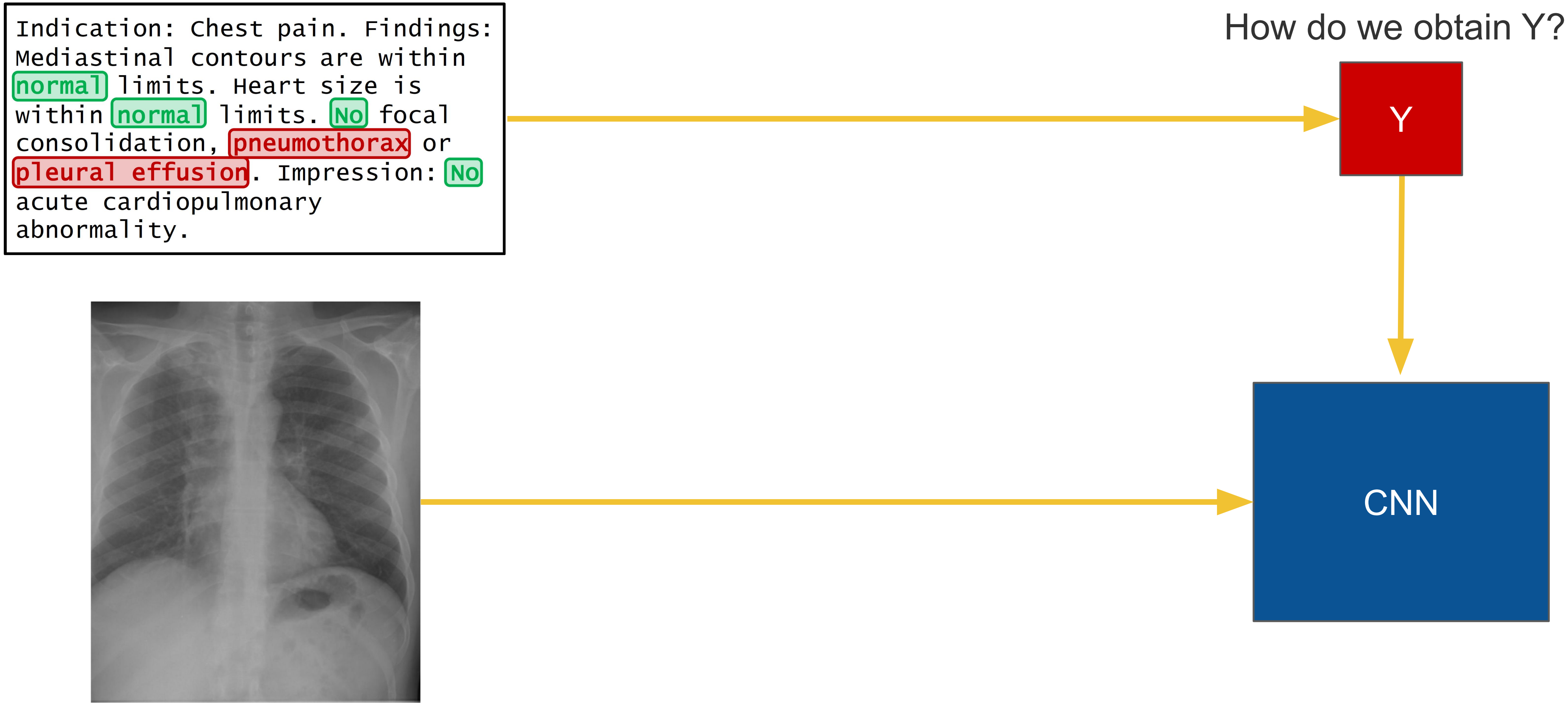


```
Indication: Chest pain. Findings:
Mediastinal contours are within
normal limits. Heart size is
within normal limits. No focal
consolidation, pneumothorax or
pleural effusion. Impression: No
acute cardiopulmonary
abnormality.
```

Can you use the accompanying medical report (text modality) to label the x-ray (image modality)?

[Credit: http://cs231n.stanford.edu/slides/2018/cs231n_2018_ds07.pdf]

Example (I): labeling with heuristics



[Credit: http://cs231n.stanford.edu/slides/2018/cs231n_2018_ds07.pdf]

Example (I): labeling with heuristics

```
Indication: Chest pain. Findings:
Mediastinal contours are within
normal limits. Heart size is
within normal limits. No focal
consolidation, pneumothorax or
pleural effusion. Impression: No
acute cardiopulmonary
abnormality.
```

Normal Report

```
def LF_pneumothorax(c):
  if re.search(r'pneumo.*', c.report.text):
      return "ABNORMAL"

def LF_pleural_effusion(c):
  if "pleural effusion" in c.report.text:
      return "ABNORMAL"

def LF_normal_report(c, thresh=2):
    if len(NORMAL_TERMS.intersection(c.
    report.words)) > thresh:
      return "NORMAL"
```

LFs
(labeling functions)

Source: Khandwala et. al 2017, Cross Modal Data Programming for Medical Images

[Credit: http://cs231n.stanford.edu/slides/2018/cs231n_2018_ds07.pdf]

Example (II): Labeling with knowledge bases

Task: relation extraction from text

- Hypothesis: If two entities belong to a certain relation, any sentence containing those two entities is likely to express that relation
- Key idea: use a *knowledge base* of relations to get lots of *noisy* training examples

Adapted from https://courses.cs.washington.edu/courses/cse517/13wi/slides/cse517wi13-RelationExtractionII.pdf

Example (II): Labeling with knowledge bases

Frequent Freebase relations

Relation name	Size	Example
/people/person/nationality	281,107	John Dugard, South Africa
/location/location/contains	253,223	Belgium, Nijlen
/people/person/profession	208,888	Dusa McDuff, Mathematician
/people/person/place_of_birth	105,799	Edwin Hubble, Marshfield
/dining/restaurant/cuisine	86,213	MacAyo's Mexican Kitchen, Mexican
/business/business_chain/location	66,529	Apple Inc., Apple Inc., South Park, NC
/biology/organism_classification_rank	42,806	Scorpaeniformes, Order
/film/film/genre	40,658	Where the Sidewalk Ends, Film noir
/film/film/language	31,103	Enter the Phoenix, Cantonese
/biology/organism_higher_classification	30,052	Calopteryx, Calopterygidae
/film/film/country	27,217	Turtle Diary, United States
/film/writer/film	23,856	Irving Shulman, Rebel Without a Cause
/film/director/film	23,539	Michael Mann, Collateral
/film/producer/film	22,079	Diane Eskenazi, Aladdin
/people/deceased_person/place_of_death	18,814	John W. Kern, Asheville
/music/artist/origin	18,619	The Octopus Project, Austin
/people/person/religion	17,582	Joseph Chartrand, Catholicism
/book/author/works_written	17,278	Paul Auster, Travels in the Scriptorium
/soccer/football_position/players	17,244	Midfielder, Chen Tao
/people/deceased_person/cause_of_death	16,709	Richard Daintree, Tuberculosis
/book/book/genre	16,431	Pony Soldiers, Science fiction
/film/film/music	14,070	Stavisky, Stephen Sondheim
/business/company/industry	13,805	ATS Medical, Health care

Adapted from https://courses.cs.washington.edu/courses/cse517/13wi/slides/cse517wi13-RelationExtractionII.pdf

Example (II): Labeling with knowledge bases

Corpus text

Bill Gates founded Microsoft in 1975.
Bill Gates, founder of Microsoft, …
Bill Gates attended Harvard from…
Google was founded by Larry Page …

Training data

Freebase

Founder: (Bill Gates, Microsoft)
Founder: (Larry Page, Google)
CollegeAttended: (Bill Gates, Harvard)

Credit: https://courses.cs.washington.edu/courses/cse517/13wi/slides/cse517wi13-RelationExtractionII.pdf

Example (II): Labeling with knowledge bases

Corpus text

Bill Gates founded Microsoft in 1975.
Bill Gates, founder of Microsoft, …
Bill Gates attended Harvard from…
Google was founded by Larry Page …

Freebase

Founder: (Bill Gates, Microsoft)
Founder: (Larry Page, Google)
CollegeAttended: (Bill Gates, Harvard)

Training data

(Bill Gates, Microsoft)
Label: Founder
Feature: X founded Y

Credit: https://courses.cs.washington.edu/courses/cse517/13wi/slides/cse517wi13-RelationExtractionII.pdf

Example (II): Labeling with knowledge bases

Corpus text

Bill Gates founded Microsoft in 1975.
Bill Gates, founder of Microsoft, …
Bill Gates attended Harvard from…
Google was founded by Larry Page …

Training data

(Bill Gates, Microsoft)
Label: Founder
Feature: X founded Y
Feature: X, founder of Y

Freebase

Founder: (Bill Gates, Microsoft)
Founder: (Larry Page, Google)
CollegeAttended: (Bill Gates, Harvard)

Credit: https://courses.cs.washington.edu/courses/cse517/13wi/slides/cse517wi13-RelationExtractionII.pdf

Example (II): Labeling with knowledge bases

Corpus text

Bill Gates founded Microsoft in 1975.
Bill Gates, founder of Microsoft, …
Bill Gates attended Harvard from…
Google was founded by Larry Page …

Freebase

Founder: (Bill Gates, Microsoft)
Founder: (Larry Page, Google)
CollegeAttended: (Bill Gates, Harvard)

Training data

(Bill Gates, Microsoft)
Label: Founder
Feature: X founded Y
Feature: X, founder of Y

(Bill Gates, Harvard)
Label: CollegeAttended
Feature: X attended Y

Credit: https://courses.cs.washington.edu/courses/cse517/13wi/slides/cse517wi13-RelationExtractionII.pdf

Example (II): Labeling with knowledge bases

Corpus text

Bill Gates founded Microsoft in 1975.
Bill Gates, founder of Microsoft, …
Bill Gates attended Harvard from…
Google was founded by Larry Page …

Freebase

Founder: (Bill Gates, Microsoft)
Founder: (Larry Page, Google)
CollegeAttended: (Bill Gates, Harvard)

Training data

(Bill Gates, Microsoft)
Label: Founder
Feature: X founded Y
Feature: X, founder of Y

(Bill Gates, Harvard)
Label: CollegeAttended
Feature: X attended Y

(Larry Page, Google)
Label: Founder
Feature: Y was founded by X

Credit: https://courses.cs.washington.edu/courses/cse517/13wi/slides/cse517wi13-RelationExtractionII.pdf

Example (II): Labeling with knowledge bases

Negative training data

Can't train a classifier with only positive data!
Need negative training data too!

Solution?
Sample 1% of unrelated pairs of entities.

Corpus text

Larry Page took a swipe at Microsoft...
...after Harvard invited Larry Page to...
Google is Bill Gates' worst fear ...

Training data

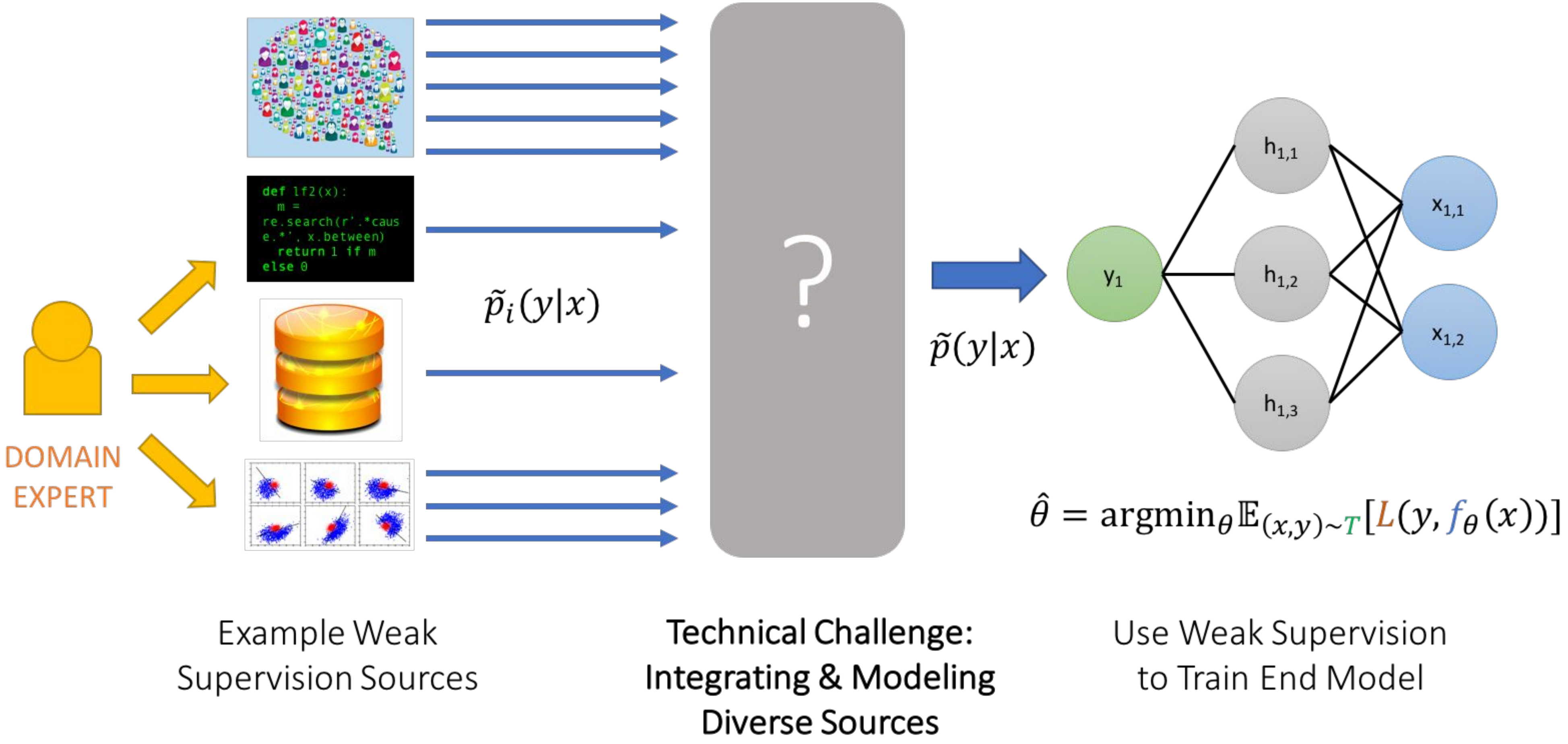
(Larry Page, Microsoft)
Label: NO_RELATION
Feature: X took a swipe at Y

(Larry Page, Harvard)
Label: NO_RELATION
Feature: Y invited X

(Bill Gates, Google)
Label: NO_RELATION
Feature: Y is X's worst fear

Credit: https://courses.cs.washington.edu/courses/cse517/13wi/slides/cse517wi13-RelationExtractionII.pdf

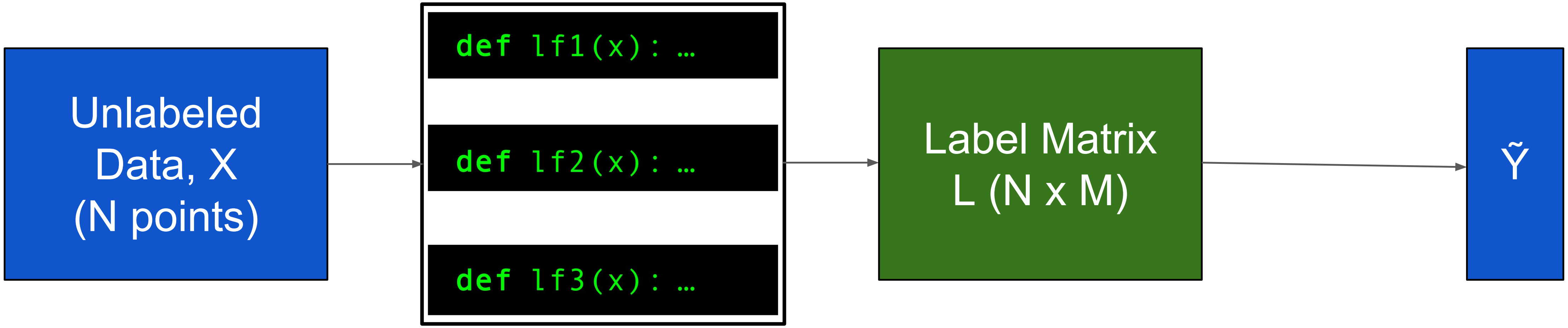
Integrating multiple noisy labels



Source: A. Ratner et. al https://dawn.cs.stanford.edu/2017/07/16/weak-supervision/

[Credit: http://cs231n.stanford.edu/slides/2018/cs231n_2018_ds07.pdf]

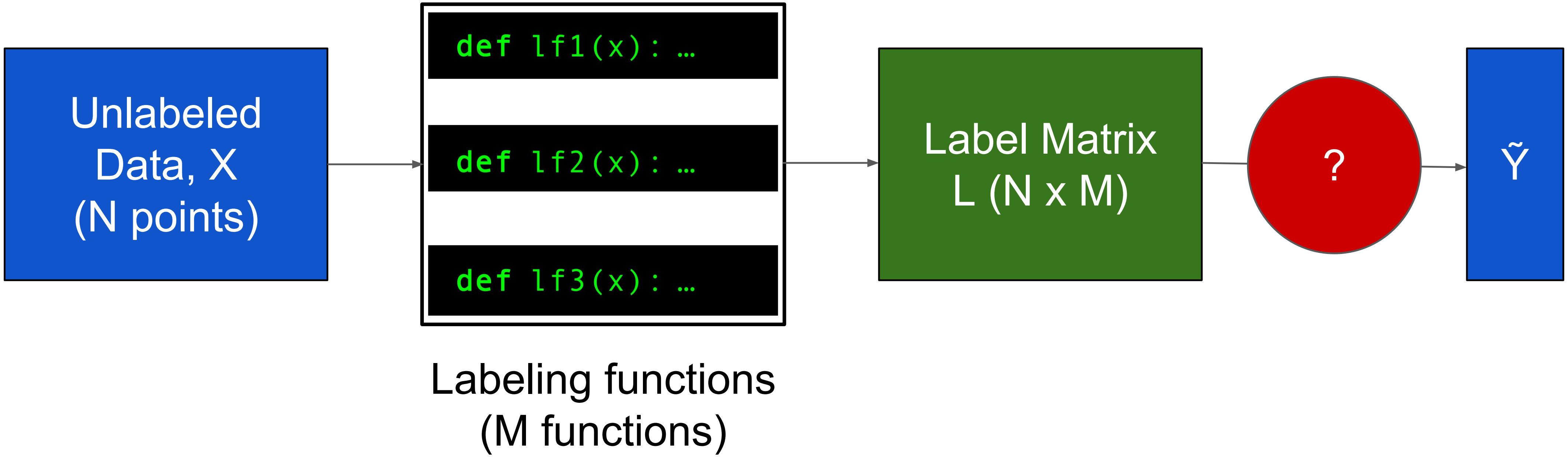
Integrating multiple noisy labels



Labeling functions
(M functions)

[Credit: http://cs231n.stanford.edu/slides/2018/cs231n_2018_ds07.pdf]

Integrating multiple noisy labels



[Credit: http://cs231n.stanford.edu/slides/2018/cs231n_2018_ds07.pdf]

Integrating multiple noisy labels

How do we obtain probabilistic labels, $\tilde{\mathbf{Y}}$, from the label matrix, **L**?

Approach 1 - Majority Vote

Take the majority vote of the labelling functions (LFs).

Let's say **L** = [[0, 1, 0, 1, 0]; [1, 1, 1, 1, 0]].

$\tilde{\mathbf{Y}}$ = [0, 1]

[Credit: http://cs231n.stanford.edu/slides/2018/cs231n_2018_ds07.pdf]

Integrating multiple noisy labels

How do we obtain probabilistic labels, $\tilde{\mathbf{Y}}$, from the label matrix, **L**?

Approach 1 - Majority Vote

Majority vote fails:

```
Indication: Chest pain. Findings:
Mediastinal contours are within
normal limits. Heart size is
within normal limits. No focal
consolidation, pneumothorax or
pleural effusion. Impression: No
acute cardiopulmonary
abnormality.
```

Normal Report

```python
def LF_pneumothorax(c):
  if re.search(r'pneumo.*', c.report.text):
      return "ABNORMAL"

def LF_pleural_effusion(c):
  if "pleural effusion" in c.report.text:
      return "ABNORMAL"

def LF_normal_report(c, thresh=2):
    if len(NORMAL_TERMS.intersection(c.
    report.words)) > thresh:
      return "NORMAL"
```

LFs

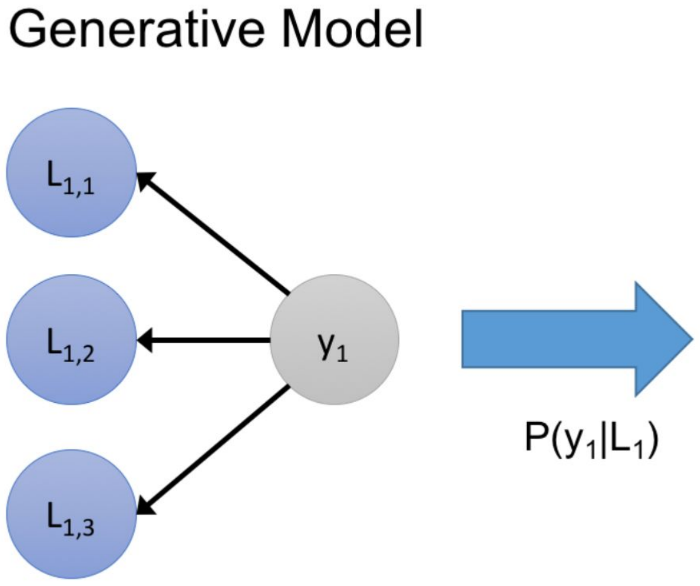
[Credit: http://cs231n.stanford.edu/slides/2018/cs231n_2018_ds07.pdf]

Integrating multiple noisy labels

How do we obtain probabilistic labels, $\tilde{\mathbf{Y}}$, from the label matrix, **L**?

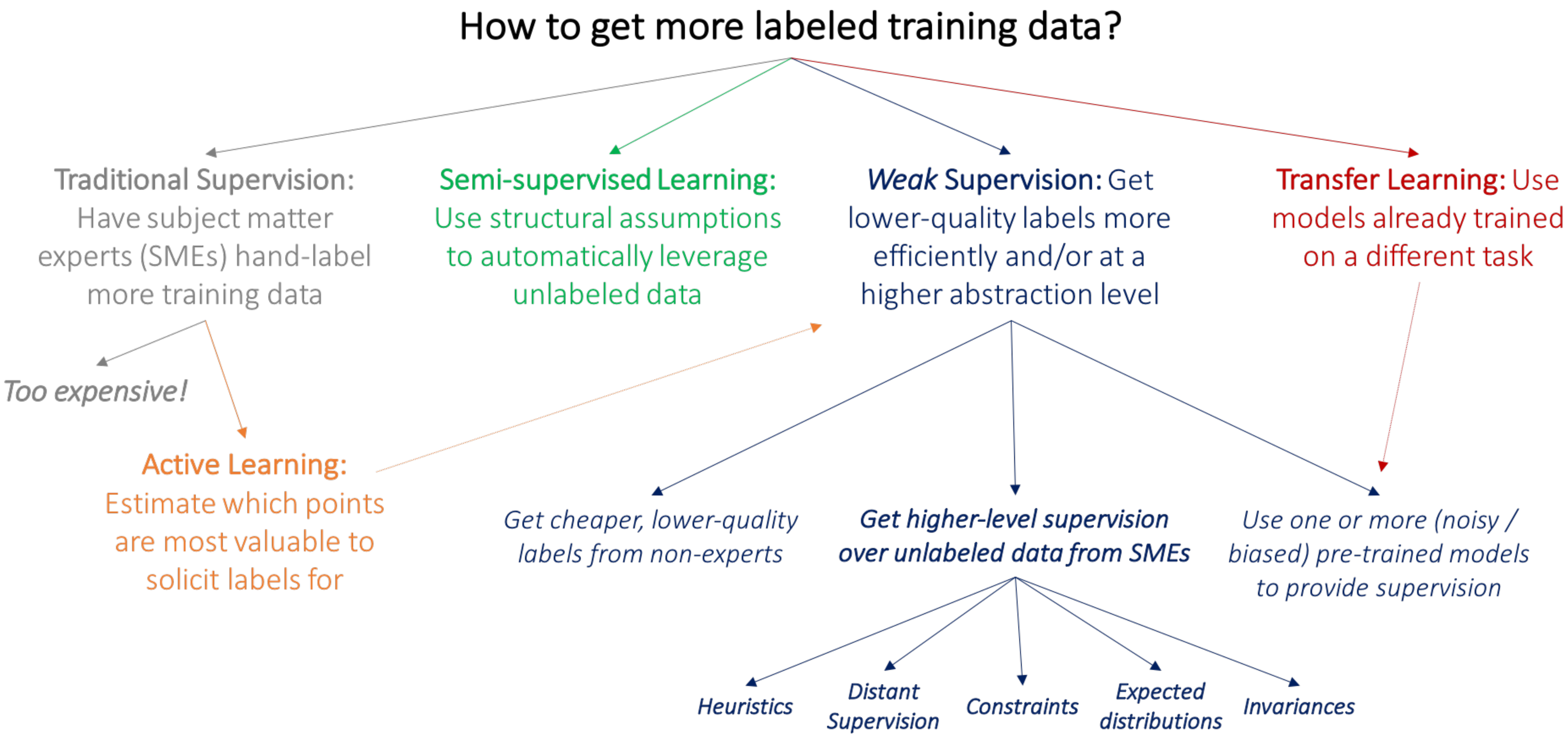
Approach 2

Train a generative model over **P(L, Y)** where **Y** are the **(unknown)** true labels



[Credit: http://cs231n.stanford.edu/slides/2018/cs231n_2018_ds07.pdf]

Summary: Weak/distant supervision



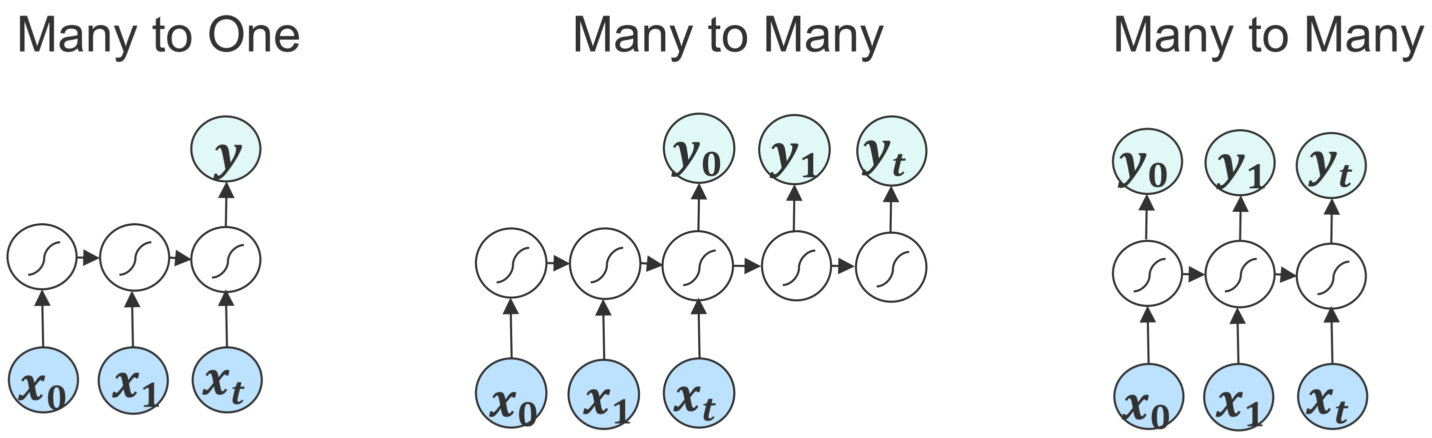
Sequence Labeling

[Slides adapted from UW CSE 447 by Noah Smith]

Motivation

Many tasks in NLP can be cast as **sequence labeling**, where each token (usually, word) gets its own label. Compare:

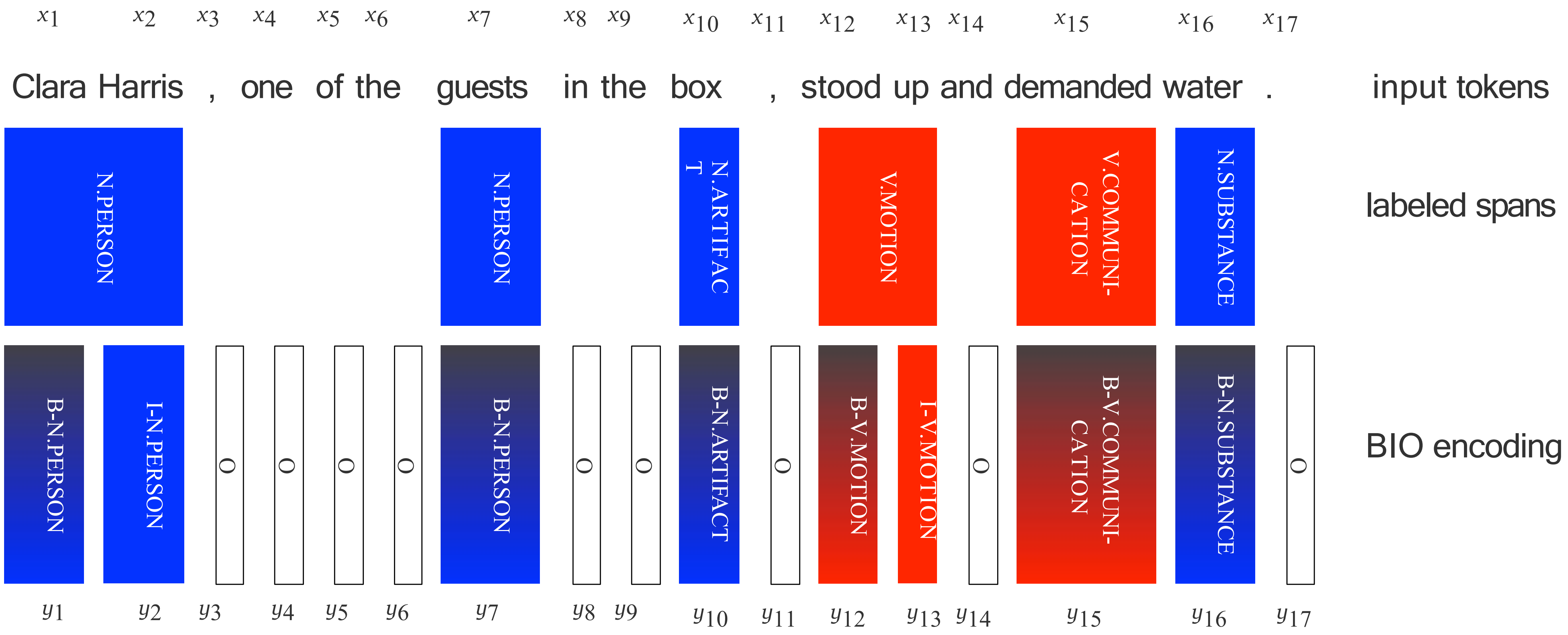
- Text classification: $\langle x_1, x_2, \ldots, x_n \rangle \rightarrowtail y \in L$
- Sequence labeling: $\langle x_1 \rightarrowtail y_1, x_2 \rightarrowtail y_2, \ldots, x_n \rightarrowtail y_n \rangle$, each $y_i \in L$
- Translation: $\boldsymbol{x} \rightarrowtail \boldsymbol{y}$



Problems Typically Cast as Sequence Labeling

- supersense tagging (Ciaramita and Johnson, 2003)
- part-of-speech tagging (Church, 1988)
- morphosyntactic tagging (Habash and Rambow, 2005)
- segmentation into words (Sproat et al., 1996) or multiword expressions (Schneider et al., 2014)
- code switching (Solorio and Liu, 2008)
- dialogue acts (Stolcke et al., 2000)
- spelling correction (Kernighan et al., 1990)
- word alignment (Vogel et al., 1996)
- named entity recognition (Bikel et al., 1999)
- compression (Conroy and O'Leary, 2001)

Supersense Tagging Example



Observations

- Lots of subproblems: Which words have supersenses? Which words group together to form a multiword expression? For those that do, which supersense?
- Every word's label depends on the words around it, and their labels.
- Segmentation problems can be cast as sequence labeling (Ramshaw and Marcus, 1995):
  - Two labels, B and I, if every word must be in some segment
  - Three labels, B, I, and O, if some words are to be "discarded"
  - Variants for five labels (E for end, S for singleton), gaps/noncontiguous spans, and nesting, exist.

  Concatenate B, I, etc., with labels to get labeled segmentation.
- Some sequences of labels might be invalid under your theory/label semantics.
- Evaluation: usually precision, recall, and F_1 on labeled segments.

Sequence Labeling

Problem statement: given a sequence of n words $\boldsymbol{x}$, assign each a label from $\mathcal{L}$. Let $L = |\mathcal{L}|$.

Every approach we see today will cast the problem as:

$$\hat{\boldsymbol{y}} = \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{L}^n} \mathrm{Score}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$$

Naïvely, that's a classification problem where the number of possible 'labels" (output sequences) depends on the input and is $O(L^n)$ in size!

Sequence Labeling v. 0: Local Classifiers

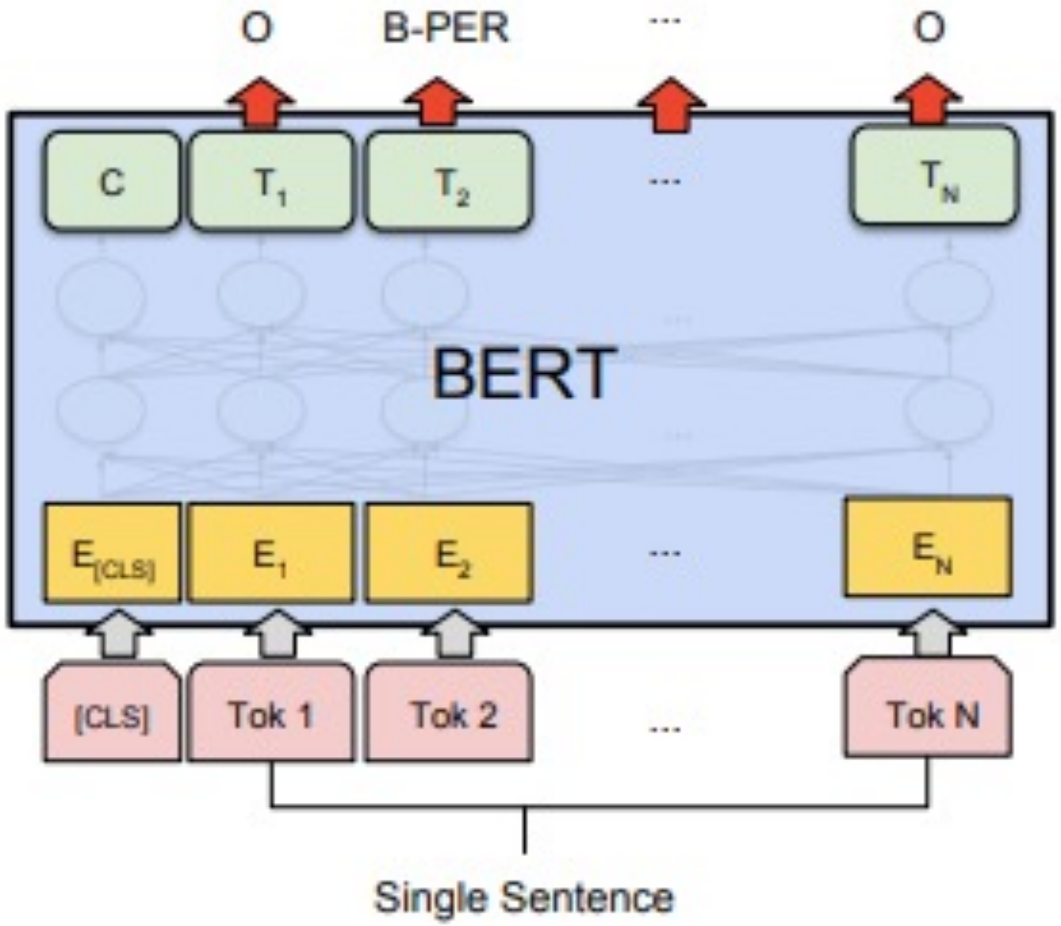
Define score of a word x_i getting label $y \in \mathcal{L}$ in context: $\mathrm{score}(\boldsymbol{x}, i, y; \boldsymbol{\theta})$, for example through a feature vector, $\mathbf{f}(\boldsymbol{x}, i, y)$. (Here, "$i$"' indicates the position of the input word to be classified.)

Train a classifier to decode locally, i.e.,

$$\begin{aligned} \hat{y}_i &= \operatorname*{argmax}_{y \in \mathcal{L}} \mathrm{score}(\boldsymbol{x}, i, y; \boldsymbol{\theta}) \\ &= \operatorname*{argmax}_{y \in \mathcal{L}} \boldsymbol{\theta}^\top \mathbf{f}(\boldsymbol{x}, i, y) \end{aligned}$$

The classifier is applied to each $x_1, x_2, \ldots$ in turn, but all the words can be made available at each position.

Sequence Labeling v. 0: Local Classifiers



Sequence Labeling v. 0: Local Classifiers

We can do better when there are predictable relationships among labels.

$$\hat{\boldsymbol{y}} = \underset{\boldsymbol{y} \in \mathcal{L}^n}{\operatorname{argmax}} \operatorname{Score}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$$

Sequence Labeling v. 1: Sequential Classifiers

Define score of a word x_i getting label y in context, *including previous labels*: $\mathrm{score}(\boldsymbol{x}, i, \hat{\boldsymbol{y}}_{1:i-1}, y; \boldsymbol{\theta})$. (From here, we won't always write $\boldsymbol{\theta}$, but the dependence remains.)

Train a classifier, e.g.,

$$\hat{y}_i = \operatorname*{argmax}_{y \in \mathcal{L}} \mathrm{score}(\boldsymbol{x}, i, \hat{\boldsymbol{y}}_{1:i-1}, y)$$

The classifier is applied to each $x_1, x_2, \ldots$ in turn. *Each one depends on the outputs of preceding iterations.*

Sequence Labeling v. 1: Sequential Classifiers

Define score of a word x_i getting label y in context, *including previous labels*: $\text{score}(\boldsymbol{x}, i, \hat{\boldsymbol{y}}_{1:i-1}, y; \boldsymbol{\theta})$. (From here, we won't always write $\boldsymbol{\theta}$, but the dependence remains.)

Train a classifier, e.g.,

$$\hat{y}_i = \operatorname*{argmax}_{y \in \mathcal{L}} \text{score}(\boldsymbol{x}, i, \hat{\boldsymbol{y}}_{1:i-1}, y)$$

The classifier is applied to each $x_1, x_2, \ldots$ in turn. *Each one depends on the outputs of preceding iterations.*

Drawback: "downstream" effects of a mistake can be catastrophic.

There is much literature on methods for training, and for decoding, with models like this. Important decoding method in NLP: beam search.

Beam Search for Sequential Classifiers

Input: $\boldsymbol{x}$ (length n), a sequential classifier's scoring function score, and beam width k

Let H_0 score hypotheses at position 0, defining only $H_0(\langle\rangle) = 0$.
For $i \in \{1, \ldots, n\}$:

- Empty C.
- For each hypothesis $\hat{\boldsymbol{y}}_{1:i-1}$ scored by H_{i-1}:
  - For each $y \in \mathcal{L}$, place new hypothesis $\hat{\boldsymbol{y}}_{1:i}y \rightarrow H_{i-1}(\hat{\boldsymbol{y}}_{1:i-1}) + \mathrm{score}(\boldsymbol{x}, i, \hat{\boldsymbol{y}}_{1:i-1}, y)$ into C.
- Let H_i be the k-best scored elements of C.

Output: best scored element of H_n.

Beam Search for Sequential Classifiers

- Runtime is $O(n^2 kL)$, space is $O(n^2 k)$.
- You can improve runtime (e.g., to $O(nkL)$) if computation is shared across different i (often true with neural networks).
- Special cases:
  - $k = 1$ is greedy left-to-right decoding.
  - At $k = L^n$, you're doing brute force, exhaustive search.

A Generative Approach

- The next approach should remind you of language models. It assumes that labeled sequences are generated according to the following story:

y_1

$$y_1 \sim p_{start}(Y)$$

A Generative Approach

- The next approach should remind you of language models. It assumes that labeled sequences are generated according to the following story:

$$\begin{array}{c} x_1 \\ \uparrow \\ y_1 \end{array}$$

$$x_1 \sim p_{emission}(X \mid y_1)$$

A Generative Approach

- The next approach should remind you of language models. It assumes that labeled sequences are generated according to the following story:

$$
\begin{array}{ccc}
x_1 & & \\
\uparrow & & \\
y_1 & \rightarrow & y_2
\end{array}
$$

$$y_2 \sim p_{transition}(Y \mid y_1)$$

A Generative Approach

- The next approach should remind you of language models. It assumes that labeled sequences are generated according to the following story:

$$\begin{array}{ccc} x_1 & & x_2 \\ \uparrow & & \uparrow \\ y_1 & \rightarrow & y_2 \end{array}$$

$$x_2 \sim p_{emission}(X \mid y_2)$$

A Generative Approach

- The next approach should remind you of language models. It assumes that labeled sequences are generated according to the following story:

$$\begin{array}{ccccc} x_1 & & x_2 & & \\ \uparrow & & \uparrow & & \\ y_1 & \rightarrow & y_2 & \rightarrow & y_3 \end{array}$$

$$y_3 \sim p_{transition}(Y \mid y_2)$$

A Generative Approach

- The next approach should remind you of language models. It assumes that labeled sequences are generated according to the following story:

$$
\begin{array}{ccccc}
x_1 & & x_2 & & x_3 \\
\uparrow & & \uparrow & & \uparrow \\
y_1 & \rightarrow & y_2 & \rightarrow & y_3
\end{array}
$$

$$x_3 \sim p_{emission}(X \mid y_3)$$

A Generative Approach

- The next approach should remind you of language models. It assumes that labeled sequences are generated according to the following story:

$$
\begin{array}{ccccccc}
x_1 & & x_2 & & x_3 & & \\
\uparrow & & \uparrow & & \uparrow & & \\
y_1 & \rightarrow & y_2 & \rightarrow & y_3 & \rightarrow &
\end{array}
$$

$$y_4 \sim p_{transition}(Y \mid y_3)$$

Sequence Labeling v. 2: Hidden Markov Models

By convention, $y_{n+1} =$ is always the "stop label."

$$
\begin{aligned}
p(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y}) &= p_{start}(y_1) \cdot \\
&\quad \prod_{i=1}^{n} p_{emission}(x_i \mid y_i) \cdot p_{transition}(y_{i+1} \mid y_i)
\end{aligned}
$$

$$
\begin{aligned}
\hat{\boldsymbol{y}} &= \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{L}^n} p(\boldsymbol{Y} = \boldsymbol{y} \mid \boldsymbol{X} = \boldsymbol{x}) \\
&= \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{L}^n} p(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y}) \\
&= \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{L}^n} \log p(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y})
\end{aligned}
$$

We can solve the global decoding problem *exactly* (i.e., find the model-optimal $\hat{\boldsymbol{y}}$) in $O(nL^2)$ time and $O(nL)$ space using the Viterbi algorithm (more later).

- Quadratic complexity on the size of $\mathcal{L}$
- For some problems the label set can be large

HMM Parameters

Classical HMM parameters are all interpretable as probabilities of events.

p_{start} is a distribution over $\mathcal{L}$. We estimate it by counting how often sequences start with each label in the training data, and normalizing.

$p_{emission}$ is a distribution over words, for each label. Many people find this counterintuitive! Estimation: counting occurrences of labels with words, and normalizing (per label, not per word).

$p_{transition}$ is exactly a bigram (first-order Markov) model over labels.

Parameterized version

- Replace the “lookup” probabilities ($p_{transition}$, $p_{emission}$, p_{start}) with scoring functions

Classical HMM (v. 2):

$$\hat{\boldsymbol{y}} = \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{L}^n} \log p_{start}(y_1) + \sum_{i=1}^{n} \left(\begin{array}{l} \log p_{emission}(x_i \mid y_i) \\ + \ \log p_{transition}(y_{i+1} \mid y_i) \end{array} \right)$$

This approach (v. 3):

$$\hat{\boldsymbol{y}} = \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{L}^n} s_{start}(y_1) + \sum_{i=1}^{n} s_{emission}(x_i, y_i) + s_{transition}(y_i, y_{i+1})$$

Parameterized version: Note

- Decoding is essentially the same as the classical HMM: Viterbi algorithm.

- Learning is now complicated and depends on the form of each "s" (but is still efficient as we will see later)

- No part of the the scoring function looks at neighboring words.

Parameterized version (cont'd)

Let each scoring component ("s") "see" the whole input. By convention, $y_0 = \bigcirc$ is always the "start label."

$$\hat{\boldsymbol{y}} = \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{L}^n} \overbrace{\sum_{i=0}^{n} s(\boldsymbol{x}, i, y_i, y_{i+1})}^{\text{Score}(\boldsymbol{x}, \boldsymbol{y})}$$

Note that $\boldsymbol{x}$ can have arbitrary length, so we need "s" functions that are capable of adapting to variable-length input.

Summary so far

Model	0	1	2	3	4
Score decomp.	$s(\boldsymbol{x}, i, y_i)$	$s(\boldsymbol{x}, i, \boldsymbol{y}_{1:i})$	emission/ transition	$s(x_i, y_i)+$ $s(y_i, y_{i+1})$	$s(\boldsymbol{x}, i, y_i, y_{i+1})$
learn	SGD	?	count & normalize	?	?
decode	local	beam search	Viterbi	Viterbi	Viterbi

Two Problems to Solve

1. Decoding: the Viterbi algorithm for choosing $\hat{\boldsymbol{y}}$.
    - Usually taught for classical HMMs (v. 2); I will teach it for v. 4, abstracting away "s."
2. Learning: estimating the parameters of each s function.
    - Depending on your choices here, you arrive at the structured perceptron, the classical conditional random field (CRF), neural CRFs, and more.

A Data Structure

input sequence

labels in $\mathcal{L}$

	x_1	x_2	$\ldots$	x_n
ℓ_1				
ℓ_2				
$\vdots$				
ℓ_L				

The cell at row j, column i will hold information pertaining to choosing $\hat{y}_i = \ell_j$.

The End of the Sequence

input sequence

labels in $\mathcal{L}$

	x_1	x_2	$\dots$	x_n
ℓ_1				
ℓ_2				
$\vdots$				
ℓ_L				

$$
\begin{aligned}
\hat{y}_n &= \operatorname*{argmax}_{y_n \in \mathcal{L}} \sum_{i=0}^{n} s(\boldsymbol{x}, i, y_i, y_{i+1}) \\
&= \operatorname*{argmax}_{y_n \in \mathcal{L}} s(\boldsymbol{x}, i, y_{n-1}, y_n) + s(\boldsymbol{x}, i, y_n,)
\end{aligned}
$$

The decision about $\hat{y}_n$ is a function of y_{n-1}, $\boldsymbol{x}$, and nothing else!

High-Level View of the Viterbi Algorithm

- The decision about $\hat{y}_n$ is a function of y_{n-1}, $\boldsymbol{x}$, and nothing else!

High-Level View of the Viterbi Algorithm

- The decision about $\hat{y}_n$ is a function of y_{n-1}, $\boldsymbol{x}$, and nothing else!
- If, for each value of y_{n-1}, we knew the best $(n-1)$-length label prefix $\boldsymbol{y}_{1:n-1}$, then picking $\hat{y}_n$ (and $\hat{y}_{n-1}$) would be easy.

High-Level View of the Viterbi Algorithm

- The decision about $\hat{y}_n$ is a function of y_{n-1}, $\boldsymbol{x}$, and nothing else!
- If, for each value of y_{n-1}, we knew the best $(n-1)$-length label prefix $\boldsymbol{y}_{1:n-1}$, then picking $\hat{y}_n$ (and $\hat{y}_{n-1}$) would be easy.
- Idea: for each position i, calculate the score of the best label prefix $\boldsymbol{y}_{1:i}$ ending in each possible value for the ith label.
  - We'll call this value $\heartsuit_i(\ell)$ for $y_i = \ell$.

High-Level View of the Viterbi Algorithm

- The decision about $\hat{y}_n$ is a function of y_{n-1}, $\boldsymbol{x}$, and nothing else!
- If, for each value of y_{n-1}, we knew the best $(n-1)$-length label prefix $\boldsymbol{y}_{1:n-1}$, then picking $\hat{y}_n$ (and $\hat{y}_{n-1}$) would be easy.
- Idea: for each position i, calculate the score of the best label prefix $\boldsymbol{y}_{1:i}$ ending in each possible value for the ith label.
  - We'll call this value $\heartsuit_i(\ell)$ for $y_i = \ell$.
- With a little bookkeeping, we can then trace backwards and recover the best label sequence.

Recurrence

First, think about the *score* of the best sequence.

Let $\heartsuit_i(y)$ be the score of the best label sequence for $\boldsymbol{x}_{1:i}$ that ends in y. It is defined recursively:

$$\heartsuit_{n+1}(\textcolor{red}{\octagon}) = \max_{y_n \in \mathcal{L}} s(\boldsymbol{x}, n, y_n, \textcolor{red}{\octagon}) + \boxed{\heartsuit_n(y_n)}$$

Recurrence

First, think about the *score* of the best sequence.

Let $\heartsuit_i(y)$ be the score of the best label sequence for $\boldsymbol{x}_{1:i}$ that ends in y. It is defined recursively:

$$\heartsuit_{n+1}(\octagon) = \max_{y_n \in \mathcal{L}} s(\boldsymbol{x}, n, y_n, \octagon) + \boxed{\heartsuit_n(y_n)}$$

$$\heartsuit_n(y) = \max_{y_{n-1} \in \mathcal{L}} s(\boldsymbol{x}, n-1, y_{n-1}, y) + \boxed{\heartsuit_{n-1}(y_{n-1})}$$

Recurrence

First, think about the *score* of the best sequence.

Let $\heartsuit_i(y)$ be the score of the best label sequence for $\boldsymbol{x}_{1:i}$ that ends in y. It is defined recursively:

$$\begin{aligned}
\heartsuit_{n+1}(\octagon) &= \max_{y_n \in \mathcal{L}} s(\boldsymbol{x}, n, y_n, \octagon) + \boxed{\heartsuit_n(y_n)} \\
\heartsuit_n(y) &= \max_{y_{n-1} \in \mathcal{L}} s(\boldsymbol{x}, n-1, y_{n-1}, y) + \boxed{\heartsuit_{n-1}(y_{n-1})} \\
\heartsuit_{n-1}(y) &= \max_{y_{n-2} \in \mathcal{L}} s(\boldsymbol{x}, n-2, y_{n-2}, y) + \boxed{\heartsuit_{n-2}(y_{n-2})}
\end{aligned}$$

Recurrence

First, think about the *score* of the best sequence.

Let $\heartsuit_i(y)$ be the score of the best label sequence for $\boldsymbol{x}_{1:i}$ that ends in y. It is defined recursively:

$$
\begin{aligned}
\heartsuit_{n+1}(\octagon) &= \max_{y_n \in \mathcal{L}} s(\boldsymbol{x}, n, y_n, \octagon) + \boxed{\heartsuit_n(y_n)} \\
\heartsuit_n(y) &= \max_{y_{n-1} \in \mathcal{L}} s(\boldsymbol{x}, n-1, y_{n-1}, y) + \boxed{\heartsuit_{n-1}(y_{n-1})} \\
\heartsuit_{n-1}(y) &= \max_{y_{n-2} \in \mathcal{L}} s(\boldsymbol{x}, n-2, y_{n-2}, y) + \boxed{\heartsuit_{n-2}(y_{n-2})} \\
&\vdots \\
\heartsuit_i(y) &= \max_{y_{i-1} \in \mathcal{L}} s(\boldsymbol{x}, i-1, y_{i-1}, y) + \boxed{\heartsuit_{i-1}(y_{i-1})}
\end{aligned}
$$

Recurrence

First, think about the *score* of the best sequence.

Let $\heartsuit_i(y)$ be the score of the best label sequence for $\boldsymbol{x}_{1:i}$ that ends in y. It is defined recursively:

$$\heartsuit_{n+1}(\text{⯃}) = \max_{y_n \in \mathcal{L}} s(\boldsymbol{x}, n, y_n, \text{⯃}) + \boxed{\heartsuit_n(y_n)}$$

$$\heartsuit_n(y) = \max_{y_{n-1} \in \mathcal{L}} s(\boldsymbol{x}, n-1, y_{n-1}, y) + \boxed{\heartsuit_{n-1}(y_{n-1})}$$

$$\heartsuit_{n-1}(y) = \max_{y_{n-2} \in \mathcal{L}} s(\boldsymbol{x}, n-2, y_{n-2}, y) + \boxed{\heartsuit_{n-2}(y_{n-2})}$$

$$\vdots$$

$$\heartsuit_i(y) = \max_{y_{i-1} \in \mathcal{L}} s(\boldsymbol{x}, i-1, y_{i-1}, y) + \boxed{\heartsuit_{i-1}(y_{i-1})}$$

$$\vdots$$

$$\heartsuit_1(y) = s(\boldsymbol{x}, 0, \bigcirc, y)$$

Viterbi Procedure (Part I: Prefix Scores)

input sequence

		x_1	x_2	$\ldots$	x_n	
	ℓ_1					
$\mathcal{L}$	ℓ_2					
	$\vdots$					
	ℓ_L					

Viterbi Procedure (Part I: Prefix Scores)

input sequence

		x_1	x_2	$\ldots$	x_n	
	ℓ_1	$\heartsuit_1(\ell_1)$				
	ℓ_2	$\heartsuit_1(\ell_2)$				
$\mathcal{L}$	$\vdots$					
	ℓ_L	$\heartsuit_1(\ell_L)$				

$$\heartsuit_1(y) = s(\boldsymbol{x}, 0, \bigcirc, y)$$

Viterbi Procedure (Part I: Prefix Scores)

		input sequence				
		x_1	x_2	$\ldots$	x_n	
	ℓ_1	$\heartsuit_1(\ell_1)$	$\heartsuit_2(\ell_1)$			
$\mathcal{L}$	ℓ_2	$\heartsuit_1(\ell_2)$	$\heartsuit_2(\ell_2)$			
	$\vdots$					
	ℓ_L	$\heartsuit_1(\ell_L)$	$\heartsuit_2(\ell_L)$			

$$\heartsuit_i(y) = \max_{y_{i-1} \in \mathcal{L}} s(\boldsymbol{x}, i-1, y_{i-1}, y) + \boxed{\heartsuit_{i-1}(y_{i-1})}$$

Viterbi Procedure (Part I: Prefix Scores)

input sequence

		x_1	x_2	$\ldots$	x_n	
$\mathcal{L}$	ℓ_1	$\heartsuit_1(\ell_1)$	$\heartsuit_2(\ell_1)$		$\heartsuit_n(\ell_1)$	
	ℓ_2	$\heartsuit_1(\ell_2)$	$\heartsuit_2(\ell_2)$		$\heartsuit_n(\ell_2)$	
	$\vdots$					
	ℓ_L	$\heartsuit_1(\ell_L)$	$\heartsuit_2(\ell_L)$		$\heartsuit_n(\ell_L)$	

$$\heartsuit_n(y) = \max_{y_{n-1} \in \mathcal{L}} s(\boldsymbol{x}, n-1, y_{n-1}, y) + \boxed{\heartsuit_{n-1}(y_{n-1})}$$

Viterbi Procedure (Part I: Prefix Scores)

input sequence

		x_1	x_2	$\ldots$	x_n	
	ℓ_1	$\heartsuit_1(\ell_1)$	$\heartsuit_2(\ell_1)$		$\heartsuit_n(\ell_1)$	
$\mathcal{L}$	ℓ_2	$\heartsuit_1(\ell_2)$	$\heartsuit_2(\ell_2)$		$\heartsuit_n(\ell_2)$	
	$\vdots$					
	ℓ_L	$\heartsuit_1(\ell_L)$	$\heartsuit_2(\ell_L)$		$\heartsuit_n(\ell_L)$	
	$\bigcirc$					$\heartsuit_{n+1}(\bigcirc)$

$$\heartsuit_{n+1}(\bigcirc) = \max_{y_n \in \mathcal{L}} s(\boldsymbol{x}, n, y_n, \bigcirc) + \boxed{\heartsuit_n(y_n)}$$

High-Level View of the Viterbi Algorithm

- The decision about $\hat{y}_n$ is a function of y_{n-1}, $\boldsymbol{x}$, and nothing else!
- If, for each value of y_{n-1}, we knew the best $(n-1)$-length label prefix $\boldsymbol{y}_{1:n-1}$, then picking $\hat{y}_n$ (and $\hat{y}_{n-1}$) would be easy.
- Idea: for each position i, calculate the score of the best label prefix $\boldsymbol{y}_{1:i}$ ending in each possible value for the ith label.
  - We'll call this value $\heartsuit_i(\ell)$ for $y_i = \ell$.
- With a little bookkeeping, we can then trace backwards and recover the best label sequence.

Viterbi Procedure (Part I: Prefix Scores and Backpointers)

input sequence

$\mathcal{L}$		x_1	x_2	$\ldots$	x_n	
	ℓ_1					
	ℓ_2					
	$\vdots$					
	ℓ_L					

Viterbi Procedure (Part I: Prefix Scores and Backpointers)

input sequence

$\mathcal{L}$

	x_1	x_2	$\ldots$	x_n	
ℓ_1	$\heartsuit_1(\ell_1)$ $\text{bp}_1(\ell_1)$				
ℓ_2	$\heartsuit_1(\ell_2)$ $\text{bp}_1(\ell_2)$				
$\vdots$					
ℓ_L	$\heartsuit_1(\ell_L)$ $\text{bp}_1(\ell_L)$				

$$\heartsuit_1(y) = s(\boldsymbol{x}, 0, \bigcirc, y)$$
$$\text{bp}_1(y) = \bigcirc$$

Viterbi Procedure (Part I: Prefix Scores and Backpointers)

input sequence

		x_1	x_2	$\ldots$	x_n	
	ℓ_1	$\heartsuit_1(\ell_1)$ $\mathrm{bp}_1(\ell_1)$	$\heartsuit_2(\ell_1)$ $\mathrm{bp}_2(\ell_1)$			
	ℓ_2	$\heartsuit_1(\ell_2)$ $\mathrm{bp}_1(\ell_2)$	$\heartsuit_2(\ell_2)$ $\mathrm{bp}_2(\ell_2)$			
$\mathcal{L}$	$\vdots$					
	ℓ_L	$\heartsuit_1(\ell_L)$ $\mathrm{bp}_1(\ell_L)$	$\heartsuit_2(\ell_L)$ $\mathrm{bp}_2(\ell_L)$			

$$\heartsuit_i(y) = \max_{y_{i-1} \in \mathcal{L}} s(\boldsymbol{x}, i-1, y_{i-1}, y) + \boxed{\heartsuit_{i-1}(y_{i-1})}$$

$$\mathrm{bp}_i(y) = \operatorname*{argmax}_{y_{i-1} \in \mathcal{L}} s(\boldsymbol{x}, i-1, y_{i-1}, y) + \boxed{\heartsuit_{i-1}(y_{i-1})}$$

Viterbi Procedure (Part I: Prefix Scores and Backpointers)

input sequence

$\mathcal{L}$

	x_1	x_2	$\ldots$	x_n	
ℓ_1	$\heartsuit_1(\ell_1)$ $\mathrm{bp}_1(\ell_1)$	$\heartsuit_2(\ell_1)$ $\mathrm{bp}_2(\ell_1)$		$\heartsuit_n(\ell_1)$ $\mathrm{bp}_n(\ell_1)$	
ℓ_2	$\heartsuit_1(\ell_2)$ $\mathrm{bp}_1(\ell_2)$	$\heartsuit_2(\ell_2)$ $\mathrm{bp}_2(\ell_2)$		$\heartsuit_n(\ell_2)$ $\mathrm{bp}_n(\ell_2)$	
$\vdots$					
ℓ_L	$\heartsuit_1(\ell_L)$ $\mathrm{bp}_1(\ell_L)$	$\heartsuit_2(\ell_L)$ $\mathrm{bp}_2(\ell_L)$		$\heartsuit_n(\ell_L)$ $\mathrm{bp}_n(\ell_L)$	

$$\heartsuit_n(y) = \max_{y_{n-1} \in \mathcal{L}} s(\boldsymbol{x}, n-1, y_{n-1}, y) + \boxed{\heartsuit_{n-1}(y_{n-1})}$$

$$\mathrm{bp}_n(y) = \operatorname*{argmax}_{y_{n-1} \in \mathcal{L}} s(\boldsymbol{x}, n-1, y_{n-1}, y) + \boxed{\heartsuit_{n-1}(y_{n-1})}$$

Viterbi Procedure (Part I: Prefix Scores and Backpointers)

input sequence

		x_1	x_2	$\ldots$	x_n	
	ℓ_1	$\heartsuit_1(\ell_1)$ $\text{bp}_1(\ell_1)$	$\heartsuit_2(\ell_1)$ $\text{bp}_2(\ell_1)$		$\heartsuit_n(\ell_1)$ $\text{bp}_n(\ell_1)$	
	ℓ_2	$\heartsuit_1(\ell_2)$ $\text{bp}_1(\ell_2)$	$\heartsuit_2(\ell_2)$ $\text{bp}_2(\ell_2)$		$\heartsuit_n(\ell_2)$ $\text{bp}_n(\ell_2)$	
$\mathcal{L}$	$\vdots$					
	ℓ_L	$\heartsuit_1(\ell_L)$ $\text{bp}_1(\ell_L)$	$\heartsuit_2(\ell_L)$ $\text{bp}_2(\ell_L)$		$\heartsuit_n(\ell_L)$ $\text{bp}_n(\ell_L)$	
	$\octagon$					$\heartsuit_{n+1}(\octagon)$ $\text{bp}_{n+1}(\octagon)$

$$\heartsuit_{n+1}(\octagon) = \max_{y_n \in \mathcal{L}} s(\boldsymbol{x}, n, y_n, \octagon) + \boxed{\heartsuit_n(y_n)}$$

$$\text{bp}_{n+1}(\octagon) = \operatorname*{argmax}_{y_n \in \mathcal{L}} s(\boldsymbol{x}, n, y_n, \octagon) + \boxed{\heartsuit_n(y_n)}$$

Full Viterbi Procedure

Input: scores $s(\boldsymbol{x}, i, y, y')$, for all $i \in \{0, \ldots, n\}$, $y, y' \in \mathcal{L}$

Output: $\hat{\boldsymbol{y}}$

1. Base case: $\heartsuit_1(y) = s(\boldsymbol{x}, 0, \bigcirc, y)$
2. For $i \in \{2, \ldots, n+1\}$:
   - Solve for $\heartsuit_i(*)$ and $\mathrm{bp}_i(*)$.

$$\heartsuit_i(y) = \max_{y_{i-1} \in \mathcal{L}} s(\boldsymbol{x}, i-1, y_{i-1}, y) + \heartsuit_{i-1}(y_{i-1}),$$

$$\mathrm{bp}_i(y) = \operatorname*{argmax}_{y_{i-1} \in \mathcal{L}} s(\boldsymbol{x}, i-1, y_{i-1}, y) + \heartsuit_{i-1}(y_{i-1})$$

   (At $n+1$ we're only interested in $y = \octagon$.)

3. $\hat{y}_{i+1} \leftarrow \octagon$
4. For $i \in \{n, \ldots, 1\}$:
   - $\hat{y}_i \leftarrow \mathrm{bp}_{i+1}(\hat{y}_{i+1})$

Viterbi Asymptotics

input sequence

labels in $\mathcal{L}$

	x_1	x_2	$\ldots$	x_n
ℓ_1				
ℓ_2				
$\vdots$				
ℓ_L				

Viterbi Asymptotics

input sequence

		x_1	x_2	$\ldots$	x_n
labels in $\mathcal{L}$	ℓ_1				
	ℓ_2				
	$\vdots$				
	ℓ_L				

Space: need to store s, and fill in the cells above.

Viterbi Asymptotics

input sequence

labels in $\mathcal{L}$

	x_1	x_2	$\dots$	x_n
ℓ_1				
ℓ_2				
$\vdots$				
ℓ_L				

Space: need to store s, and fill in the cells above. $O(nL^2)$ for s (in the most general case, often less), $O(nL)$ for cells

Viterbi Asymptotics

	input sequence			
	x_1	x_2	$\ldots$	x_n
ℓ_1				
ℓ_2				
labels in $\mathcal{L}$ $\vdots$				
ℓ_L				

Space: need to store s, and fill in the cells above. $O(nL^2)$ for s (in the most general case, often less), $O(nL)$ for cells

Runtime: each cell requires an "argmax."

Viterbi Asymptotics

	input sequence			
labels in $\mathcal{L}$	x_1	x_2	$\ldots$	x_n
ℓ_1				
ℓ_2				
$\vdots$				
ℓ_L				

Space: need to store s, and fill in the cells above. $O(nL^2)$ for s (in the most general case, often less), $O(nL)$ for cells

Runtime: each cell requires an "argmax." $O(nL^2)$

Why it Works

Viterbi exploits the distributivity property:

$$
\begin{aligned}
\max_{\boldsymbol{y}_{1:n}} \sum_{i=0}^{n} s(\boldsymbol{x}, i, y_i, y_{i+1}) &= \max_{y_n} s(\boldsymbol{x}, i, y_n, \bigcirc) + \max_{\boldsymbol{y}_{1:n-1}} \sum_{i=0}^{n-1} s(\boldsymbol{x}, i, y_i, y_{i+1}) \\
&= \max_{y_n} s(\boldsymbol{x}, i, y_n, \bigcirc) + \max_{y_{n-1}} s(\boldsymbol{x}, i, y_n - 1, y_n) \\
&\quad + \max_{\boldsymbol{y}_{1:n-2}} \sum_{i=0}^{n-2} s(\boldsymbol{x}, i, y_i, y_{i+1})
\end{aligned}
$$

Max plus max plus max plus max plus . . .

Back to "s"

We haven't said much about the function that scores candidate label pairs at different positions, $s(\boldsymbol{x}, i, y, y')$.

This function is very important; two common choices are:

- Expert-designed, task-specific features $\mathbf{f}(\boldsymbol{x}, i, y, y')$ and weights $\boldsymbol{\theta}$
- A neural network that encodes x_i in context, y_i, and y_{i+1} and gives back a goodness score

Either way, let $\boldsymbol{\theta}$ denote the parameters of s. From now on, we'll use $s(\boldsymbol{x}, i, y, y'; \boldsymbol{\theta})$ and $\mathrm{Score}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$ to emphasize that "s" is a function of parameters $\boldsymbol{\theta}$ we need to estimate.

Probabilistic View of Learning

As we've done before, we start with the principle of maximum likelihood to estimate $\boldsymbol{\theta}$:

$$\begin{aligned}
\boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta} \in \mathbb{R}^d} \prod_{i=1}^{T} p(\boldsymbol{Y} = \boldsymbol{y}_i \mid \boldsymbol{X} = \boldsymbol{x}_i; \boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta} \in \mathbb{R}^d} \sum_{i=1}^{T} \log p(\boldsymbol{Y} = \boldsymbol{y}_i \mid \boldsymbol{X} = \boldsymbol{x}_i; \boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^d} \sum_{i=1}^{T} \underbrace{- \log p(\boldsymbol{Y} = \boldsymbol{y}_i \mid \boldsymbol{X} = \boldsymbol{x}_i; \boldsymbol{\theta})}_{\text{sometimes called ``log loss'' or ``cross entropy''}}
\end{aligned}$$

Next, we'll drill down into "$p(\boldsymbol{Y} = \boldsymbol{y}_i \mid \boldsymbol{X} = \boldsymbol{x}_i; \boldsymbol{\theta})$."

Conditional Random Fields

Lafferty et al. (2001)

CRFs are a tremendously influential model that generalizes multinomial logistic regression to structured outputs like sequences.

$$p_{\text{CRF}}(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta}) = \frac{\exp \text{Score}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})}{Z(\boldsymbol{x}; \boldsymbol{\theta})}$$

$$Z(\boldsymbol{x}; \boldsymbol{\theta}) = \sum_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \exp \text{Score}(\boldsymbol{x}, \boldsymbol{y}'; \boldsymbol{\theta})$$

$$-\log p_{\text{CRF}}(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta}) = -\underbrace{\text{Score}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})}_{\text{“hope”}} + \underbrace{\log Z(\boldsymbol{x}; \boldsymbol{\theta})}_{\text{“fear”}}$$

So, our "CRF":

- Uses Viterbi for decoding (our v. 4 sequence labeler)
- Trains parameters to maximize likelihood (like MLR and NNs)

Sequence-Level Log Loss

Here's the maximum likelihood learning problem (equivalently, sequence-level log loss):

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta} \in \mathbb{R}^d} \sum_{i=1}^{T} -\mathrm{Score}(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{\theta}) + \log Z(\boldsymbol{x}_i; \boldsymbol{\theta})$$

If we can calculate and differentiate (w.r.t. $\boldsymbol{\theta}$) the Score and Z functions, we can use SGD to learn.

Reflection

Given a training instance $\langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle$, what do you need to do to calculate $\mathrm{Score}(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{\theta})$?

Calculating $Z(\boldsymbol{x}; \boldsymbol{\theta})$

Good news! The algorithm that gives us Z is *almost exactly like* the Viterbi algorithm.

Forward algorithm: sums the $\exp \text{Score}$ values for all label sequences, given $\boldsymbol{x}$, in the same asymptotic time and space as Viterbi.

Let $\alpha_i(y)$ be the sum of all (exponentiated) scores of label prefixes of length i, ending in y.

Some Algebra

Given the decomposition

$$\mathrm{Score}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) = \sum_{i=0}^{n} s(\boldsymbol{x}, i, y_i, y_{i+1}; \boldsymbol{\theta}),$$

it holds that

$$\exp \mathrm{Score}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) = \prod_{i=0}^{n} e^{s(\boldsymbol{x}, i, y_i, y_{i+1}; \boldsymbol{\theta})},$$

and therefore

$$Z(\boldsymbol{x}; \boldsymbol{\theta}) = \sum_{\boldsymbol{y'} \in \mathcal{Y}(\boldsymbol{x})} \prod_{i=0}^{n} e^{s(\boldsymbol{x}, i, y'_i, y'_{i+1}; \boldsymbol{\theta})}$$

Forward Algorithm

Input: scores $s(\boldsymbol{x}, i, y, y'; \boldsymbol{\theta})$, for all $i \in \{0, \ldots, n\}$, $y, y' \in \mathcal{L}$

Output: $Z(\boldsymbol{x}; \boldsymbol{\theta})$

1. Base case: $\alpha_1(y) = e^{s(\boldsymbol{x}, 0, \bigcirc, y; \boldsymbol{\theta})}$
2. For $i \in \{2, \ldots, n+1\}$:
   - Solve for $\alpha_i(*)$.

$$\alpha_i(y) = \sum_{y_{i-1} \in \mathcal{L}} e^{s(\boldsymbol{x}, i-1, y_{i-1}, y; \boldsymbol{\theta})} \times \alpha_{i-1}(y_{i-1})$$

   (At $n+1$ we're only interested in $y = \text{⯃}$.)
3. Return $\alpha_{n+1}(\text{⯃})$, which is equal to $Z(\boldsymbol{x}; \boldsymbol{\theta})$.

Intuitions about the Forward Algorithm

Just as Viterbi changes "scary max over big sum" to "max plus max plus max plus . . . ,"
the Forward algorithm changes "scary sum over big product" to "plus times plus times plus times . . . ."

If you organize the operations in the other direction, you get the Backward algorithm.

You can differentiate Z with respect to s, because it's all just $\exp$, addition, and multiplication. If you mechanically derive the partial derivatives, you will rediscover the Backward algorithm.

Questions?