Natural Language Processing (CSE 517 & 447): Conditional Random Fields

Noah Smith

University of Washington
nasmith@cs.washington.edu

Winter 2022

Readings: Eisenstein (2019) 7 and 8

Motivation

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

- Text classification: $\langle x_1, x_2, \ldots, x_n \rangle \mapsto y \in \mathcal{L}$
- Sequence labeling: $\langle x_1 \mapsto y_1, x_2 \mapsto y_2, \ldots, x_n \mapsto y_n \rangle$, each $y_i \in \mathcal{L}$
- Translation: $\boldsymbol{x} \mapsto \boldsymbol{y} \in \mathcal{V}^*_{target}$

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)

Example Problem: Supersenses

A problem with a long history: word-sense disambiguation.

Example Problem: Supersenses

A problem with a long history: word-sense disambiguation.

Classical approaches assumed you had a list of ambiguous words and their senses.

- E.g., from a dictionary

Example Problem: Supersenses

A problem with a long history: word-sense disambiguation.

Classical approaches assumed you had a list of ambiguous words and their senses.

- E.g., from a dictionary

Ciaramita and Johnson (2003) and Ciaramita and Altun (2006) used a lexicon called WordNet to define 41 semantic classes for words.

- WordNet (Fellbaum, 1998) is a fascinating resource in its own right! See `http://wordnetweb.princeton.edu/perl/webwn` to get an idea.

Example Problem: Supersenses

A problem with a long history: word-sense disambiguation.

Classical approaches assumed you had a list of ambiguous words and their senses.

- E.g., from a dictionary

Ciaramita and Johnson (2003) and Ciaramita and Altun (2006) used a lexicon called WordNet to define 41 semantic classes for words.

- WordNet (Fellbaum, 1998) is a fascinating resource in its own right! See `http://wordnetweb.princeton.edu/perl/webwn` to get an idea.

This represents a coarsening of the annotations in the Semcor corpus (Miller et al., 1993).

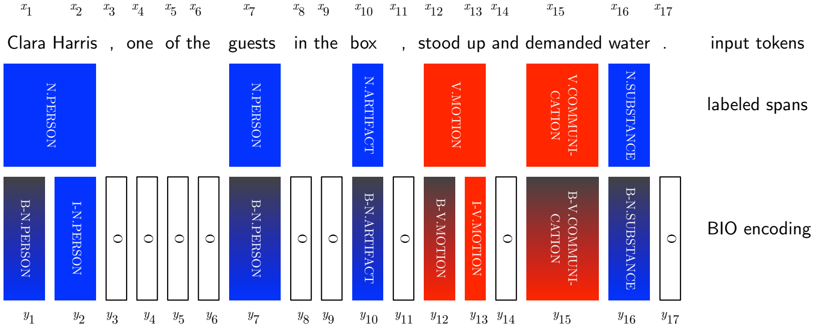
Example: *box*'s Thirteen Synonym Sets, Eight Supersenses

1. box: a (usually rectangular) container; may have a lid. "he rummaged through a box of spare parts"
2. box/loge: private area in a theater or grandstand where a small group can watch the performance. "the royal box was empty"
3. box/boxful: the quantity contained in a box. "he gave her a box of chocolates"
4. corner/box: a predicament from which a skillful or graceful escape is impossible. "his lying got him into a tight corner"
5. box: a rectangular drawing. "the flowchart contained many boxes"
6. box/boxwood: evergreen shrubs or small trees
7. box: any one of several designated areas on a ball field where the batter or catcher or coaches are positioned. "the umpire warned the batter to stay in the batter's box"
8. box/box seat: the driver's seat on a coach. "an armed guard sat in the box with the driver"
9. box: separate partitioned area in a public place for a few people. "the sentry stayed in his box to avoid the cold"
10. box: a blow with the hand (usually on the ear). "I gave him a good box on the ear"
11. box/package: put into a box. "box the gift, please"
12. box: hit with the fist. "I'll box your ears!"
13. box: engage in a boxing match.

Example: *box*'s Thirteen Synonym Sets, Eight Supersenses

1. box: a (usually rectangular) container; may have a lid. "he rummaged through a box of spare parts" ⇝ N.ARTIFACT
2. box/loge: private area in a theater or grandstand where a small group can watch the performance. "the royal box was empty" ⇝ N.ARTIFACT
3. box/boxful: the quantity contained in a box. "he gave her a box of chocolates" ⇝ N.QUANTITY
4. corner/box: a predicament from which a skillful or graceful escape is impossible. "his lying got him into a tight corner" ⇝ N.STATE
5. box: a rectangular drawing. "the flowchart contained many boxes" ⇝ N.SHAPE
6. box/boxwood: evergreen shrubs or small trees ⇝ N.PLANT
7. box: any one of several designated areas on a ball field where the batter or catcher or coaches are positioned. "the umpire warned the batter to stay in the batter's box" ⇝ N.ARTIFACT
8. box/box seat: the driver's seat on a coach. "an armed guard sat in the box with the driver" ⇝ N.ARTIFACT
9. box: separate partitioned area in a public place for a few people. "the sentry stayed in his box to avoid the cold" ⇝ N.ARTIFACT
10. box: a blow with the hand (usually on the ear). "I gave him a good box on the ear" ⇝ N.ACT
11. box/package: put into a box. "box the gift, please" ⇝ V.CONTACT
12. box: hit with the fist. "I'll box your ears!" ⇝ V.CONTACT
13. box: engage in a boxing match. ⇝ V.COMPETITION

Supersense Tagging Example



x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	
Clara	Harris	,	one	of	the	guests	in	the	box	,	stood	up	and	demanded	water	.	input tokens
N.PERSON						N.PERSON			N.ARTIFACT		V.MOTION			V.COMMUNICATION	N.SUBSTANCE		labeled spans
B-N.PERSON	I-N.PERSON	O	O	O	O	B-N.PERSON	O	O	B-N.ARTIFACT	O	B-V.MOTION	I-V.MOTION	O	B-V.COMMUNICATION	B-N.SUBSTANCE	O	BIO encoding
y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}	y_{16}	y_{17}	

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.

Big Abstraction: Linguistic Analysis

Every linguistic analyzer is comprised of:

1. Theoretical motivation from linguistics and/or the text domain
2. An algorithm that maps $\mathcal{V}^\dagger$ to some output space $\mathcal{Y}$.
   - Some $\mathcal{Y}$ are very specialized, but others, like the one we discuss here, show up again and again.
3. An implementation of the algorithm
   - Once upon a time: rule systems and crafted rules
   - More robust: supervised learning from annotated data
   - Today: unsupervised pretraining followed by supervised finetuning

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.,

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

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

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.,

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

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

Sometimes this works! E.g., one or two-layer neural network on top of contextual word vectors (which are features of the whole input $\boldsymbol{x}$).

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.,

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

$$\overset{\text{MLR}}{=} \operatorname*{argmax}_{y \in \mathcal{L}} \boldsymbol{\theta}^\top \mathbf{f}(\boldsymbol{x}, i, y)$$

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

Sometimes this works! E.g., one or two-layer neural network on top of contextual word vectors (which are features of the whole input $\boldsymbol{x}$).

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

Reflection

If we return to the original formulation,

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

how can we write "Score" in terms of the notation on the last slide?

Local Classifiers (v. 0)

Lightweight; no need to learn anything new! But labels can't affect each other.

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*: $\mathrm{score}(\boldsymbol{x}, i, \hat{\boldsymbol{y}}_{1:i-1}, y; \boldsymbol{\theta})$. (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.*

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

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.

Notes on Beam Search for Sequential Classifiers

- Runtime is $O(n^2kL)$, space is $O(n^2k)$.
- 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.
- Generally: no guarantee.

Reflection

Suppose your label set is built out of BIO tags. For an output $\hat{\boldsymbol{y}}$ to be well-formed, it suffices to ensure that it contains no "OI" label bigrams.

How would you modify beam search to guarantee well-formedness?

Sequential Classifiers (v. 1)

Very powerful! Algorithms lack guarantees.

A Generative Approach

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

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:

$$\begin{tikzcd} x_1 \\ y_1 \arrow[u] \end{tikzcd}$$

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

A Generative Approach

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

$$\begin{tikzcd}
x_1 & \\
y_1 \arrow[u] \arrow[r] & y_2
\end{tikzcd}$$

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

A Generative Approach

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

$$\begin{tikzcd}
x_1 & x_2 \\
y_1 \arrow[u] \arrow[r] & y_2 \arrow[u]
\end{tikzcd}$$

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

A Generative Approach

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

$$\begin{tikzcd}
x_1 & x_2 & \\
y_1 \arrow[u] \arrow[r] & y_2 \arrow[u] \arrow[r] & y_3
\end{tikzcd}$$

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

A Generative Approach

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

$$\begin{tikzcd}
x_1 & x_2 & x_3 \\
y_1 \arrow[u] \arrow[r] & y_2 \arrow[u] \arrow[r] & y_3 \arrow[u]
\end{tikzcd}$$

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

A Generative Approach

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

$$\begin{tikzcd}
x_1 & x_2 & x_3 & \\
y_1 \arrow[u] \arrow[r] & y_2 \arrow[u] \arrow[r] & y_3 \arrow[u] \arrow[r] & y_4
\end{tikzcd}$$

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

A Generative Approach

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

$$\begin{tikzcd}
x_1 & x_2 & x_3 & x_4 \\
y_1 \arrow[u] \arrow[r] & y_2 \arrow[u] \arrow[r] & y_3 \arrow[u] \arrow[r] & y_4 \arrow[u]
\end{tikzcd}$$

$$x_4 \sim p_{emission}(X \mid y_4)$$

A Generative Approach

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

$$\begin{tikzcd}
x_1 & & x_2 & & x_3 & & x_4 & & \\
y_1 \arrow[u] \arrow[r] & {} \arrow[r, phantom] & y_2 \arrow[u] \arrow[r] & {} \arrow[r, phantom] & y_3 \arrow[u] \arrow[r] & {} \arrow[r, phantom] & y_4 \arrow[u] \arrow[r] & {} \arrow[r, phantom] & \text{⯃}
\end{tikzcd}$$

$$y_5 \sim p_{transition}(Y \mid y_4)$$

Sequence Labeling v. 2: Hidden Markov Models

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

$$\begin{aligned}
p(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y}) &= p_{start}(y_1) \cdot \\
&\qquad \prod_{i=1}^{n} p_{emission}(x_i \mid y_i) \cdot p_{transition}(y_{i+1} \mid y_i) \\
\hat{\boldsymbol{y}} &= \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{L}^n} p(\boldsymbol{Y} = \boldsymbol{y} \mid \boldsymbol{X} = \boldsymbol{x}) \\
&= \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{L}^n} p(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y}) \\
&= \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{L}^n} \log p(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y})
\end{aligned}$$

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).

HMM Parameters

Classical HMM parameters are all interpretable as probabilities of events.

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.

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).

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.

Classical HMMs vs. Classifiers

With classifiers (local or sequential), the hard work is:

Classical HMMs vs. Classifiers

With classifiers (local or sequential), the hard work is:

- For humans: choosing features or designing a neural architecture that can learn good features

Classical HMMs vs. Classifiers

With classifiers (local or sequential), the hard work is:

- For humans: choosing features or designing a neural architecture that can learn good features
- For machines: estimating the parameters (typically by SGD); (in the sequential case) searching for "argmax"

Classical HMMs vs. Classifiers

With classifiers (local or sequential), the hard work is:

- For humans: choosing features or designing a neural architecture that can learn good features
- For machines: estimating the parameters (typically by SGD); (in the sequential case) searching for "argmax"

With classical HMMs, the parameters ($p_{transition}, p_{emission}, p_{start}$) have a closed form if you have labeled data! The hardest part is implementing the algorithm for choosing the "argmax" label sequence. Downside:

- You don't get to design or learn features.

Reflection

The runtime of the model-optimal decoding algorithm for HMMs depends quadratically on the size of $\mathcal{L}$. For some problems (e.g., supersense tagging) the label set can be large. Can you think of a way to trade the guarantee of model-optimality for speed, while still using the HMM?

Hidden Markov Models (v. 2)

Algorithmically beautiful; lack of features is unsatisfying.

Sequence Labeling v. 3

To endow HMMs with features, we can replace the "lookup" probabilities ($p_{transition}, p_{emission}, p_{start}$) with scoring functions.

This idea was explored by Berg-Kirkpatrick et al. (2010).

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):

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

Each "s" could be a linear scoring function (like in MLR), perhaps using word or label vectors. For now, I'm hiding the parameters of each s.

Notes on V. 3

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

Notes on V. 3

- Decoding is essentially the same as the HMM: Viterbi algorithm.
- Learning is now complicated and depends on the form of each "s," though I promise each iteration will be efficient. (Put this on my tab, along with Viterbi.)

Notes on V. 3

- Decoding is essentially the same as the HMM: Viterbi algorithm.
- Learning is now complicated and depends on the form of each "s," though I promise each iteration will be efficient. (Put this on my tab, along with Viterbi.)
- No part of the the scoring function looks at neighboring words.

V. 3

Brings features to HMMs, but learning is going to require more than just counting and normalizing.

Sequence Labeling v. 4

Let each scoring component ("s") "see" the whole input. By convention, $y_0 =$ ○ 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.

Notes on V. 4

- Decoding is essentially the same as the HMM and v. 3: Viterbi algorithm.

Notes on V. 4

- Decoding is essentially the same as the HMM and v. 3: Viterbi algorithm.
- As with v. 3, learning is complicated and depends on the form of each "s."

Notes on V. 4

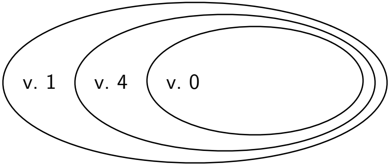
- Decoding is essentially the same as the HMM and v. 3: Viterbi algorithm.
- As with v. 3, learning is complicated and depends on the form of each “s.”
- This model strictly generalizes local classifiers (v. 0), the HMM (v. 2), and v. 3.

V. 4

Even better features for HMMs, with the promise of efficient decoding and learning.

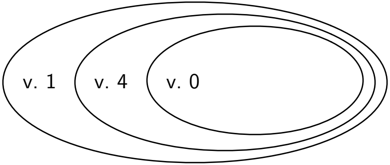
Reflection

Claim: As we move from v. 1 (sequential classifiers) to v. 4 to v. 0 (local classifiers), the scoring functions available become strictly less expressive.



Reflection

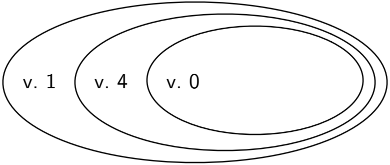
Claim: As we move from v. 1 (sequential classifiers) to v. 4 to v. 0 (local classifiers), the scoring functions available become strictly less expressive.



Compare v. 1 and v. 4. What kinds of features can you use in v. 1 that you can't use in v. 4?

Reflection

Claim: As we move from v. 1 (sequential classifiers) to v. 4 to v. 0 (local classifiers), the scoring functions available become strictly less expressive.



Compare v. 1 and v. 4. What kinds of features can you use in v. 1 that you can't use in v. 4?

Now consider v. 4 and v. 0. What kinds of features can you use in v. 4 that you can't use in v. 0?

Where We Are

	 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

The Main Dish

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	$\dots$	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	$\ldots$	x_n
ℓ_1				
ℓ_2				
$\vdots$				
ℓ_L				

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

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:

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

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:

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

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:

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

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:

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

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:

$$
\begin{aligned}
\heartsuit_{n+1}(\text{⯃}) &= \max_{y_n \in \mathcal{L}} s(\boldsymbol{x}, n, y_n, \text{⯃}) + \boxed{\heartsuit_n(y_n)} \\
\heartsuit_n(y) &= \max_{y_{n-1} \in \mathcal{L}} s(\boldsymbol{x}, n-1, y_{n-1}, y) + \boxed{\heartsuit_{n-1}(y_{n-1})} \\
\heartsuit_{n-1}(y) &= \max_{y_{n-2} \in \mathcal{L}} s(\boldsymbol{x}, n-2, y_{n-2}, y) + \boxed{\heartsuit_{n-2}(y_{n-2})} \\
&\vdots \\
\heartsuit_i(y) &= \max_{y_{i-1} \in \mathcal{L}} s(\boldsymbol{x}, i-1, y_{i-1}, y) + \boxed{\heartsuit_{i-1}(y_{i-1})} \\
&\vdots \\
\heartsuit_1(y) &= s(\boldsymbol{x}, 0, \bigcirc, y)
\end{aligned}
$$

Viterbi Procedure (Part I: Prefix Scores)

input sequence

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

Viterbi Procedure (Part I: Prefix Scores)

input sequence

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

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

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

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

Viterbi Procedure (Part I: Prefix Scores and Backpointers)

input sequence

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

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

$$\mathrm{bp}_1(y) = \bigcirc$$

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

		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	$\heartsuit_1(\ell_L)$ $\text{bp}_1(\ell_L)$	$\heartsuit_2(\ell_L)$ $\text{bp}_2(\ell_L)$		$\heartsuit_n(\ell_L)$ $\text{bp}_n(\ell_L)$	
	🛑					

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

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

Viterbi Procedure (Part I: Prefix Scores and Backpointers)

input sequence

		x_1	x_2	$\dots$	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	$\heartsuit_1(\ell_2)$ $\mathrm{bp}_1(\ell_2)$	$\heartsuit_2(\ell_2)$ $\mathrm{bp}_2(\ell_2)$		$\heartsuit_n(\ell_2)$ $\mathrm{bp}_n(\ell_2)$	
$\mathcal{L}$	$\vdots$					
	ℓ_L	$\heartsuit_1(\ell_L)$ $\mathrm{bp}_1(\ell_L)$	$\heartsuit_2(\ell_L)$ $\mathrm{bp}_2(\ell_L)$		$\heartsuit_n(\ell_L)$ $\mathrm{bp}_n(\ell_L)$	
	⯃					$\heartsuit_{n+1}(\text{⯃})$ $\mathrm{bp}_{n+1}(\text{⯃})$

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

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

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 $y =$ 🛑.)

3. $\hat{y}_{i+1} \leftarrow$ 🛑
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	$\dots$	x_n
ℓ_1				
ℓ_2				
$\vdots$				
ℓ_L				

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.

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

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."

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

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.

$$
\begin{aligned}
p_{\mathrm{CRF}}(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta}) &= \frac{\exp \mathrm{Score}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})}{Z(\boldsymbol{x}; \boldsymbol{\theta})} \\
Z(\boldsymbol{x}; \boldsymbol{\theta}) &= \sum_{\boldsymbol{y'} \in \mathcal{Y}(\boldsymbol{x})} \exp \mathrm{Score}(\boldsymbol{x}, \boldsymbol{y'}; \boldsymbol{\theta}) \\
-\log p_{\mathrm{CRF}}(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta}) &= -\underbrace{\mathrm{Score}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})}_{\text{“hope”}} + \underbrace{\log Z(\boldsymbol{x}; \boldsymbol{\theta})}_{\text{“fear”}}
\end{aligned}
$$

So, our “CRF”:

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

Conditional Random Field

Lafferty et al. (2001)

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 $i \in \{0, \dots, n\}$, $y, y' \in \mathcal{L}$

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

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

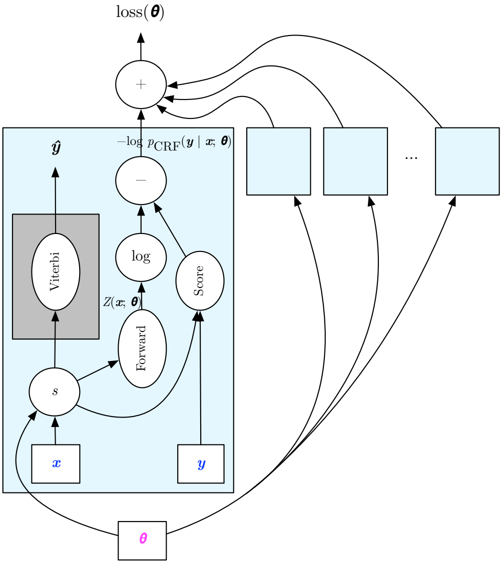
Intuitions about the Forward Algorithm

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.

Computation Graph View of CRF



$\text{loss}(\boldsymbol{\theta})$
$+$
$-\log p_{\text{CRF}}(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$
$\hat{\boldsymbol{y}}$
$-$
log
Viterbi
Score
$Z(\boldsymbol{x}; \boldsymbol{\theta})$
Forward
s
$\boldsymbol{x}$
$\boldsymbol{y}$
...
$\boldsymbol{\theta}$

Reflection

Earlier in the lecture, I promised that learning would have some guarantees. Consider:

- The runtime and space requirements for calculating the loss and gradient, as a function of the data.
- The conditions under which we can confidently expect convergence to a global optimum of the likelihood if we use SGD.

An Alternative: Structured Perceptron

Recall that CRF = v. 4 + sequence-level log loss.

Perceptron loss (Collins, 2002):

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

The structured perceptron = v. 4 + perceptron loss.

Regularization

Just as in classification with linear and non-linear models, you'll want to take steps to avoid overfitting.

The same tools (e.g., ℓ_2 and ℓ_1 penalties for linear model weights, and dropout for neural networks) can be used here.

Digestif: Connections and Generalizations

V. 2–4 are weighted finite-state machines (think of labels as states).

The models we saw today are all "first order" sequence models in the sense that each y_i only interacts with one immediate neighbor through s.

- Second-order: $\text{Score}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=0}^{n} s(\boldsymbol{x}, i, y_i, y_{i+1}, y_{i+2})$
- mth-order: $\text{Score}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=0}^{n} s(\boldsymbol{x}, i, \boldsymbol{y}_{i:i+m})$

Viterbi for mth order has $O(nL^{m+1})$ runtime.

References I

Taylor Berg-Kirkpatrick, Alexandre Bouchard-Côté, John DeNero, and Dan Klein. Painless unsupervised learning with features. In *Proc. of NAACL*, 2010.

Daniel M. Bikel, Richard Schwartz, and Ralph M. Weischedel. An algorithm that learns what's in a name. *Machine learning*, 34(1–3):211–231, 1999.

Kenneth W. Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proc. of ANLP*, 1988.

Massimiliano Ciaramita and Yasemin Altun. Broad-coverage sense disambiguation and information extraction with a supersense sequence tagger. In *Proc. of EMNLP*, 2006.

Massimiliano Ciaramita and Mark Johnson. Supersense tagging of unknown nouns in WordNet. In *Proc. of EMNLP*, 2003.

Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proc. of EMNLP*, 2002.

John M. Conroy and Dianne P. O'Leary. Text summarization via hidden Markov models. In *Proc. of SIGIR*, 2001.

Jacob Eisenstein. *Introduction to Natural Language Processing*. MIT Press, 2019.

Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

Nizar Habash and Owen Rambow. Arabic tokenization, part-of-speech tagging and morphological disambiguation in one fell swoop. In *Proc. of ACL*, 2005.

References II

Mark D. Kernighan, Kenneth W. Church, and William A. Gale. A spelling correction program based on a noisy channel model. In *Proc. of COLING*, 1990.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML*, 2001.

G. A. Miller, C. Leacock, T. Randee, and R. Bunker. A semantic concordance. In *Proc. of HLT*, 1993.

Lance A Ramshaw and Mitchell P. Marcus. Text chunking using transformation-based learning, 1995. URL `http://arxiv.org/pdf/cmp-lg/9505040.pdf`.

Nathan Schneider, Spencer Onuffer, Nora Kazour, Emily Danchik, Michael T. Mordowanec, Henrietta Conrad, and Noah A. Smith. Comprehensive annotation of multiword expressions in a social web corpus. In *Proc. of LREC*, 2014.

Thamar Solorio and Yang Liu. Learning to predict code-switching points. In *Proc. of EMNLP*, 2008.

Richard W. Sproat, Chilin Shih, William Gale, and Nancy Chang. A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics*, 22(3):377–404, 1996. URL `https://www.aclweb.org/anthology/J96-3004`.

Andreas Stolcke, Klaus Ries, Noah Coccaro, Elizabeth Shriberg, Rebecca Bates, Daniel Jurafsky, Paul Taylor, Rachel Martin, Carol Van Ess-Dykema, and Marie Meteer. Dialogue act modeling for automatic tagging and recognition of conversational speech. *Computational Linguistics*, 26(3):339–374, 2000. URL `https://www.aclweb.org/anthology/J00-3003`.

References III

Stephan Vogel, Hermann Ney, and Christoph Tillmann. HMM-based word alignment in statistical translation. In *Proc. of COLING*, 1996.