Loss-augmented Structured Prediction

CMSC 723 / LING 723 / INST 725

Marine Carpuat

Figures, algorithms & equations from CIML chap 17
POS tagging
Sequence labeling with the perceptron

Sequence labeling problem
• Input:
  • sequence of tokens $x = [x_1 ... x_L]$
  • Variable length $L$

• Output (aka label):
  • sequence of tags $y = [y_1 ... y_L]$
  • # tags = $K$
  • Size of output space?

Structured Perceptron
• Perceptron algorithm can be used for sequence labeling

• But there are challenges
  • How to compute $\text{argmax}$ efficiently?
  • What are appropriate features?

• Approach: leverage structure of output space
Solving the argmax problem for sequences with dynamic programming

\[ x = "\text{monsters eat tasty bunnies}" \]
\[ y = \text{noun verb adj noun} \]

- Efficient algorithms possible if the feature function decomposes over the input
- This holds for unary and markov features used for POS tagging
Feature functions for sequence labeling

\[ x = "\text{monsters eat tasty bunnies}" \]
\[ y = \text{noun verb adj noun} \]

- **Standard features of POS tagging**
  - **Unary features:** \# times word \( w \) has been labeled with tag \( l \) for all words \( w \) and all tags \( l \)
  - **Markov features:** \# times tag \( l \) is adjacent to tag \( l' \) in output for all tags \( l \) and \( l' \)

- Size of feature representation is constant wrt input length
Solving the argmax problem for sequences

- Trellis sequence labeling
  - Any path represents a labeling of input sentence
  - Gold standard path in red
  - Each edge receives a weight such that adding weights along the path corresponds to score for input/output configuration

- Any max-weight max-weight path algorithm can find the argmax
  - e.g. Viterbi algorithm $O(LK^2)$
Defining weights of edge in trellis

\[ w \cdot \phi(x, y) = w \cdot \sum_{l=1}^{L} \phi_l(x, y) \]

 decomposition of structure \quad (17.35)

\[ = \sum_{l=1}^{L} w \cdot \phi_l(x, y) \]

 associative law \quad (17.36)

- Weight of edge that goes from time \(l-1\) to time \(l\), and transitions from \(y\) to \(y'\)

\[ w \cdot \phi_l(x, \cdots \circ y \circ y') \]
Dynamic program

- Define: the score of best possible output prefix up to and including position $l$ that labels the $l$-th word with label $k$

$$\alpha_{l,k} = \max_{\hat{y}_{1:l-1}} w \cdot \phi_{1:l}(x, \hat{y} \circ k)$$

- With decomposable features, alphas can be computed recursively

$$\alpha_{l+1,k} = \max_{k'} \left[ \alpha_{l,k'} + w \cdot \phi_{l+1}(x, \langle \ldots, k', k \rangle) \right]$$
Algorithm 42 \textbf{ArgmaxForSequences}(x, w)

1: \( L \leftarrow \text{len}(x) \)
2: \( \alpha_{l,k} \leftarrow 0, \quad \zeta_{k,l} \leftarrow 0, \quad \forall k = 1 \ldots K, \quad \forall l = 0 \ldots L \) \hspace{1cm} // initialize variables
3: \textbf{for} \( l = 0 \ldots L-1 \) \textbf{do}
4: \hspace{1cm} \textbf{for} \( k = 1 \ldots K \) \textbf{do}
5: \hspace{2cm} \( \alpha_{l+1,k} \leftarrow \max_{k'} \left[ \alpha_{l,k'} + w \cdot \phi_{l+1}(x, \ldots, k', k) \right] \) \hspace{1cm} // recursion:
6: \hspace{3cm} \text{here, } \phi_{l+1}(\ldots k', k \ldots) \text{ is the set of features associated with}
7: \hspace{3cm} \text{output position } l + 1 \text{ and two adjacent labels } k' \text{ and } k \text{ at that position}
8: \hspace{1cm} \zeta_{l+1,k} \leftarrow \text{the } k' \text{ that achieves the maximum above} \hspace{1cm} // \text{store backpointer}
9: \hspace{1cm} \textbf{end for}
10: \textbf{end for}
11: y \leftarrow \langle 0, 0, \ldots, 0 \rangle \hspace{1cm} \text{// initialize predicted output to } L\text{-many zeros}
12: y_L \leftarrow \text{argmax}_k \alpha_{L,k} \hspace{1cm} \text{// extract highest scoring final label}
13: \textbf{for} \( l = L-1 \ldots 1 \) \textbf{do}
14: \hspace{1cm} y_l \leftarrow \zeta_{l,y_{l+1}} \hspace{1cm} \text{// traceback } \zeta \text{ based on } y_{l+1}
15: \textbf{end for}
16: \textbf{return} \ y \hspace{1cm} \text{// return predicted output}
A more general approach for argmax

Integer Linear Programming

• ILP: optimization problem of the form, for a fixed vector $a$

$$\max_{z} \ a \cdot z \ \text{subj. to linear constraints on } z$$

  • With integer constraints

• Pro: can leverage well-engineered solvers (e.g., Gurobi)

• Con: not always most efficient
POS tagging as ILP

- Markov features as binary indicator variables
  \[ z_{l,k',k} = 1 \text{[label } l \text{ is } k \text{ and label } l-1 \text{ is } k'] \]

- Output sequence: \( y(z) \) obtained by reading off variables \( z \)

- Define \( a_z \) such that \( a_z \) is equal to score
  \[ a_{l,k',k} = w \cdot \phi_l(x, \langle \ldots, k', k \rangle) \]

- Enforcing constraints for well formed solutions
  1. That all the \( z \)s are binary. That’s easy: just say \( z_{l,k',k} \in \{0,1\} \), for all \( l,k',k \).
  2. That for a given position \( l \), there is exactly one active \( z \). We can do this with an equality constraint: \( \Sigma_k \Sigma_{k'} z_{l,k',k} = 1 \) for all \( l \).
  3. That the \( z \)s are internally consistent: if the label at position 5 is supposed to be “noun” then both \( z_{5,n} \) and \( z_{6,v} \) need to agree on this. We can do this as: \( \Sigma_{k''} z_{l,k',k} = \Sigma_{k''} z_{l+1,k,k''} \) for all \( l,k \). Effectively what this is saying is that \( z_{5,?_verb} = z_{6,verb} \) where the “?” means “sum over all possibilities.”
Sequence labeling

• Structured perceptron
  • A general algorithm for structured prediction problems such as sequence labeling

• The Argmax problem
  • Efficient argmax for sequences with Viterbi algorithm, given some assumptions on feature structure
  • A more general solution: Integer Linear Programming

• Loss-augmented structured prediction
  • Training algorithm
  • Loss-augmented argmax
In structured perceptron, all errors are equally bad.
All bad output sequences are not equally bad

- Consider $y^\# = [A, A, A, A]$,
- $\hat{y}^\# = [N, V, N, N]$

- Hamming Loss

- Gives a more nuanced evaluation of output than 0–1 loss

$$\ell^{(\text{Ham})}(y, \hat{y}) = \sum_{l=1}^{L} 1[y_l \neq \hat{y}_l]$$
Loss functions for structured prediction

• Recall learning as optimization for classification
  
  \[ \min_w \frac{1}{2} \|w\|^2 + C \sum_n \ell^{(\text{hin})}(y_n, w \cdot x_n + b) \]

• Let’s define a structure-aware optimization objective
  
  \[ \min_w \frac{1}{2} \|w\|^2 + C \sum_n \ell^{(s-h)}(y_n, x_n, w) \]

  \[ \ell^{(s-h)}(y_n, x_n, w) = \max \left\{ 0, \max_{\hat{y} \in \mathcal{Y}(x_n)} \left[ s_w(x_n, \hat{y}) + \ell^{(\text{Ham})}(y_n, \hat{y}) \right] - s_w(x_n, y_n) \right\} \]

Structured hinge loss
  • 0 if true output beats score of every imposter output
  • Otherwise: scales linearly as function of score diff between most confusing imposter and true output
Optimization: stochastic subgradient descent

• Subgradients of structured hinge loss?

\[
\nabla_w \ell^{(s-h)}(y, x, w) \quad \text{if the loss is } > 0
\]

\[
= \nabla_w \left\{ \max_{\hat{y} \in \mathcal{Y}(x_n)} \left[ w \cdot \phi(x_n, \hat{y}) + \ell(y_n, \hat{y}) \right] - w \cdot \phi(x_n, y_n) \right\} 
\]

\[
(17.25)
\]

expand definition using arbitrary structured loss \( \ell \)

\[
= \nabla_w \left\{ \max_{\hat{y} \in \mathcal{Y}(x_n)} \left[ w \cdot \phi(x_n, \hat{y}) + \ell(y_n, \hat{y}) \right] - w \cdot \phi(x_n, y_n) \right\} 
\]

\[
(17.26)
\]

define \( \hat{y}_n \) to be the output that attains the maximum above, rearrange

\[
= \nabla_w \left\{ w \cdot \phi(x_n, \hat{y}) - w \cdot \phi(x_n, y_n) + \ell(y_n, \hat{y}) \right\} 
\]

\[
(17.27)
\]
take gradient

\[
= \phi(x_n, \hat{y}) - \phi(x_n, y_n)
\]

\[
(17.28)
\]
Optimization: stochastic subgradient descent

- subgradients of structured hinge loss

\[
\nabla_w \ell^{(s-h)}(y_n, x_n, w) = \begin{cases} 
0 & \text{if } \ell^{(s-h)}(y_n, x_n, w) = 0 \\
\phi(x_n, \hat{y}_n) - \phi(x_n, y_n) & \text{otherwise}
\end{cases}
\]

where \( \hat{y}_n = \arg\max_{\hat{y}_n \in \mathcal{Y}(x_n)} \left[ w \cdot \phi(x_n, \hat{y}_n) + \ell(y_n, \hat{y}_n) \right] \) (17.29)
Optimization: stochastic subgradient descent
Resulting training algorithm

**Algorithm 41** \texttt{StochSubGradStructSVM}(D, MaxIter, \lambda, \ell)

1: \texttt{w} \leftarrow 0 \quad // initialize weights
2: \texttt{for iter = 1 \ldots MaxIter do}
3: \quad \texttt{for all} (x,y) \in D \texttt{do}
4: \quad \hat{y} \leftarrow \arg\max_{\hat{y} \in \mathcal{Y}(x)} w \cdot \phi(x, \hat{y}) + \ell(y, \hat{y}) \quad // loss-augmented prediction
5: \quad \texttt{if} \hat{y} \neq y \texttt{then}
6: \quad \quad w \leftarrow w + \phi(x, y) - \phi(x, \hat{y}) \quad // update weights
7: \quad \texttt{end if}
8: \quad w \leftarrow w - \frac{\lambda}{N} w \quad // shrink weights due to regularizer
9: \texttt{end for}
10: \texttt{end for}
11: \texttt{return} w \quad // return learned weights

Only 2 differences compared to structured perceptron!
Loss-augmented inference/search
Recall dynamic programming solution without Hamming loss

$$\tilde{\alpha}_{l+1,k} = \max_{\hat{y}_{1:l}} \omega \cdot \phi_{1:l+1}(x, \hat{y} \circ k)$$

$$= \max_{k'} \left[ \tilde{\alpha}_{l,k'} + \omega \cdot \phi_{l+1}(x, \langle \ldots, k', k \rangle) \right]$$
Loss-augmented inference/search
Dynamic programming with Hamming loss

\[
\tilde{\alpha}_{l+1, k} = \max_{\hat{y}_{1:l}} \omega \cdot \phi_{1:l+1}(x, \hat{y} \circ k) + \ell_{1:l+1}^{(\text{Ham})}(y, \hat{y} \circ k) \\
= \max_{k'} \left[ \tilde{\alpha}_{l, k'} + \omega \cdot \phi_{l+1}(x, \langle \ldots, k', k \rangle) \right] + 1[k \neq y_{l+1}]
\]

We can use Viterbi algorithm as before as long as the loss function decomposes over the input consistently w features!
Sequence labeling

• Structured perceptron
  • A general algorithm for structured prediction problems such as sequence labeling

• The Argmax problem
  • Efficient argmax for sequences with Viterbi algorithm, given some assumptions on feature structure
  • A more general solution: Integer Linear Programming

• Loss-augmented structured prediction
  • Training algorithm
  • Loss-augmented argmax
Syntax & Grammars
From Sequences to Trees
This is a simple sentence. But it is an instructive one.
Syntax & Grammar

• Syntax
  • From Greek syntaxis, meaning “setting out together”
  • refers to the way words are arranged together.

• Grammar
  • Set of structural rules governing composition of clauses, phrases, and words in any given natural language
  • Descriptive, not prescriptive
  • Panini’s grammar of Sanskrit ~2000 years ago
Syntax and Grammar

• Goal of syntactic theory
  • “explain how people combine words to form sentences and how children attain knowledge of sentence structure”

• Grammar
  • implicit knowledge of a native speaker
  • acquired without explicit instruction
  • minimally able to generate all and only the possible sentences of the language

[Philips, 2003]
Syntax in NLP

• Syntactic analysis often a key component in applications
  • Grammar checkers
  • Dialogue systems
  • Question answering
  • Information extraction
  • Machine translation
  • ...

Two views of syntactic structure

• Constituency (phrase structure)
  • Phrase structure organizes words in nested constituents

• Dependency structure
  • Shows which words depend on (modify or are arguments of) which on other words
Constituency

• Basic idea: groups of words act as a single unit

• Constituents form coherent classes that behave similarly
  • With respect to their internal structure: e.g., at the core of a noun phrase is a noun
  • With respect to other constituents: e.g., noun phrases generally occur before verbs
Constituency: Example

• The following are all noun phrases in English...

<table>
<thead>
<tr>
<th>Harry the Horse</th>
<th>a high-class spot such as Mindy’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>the Broadway coppers</td>
<td>the reason he comes into the Hot Box</td>
</tr>
<tr>
<td>they</td>
<td>three parties from Brooklyn</td>
</tr>
</tbody>
</table>

• Why?
  • They can all precede verbs
  • They can all be preposed/postposed
  • ...
Grammars and Constituency

• For a particular language:
  • What are the “right” set of constituents?
  • What rules govern how they combine?

• Answer: not obvious and difficult
  • That’s why there are many different theories of grammar and competing analyses of the same data!

• Our approach
  • Focus primarily on the “machinery”
Context-Free Grammars

• Context-free grammars (CFGs)
  • Aka phrase structure grammars
  • Aka Backus-Naur form (BNF)

• Consist of
  • Rules
  • Terminals
  • Non-terminals
Context-Free Grammars

• Terminals
  • We’ll take these to be words

• Non-Terminals
  • The constituents in a language (e.g., noun phrase)

• Rules
  • Consist of a single non-terminal on the left and any number of terminals and non-terminals on the right
# An Example Grammar

<table>
<thead>
<tr>
<th>Grammar Rules</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP \ VP$</td>
<td>I + want a morning flight</td>
</tr>
<tr>
<td>$NP \rightarrow$ Pronoun</td>
<td>I</td>
</tr>
<tr>
<td>Proper-Noun</td>
<td>Los Angeles</td>
</tr>
<tr>
<td>Det Nominal</td>
<td>a + flight</td>
</tr>
<tr>
<td>Nominal $\rightarrow$ Nominal Noun</td>
<td>morning + flight</td>
</tr>
<tr>
<td>Noun</td>
<td>flights</td>
</tr>
<tr>
<td>$VP \rightarrow$ Verb</td>
<td>do</td>
</tr>
<tr>
<td>Verb NP</td>
<td>want + a flight</td>
</tr>
<tr>
<td>Verb NP PP</td>
<td>leave + Boston + in the morning</td>
</tr>
<tr>
<td>Verb PP</td>
<td>leaving + on Thursday</td>
</tr>
<tr>
<td>$PP \rightarrow$ Preposition NP</td>
<td>from + Los Angeles</td>
</tr>
</tbody>
</table>
Parse Tree: Example

S
  NP
    Pro
    I
  VP
    Verb
    prefer
  NP
    Det
    a
    Nom
    Noun
    flight
  Noun
  morning
Dependency Grammars

• CFGs focus on constituents
  • Non-terminals don’t actually appear in the sentence

• In dependency grammar, a parse is a graph (usually a tree) where:
  • Nodes represent words
  • Edges represent dependency relations between words
    (typed or untyped, directed or undirected)
Dependency Grammars

• Syntactic structure = lexical items linked by binary asymmetrical relations called dependencies

Diagram:

- Dependency Type
- Head
- Dependent (modifier / object / compliment)
## Dependency Relations

<table>
<thead>
<tr>
<th>Argument Dependencies</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsubj</td>
<td>nominal subject</td>
</tr>
<tr>
<td>csubj</td>
<td>clausal subject</td>
</tr>
<tr>
<td>dobj</td>
<td>direct object</td>
</tr>
<tr>
<td>iobj</td>
<td>indirect object</td>
</tr>
<tr>
<td>pobj</td>
<td>object of preposition</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modifier Dependencies</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmod</td>
<td>temporal modifier</td>
</tr>
<tr>
<td>appos</td>
<td>appositional modifier</td>
</tr>
<tr>
<td>det</td>
<td>determiner</td>
</tr>
<tr>
<td>prep</td>
<td>prepositional modifier</td>
</tr>
</tbody>
</table>
They hid the letter on the shelf

Compare with constituent parse…

What's the relation?
<table>
<thead>
<tr>
<th>Relation</th>
<th>Examples with head and dependent</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSUBJ</td>
<td>United canceled the flight.</td>
</tr>
<tr>
<td>DOBJ</td>
<td>United diverted the flight to Reno.</td>
</tr>
<tr>
<td></td>
<td>We booked her the first flight to Miami.</td>
</tr>
<tr>
<td>IOBJ</td>
<td>We booked her the flight to Miami.</td>
</tr>
<tr>
<td>NMOD</td>
<td>We took the morning flight.</td>
</tr>
<tr>
<td>AMOD</td>
<td>Book the cheapest flight.</td>
</tr>
<tr>
<td>NUMMOD</td>
<td>Before the storm JetBlue canceled 1000 flights.</td>
</tr>
<tr>
<td>APPOS</td>
<td>United, a unit of UAL, matched the fares.</td>
</tr>
<tr>
<td>DET</td>
<td>The flight was canceled.</td>
</tr>
<tr>
<td></td>
<td>Which flight was delayed?</td>
</tr>
<tr>
<td>CONJ</td>
<td>We flew to Denver and drove to Steamboat.</td>
</tr>
<tr>
<td>CC</td>
<td>We flew to Denver and drove to Steamboat.</td>
</tr>
<tr>
<td>CASE</td>
<td>Book the flight through Houston.</td>
</tr>
</tbody>
</table>

**Figure 14.3** Examples of core Universal Dependency relations.
Universal Dependencies project

• Set of dependency relations that are
  • Linguistically motivated
  • Computationally useful
  • Cross-linguistically applicable
  • [Nivre et al. 2016]

• Universaldependencies.org
Summary

• Syntax & Grammar

• Two views of syntactic structures
  • Context-Free Grammars
  • Dependency grammars
  • Can be used to capture various facts about the structure of language (but not all!)

• Treebanks as an important resource for NLP