NLP 202: Graph-Based Dependency Parsing

Jeffrey Flanigan

Winter 2023

University of California Santa Cruz
jmflanig@ucsc.edu

Plan for Today

- Perceptron with non-linear scoring function
- Perceptron for sequence labeling
- Perceptron for graph-based dependency parsing
- Relation between Perceptron and logistic regression/CRFs
- CRFs for dependency parsing

CKY Example

Quick Recap: Perceptron Algorithm for Structured Outputs

Structured Perceptron (linear scoring function)

Given a training set $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$ where $\mathbf{x}_i \in \mathcal{X}$, $\mathbf{y}_i \in \mathcal{Y}(\mathbf{x}_i)$ and linear scoring function $\mathbf{w}^T\mathbf{f}(\mathbf{x}_i, \mathbf{y})$

1. Initialize $\mathbf{w} = \mathbf{0} \in R^n$
2. For epoch in $1 \dots T$:
    1. Shuffle the data
    2. For each training example $(\mathbf{x}_i, y_i) \in D$:
        - Make prediction $\hat{\mathbf{y}} = \mathrm{argmax}_{\mathbf{y}}\, \mathbf{w}^T\mathbf{f}(\mathbf{x}_i, \mathbf{y})$
        - Update $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(\mathbf{x}_i, \mathbf{y}_i) - \mathbf{f}(\mathbf{x}_i, \hat{\mathbf{y}})$
3. Return $\mathbf{w}$

Prediction on a new example $\mathbf{x}$: $\hat{\mathbf{y}} = \mathrm{argmax}_{\mathbf{y}}\, \mathbf{w}^T\mathbf{f}(\mathbf{x}, \mathbf{y})$

Structured Perceptron (non-linear scoring function)

Given a training set $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$ where $\mathbf{x}_i \in \mathcal{X}$, $\mathbf{y}_i \in \mathcal{Y}(\mathbf{x}_i)$ and scoring function $score_\theta(\mathbf{x}, \mathbf{y})$

1. Initialize $\theta = \mathbf{0} \in R^n$
2. For epoch in $1 \dots T$:
    1. Shuffle the data
    2. For each training example $(\mathbf{x}_i, \mathbf{y}_i) \in D$:
        - Make prediction $\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}} score_\theta(\mathbf{x}_i, \mathbf{y})$
        - Update

$$\theta \leftarrow \theta + \frac{\partial score_\theta(\mathbf{x}_i, \mathbf{y}_i)}{\partial \theta} - \frac{\partial score_\theta(\mathbf{x}_i, \hat{\mathbf{y}})}{\partial \theta}$$

3. Return θ

Prediction on a new example $\mathbf{x}$: $\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}} score_\theta(\mathbf{x}, \mathbf{y})$

Learning the Parameters

- Perceptron loss with linear scoring function:

$$\min_{\mathbf{w}} \sum_{i=1}^{N} \left[\left(\max_{y' \in \mathcal{Y}} \mathbf{w} \cdot f(\boldsymbol{x}_i, \boldsymbol{y}') \right) - \mathbf{w} \cdot f(\boldsymbol{x}_i, \boldsymbol{y}_i) \right]$$

- Perceptron loss with non-linear scoring function:

$$\min_{\theta} \sum_{i=1}^{N} \left[\left(\max_{y' \in \mathcal{Y}} score_{\theta}(\boldsymbol{x}_i, \boldsymbol{y}') \right) - score_{\theta}(\boldsymbol{x}_i, \boldsymbol{y}_i) \right]$$

Learning the Parameters

- Perceptron loss with linear scoring function:

$$\min_{\mathbf{w}} \sum_{i=1}^{N} \left[\left(\max_{y' \in \mathcal{Y}} \mathbf{w} \cdot f(\boldsymbol{x}_i, \boldsymbol{y}') \right) - \mathbf{w} \cdot f(\boldsymbol{x}_i, \boldsymbol{y}_i) \right]$$

- Perceptron loss with non-linear scoring function:

$$\min_{\theta} \sum_{i=1}^{N} \left[\left(\max_{y' \in \mathcal{Y}} score_{\theta}(\boldsymbol{x}_i, \boldsymbol{y}') \right) - score_{\theta}(\boldsymbol{x}_i, \boldsymbol{y}_i) \right]$$

- Standard Perceptron algorithm uses Stochastic Sub-Gradient Descent (SSGD) to minimize.
  But you can use any optimizer you want (Adagrad, Adam, etc).

Learning the Parameters

Perceptron solves the following minization problem:

$$\min_{\theta} \sum_{i=1}^{N} \left[\left(\max_{y' \in \mathcal{Y}} score_{\theta}(\boldsymbol{x}_i, \boldsymbol{y}') \right) - score_{\theta}(\boldsymbol{x}_i, \boldsymbol{y}_i) \right]$$

Stochastic subgradient descent (SSGD) with stepsize α:

- For $i \in \{1, \ldots, N\}$:
  - Shuffle the training data, and for each example i do the following update:
    - Make prediction $\hat{\boldsymbol{y}} = \operatorname{argmax}_{\boldsymbol{y}} score_{\theta}(\boldsymbol{x}_i, \boldsymbol{y})$
    - Update

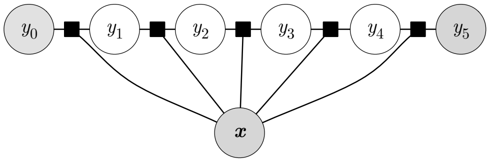
$$\theta \leftarrow \theta + \alpha \left(\frac{\partial score_{\theta}(\boldsymbol{x}_i, \boldsymbol{y}_i)}{\partial \theta} - \frac{\partial score_{\theta}(\boldsymbol{x}_i, \hat{\boldsymbol{y}})}{\partial \theta} \right)$$

Example: Sequence labeling with Perceptron Algorithm (Collins, 1999)

- Loss function = Perceptron loss

$$L(\boldsymbol{x}, \boldsymbol{y}, \theta) = -score_\theta(\boldsymbol{x}, \boldsymbol{y}) + \max_{\boldsymbol{y}' \in \mathcal{Y}} score_\theta(\boldsymbol{x}, \boldsymbol{y}')$$

- $score_\theta(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=0}^{n} s_\theta(\boldsymbol{x}, i, y_i, y_{i+1})$
- Decoding algorithm = Viterbi algorithm (from last quarter)
- Optimizer = stochastic subgradient descent (SSGD)



Last Quarter: Viterbi with arbitrary scoring functions

- Viterbi algorithm can find the exact argmax

$$\hat{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{y}} \sum_{i=1}^{n} s_\theta(\mathbf{x}, i, y_i, y_{i-1})$$

- Works for any scoring function and any semiring
- We can use the Viterbi algorithm!

Sequence Labeling with Linear Scoring Function

$$
\begin{aligned}
score_\theta(\boldsymbol{x}, \boldsymbol{y}) &= \sum_{i=0}^{n} s_\theta(\boldsymbol{x}, i, y_i, y_{i+1}) \\
&= \sum_{i=0}^{n} \theta \cdot f(\boldsymbol{x}, i, y_i, y_{i+1}) \\
&= \theta \cdot \sum_{i=0}^{n} f(\boldsymbol{x}, i, y_i, y_{i+1}) \\
&= \theta \cdot F(\boldsymbol{x}, \boldsymbol{y})
\end{aligned}
$$

$f(\boldsymbol{x}, i, y_i, y_{i+1})$ are **local features**
$F(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=0}^{n} f(\boldsymbol{x}, i, y_i, y_{i+1})$ is the total feature vector

Structured Perceptron Algorithm (Collins, 1999)

Perceptron loss:

$$L(\mathbf{w}, \mathcal{D}) = \sum_{i=1}^{N} \max_{\boldsymbol{y}' \in \mathcal{Y}} \mathbf{w} \cdot f(\boldsymbol{x}_i, \boldsymbol{y}') - \mathbf{w} \cdot f(\boldsymbol{x}_i, \boldsymbol{y}_i)$$

Learning algorithm (stochastic subgradient descent, SSGD):

- For T epochs (passes through the training data):
    - Shuffle the training data, and for each example $(\boldsymbol{x}, \boldsymbol{y})$ do the following update:
        - $\hat{\boldsymbol{y}} \leftarrow \mathrm{argmax}_{\boldsymbol{y}' \in \mathcal{Y}} \mathbf{w} \cdot F(\boldsymbol{x}, \boldsymbol{y}')$
        - $\mathbf{w} \leftarrow \mathbf{w} - \alpha \; g(\boldsymbol{x}, \boldsymbol{y})$
          where the gradient $g(\boldsymbol{x}, \boldsymbol{y}) = F(\boldsymbol{x}, \hat{\boldsymbol{y}}) - F(\boldsymbol{x}, \boldsymbol{y})$
          and α is the stepsize (usually set to 1 for linear models with perceptron loss)

Features

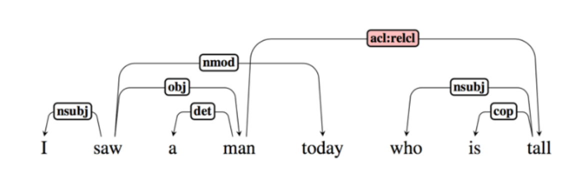
	will $\phi(x, 1, y_1, y_0)$	to $\phi(x, 2, y_2, y_1)$	fight $\phi(x, 3, y_3, y_2)$	$\mathbf{\Phi}$(x, NN TO VB)
x_i=will ^ y_i = NN	1	0	0	1
y_{i-1}=START ^ y_i = NN	1	0	0	1
x_i=will ^ y_i = MD	0	0	0	0
y_{i-1}=START ^ y_i = MD	0	0	0	0
…				
x_i=to ^ y_i = TO	0	1	0	1
y_{i-1}=NN ^ y_i = TO	0	1	0	1
y_{i-1}=MD ^ y_i = TO	0	0	0	0
…				
x_i=fight ^ y_i = VB	0	0	1	1
y_{i-1}=TO ^ y_i = VB	0	0	1	1

Can also use a neural scoring function instead.

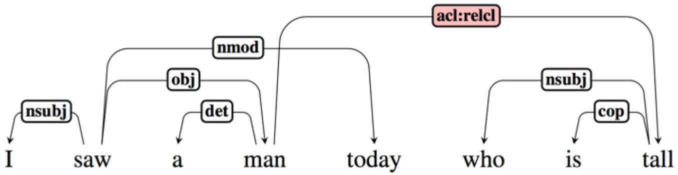
Application to Dependency Parsing

1. Transition-based parsing with a stack.
2. **Chu-Liu-Edmonds algorithm for arborescences (directed trees).**
3. Dynamic programming with the Eisner algorithm (next week).

Graph-based methods allow non-projective parsing



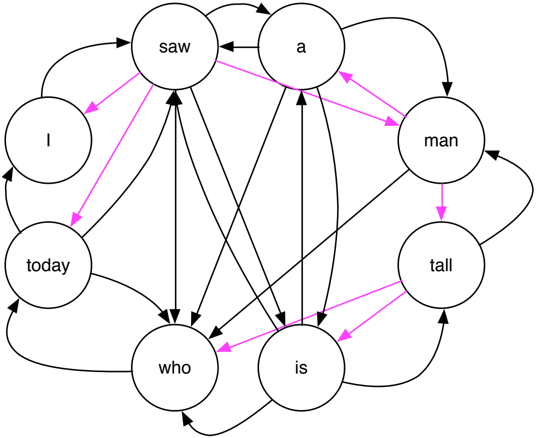
Projectivity



- What happens if you run an oracle on a non-projective sentence?

MST Parsing

- We start out with a fully connected graph with a score for each edge
- N^2 edges total

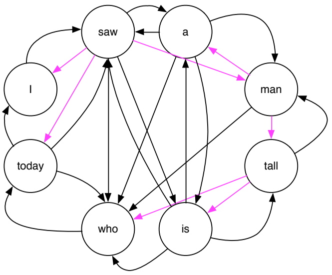


(Assume one edge connects each node as dependent and node as head, N^2 total)

MST Parsing

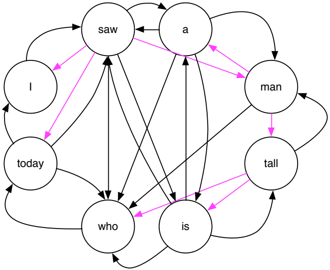
- From this graph G, we want to find a spanning tree (tree that spans G [includes all the vertices in G])

- If the edges have weights, the best parse is the maximal spanning tree (the spanning tree with the highest total weight).



MST Parsing

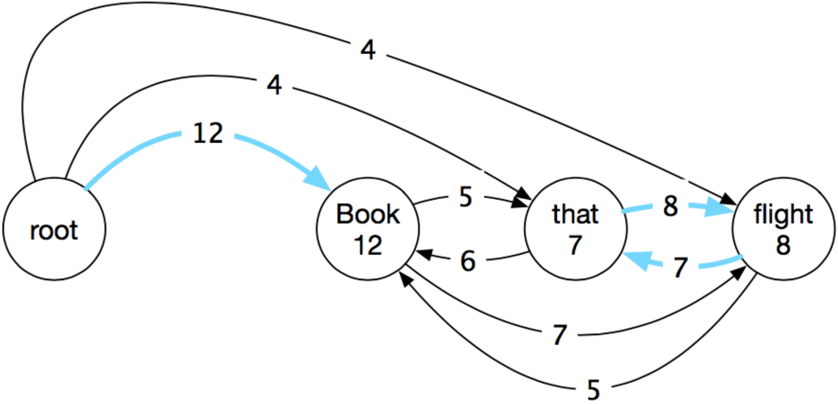
- To find the MST of any graph, we can use the Chu-Liu-Edmonds algorithm in $O(n^3)$ time.
- More efficient Gabow et al. find the MST in $O(n^2+n \log n)$



Chu-Liu-Edmonds (Chu and Liu 1965, Edmonds 1967)

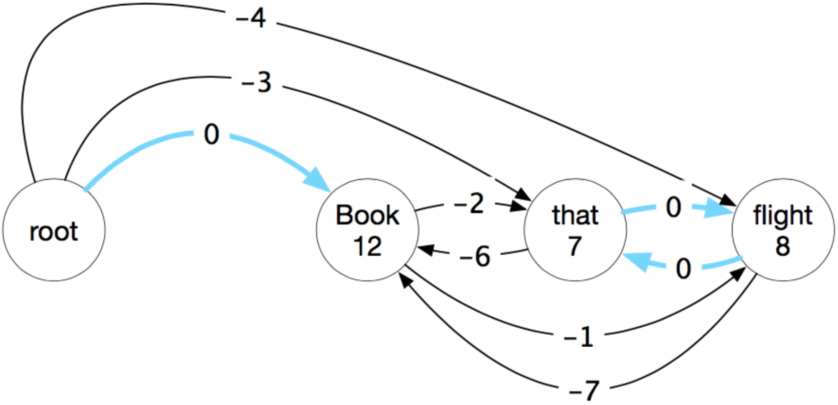
- We have a graph and want to find its spanning tree
- **Greedily select** the best incoming edge to each node (and subtract its score from all incoming edges)
- If there are cycles, select a cycle and **contract** it into a single node
- **Recursively call** the algorithm on the graph with the contracted node
- **Expand** the contracted node, deleting an edge appropriately

Chu-Liu-Edmonds (1): Find the Best Incoming



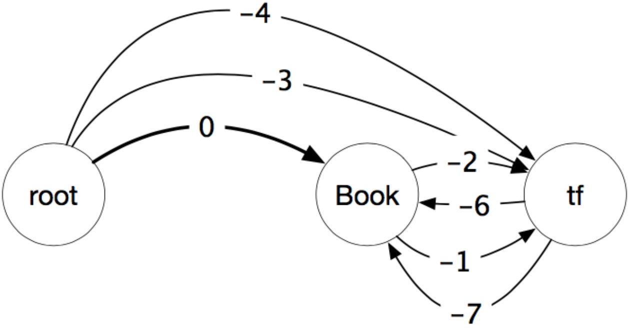
(Figure Credit: Jurafsky and Martin)

Chu-Liu-Edmonds (2): Subtract the Max for Each



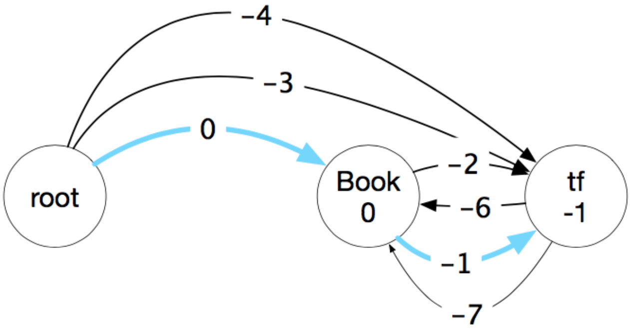
(Figure Credit: Jurafsky and Martin)

Chu-Liu-Edmonds (3): Contract a Node



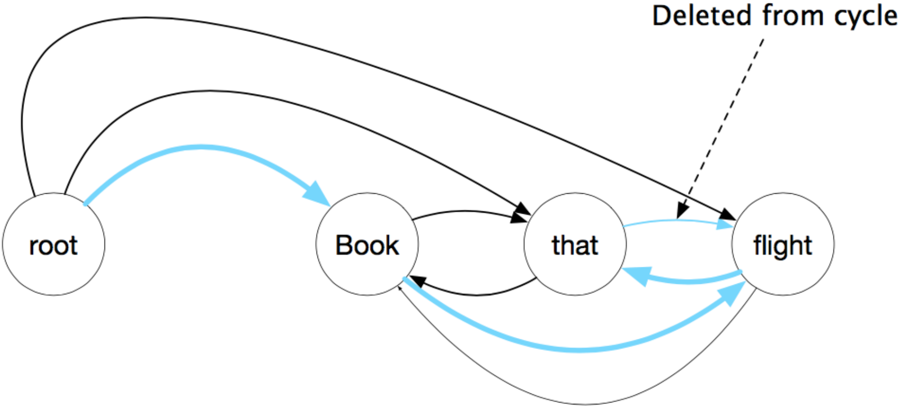
(Figure Credit: Jurafsky and Martin)

Chu-Liu-Edmonds (4): Recursively Call Algorithm



(Figure Credit: Jurafsky and Martin)

Chu-Liu-Edmonds (5): Expand Nodes and Delete Edge



(Figure Credit: Jurafsky and Martin)

Features for Graph-based Parsing (McDonald et al. 2005)

- What features did we use before neural nets?

a)

Basic Uni-gram Features
p-word, p-pos
p-word
p-pos
c-word, c-pos
c-word
c-pos

b)

Basic Big-ram Features
p-word, p-pos, c-word, c-pos
p-pos, c-word, c-pos
p-word, c-word, c-pos
p-word, p-pos, c-pos
p-word, p-pos, c-word
p-word, c-word
p-pos, c-pos

c)

In Between POS Features
p-pos, b-pos, c-pos
Surrounding Word POS Features
p-pos, p-pos+1, c-pos-1, c-pos
p-pos-1, p-pos, c-pos-1, c-pos
p-pos, p-pos+1, c-pos, c-pos+1
p-pos-1, p-pos, c-pos, c-pos+1

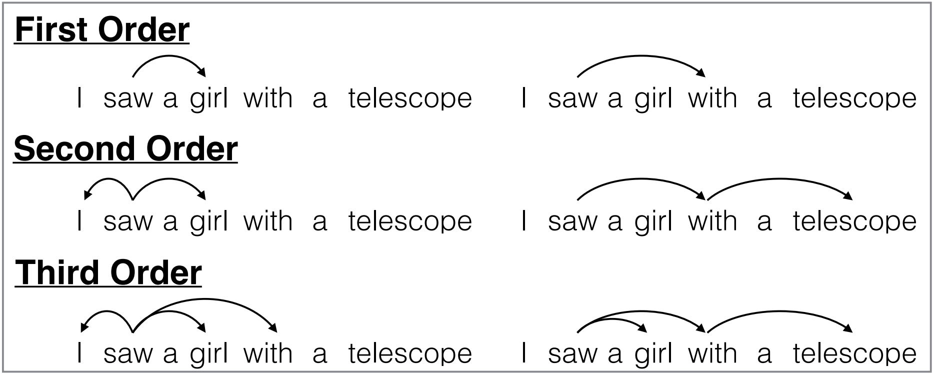
Table 1: Features used by system. p-word: word of parent node in dependency tree. c-word: word of child node. p-pos: POS of parent node. c-pos: POS of child node. p-pos+1: POS to the right of parent in sentence. p-pos-1: POS to the left of parent. c-pos+1: POS to the right of child. c-pos-1: POS to the left of child. b-pos: POS of a word in between parent and child nodes.

- All conjoined with arc direction and arc distance
- Also use POS combination features
- Also represent words w/ prefix if they are long

Higher-order Dependency Parsing

(e.g. Zhang and McDonald 2012)

- Consider multiple edges at a time when calculating scores



- + Can extract more expressive features
- - Higher computational complexity, approximate search necessary

Relation Between **Perceptron Loss** and **Conditional Log Likelihood**

Multinomial Logistic Regression: Conditional Log Likelihood

Logistic regression minimizes conditional log likelihood

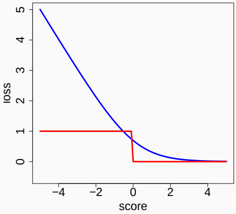
$$\begin{aligned}
-\log\left(p_{\mathbf{w}}(Y = y \mid \boldsymbol{x})\right) &= -\log\left(\frac{\exp \mathbf{w} \cdot f(\boldsymbol{x}, y)}{\sum_{y' \in \mathcal{Y}} \exp \mathbf{w} \cdot f(\boldsymbol{x}, y')}\right) \\
&= -\mathbf{w} \cdot f(\boldsymbol{x}, y) + \left(\log \sum_{y' \in \mathcal{Y}} \exp \mathbf{w} \cdot f(\boldsymbol{x}, y')\right)
\end{aligned}$$

Conditional Log Likelihood

Negated log-likelihood, also known as **log loss**, **conditional negative log-likelihood (CNNL)**, or **cross-entropy**:

$$\left(\log \sum_{y' \in \mathcal{Y}} \exp \mathbf{w} \cdot f(\boldsymbol{x}, y')\right) - \mathbf{w} \cdot f(\boldsymbol{x}, y)$$

In the binary case, where "score" is the score of the correct label:

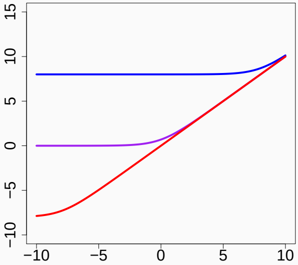


"Log Sum Exp"

Consider the "$\log \sum \exp$" part of the loss function, with two labels.

For illustration purposes, consider one score it be fixed.

This function acts like a "soft max"



$\log(e^x + e^8)$, $\log(e^x + e^0)$, $\log(e^x + e^{-8})$

Softmax

The **softmax** function produces a probability distribution from a set of scores v_i:

$$\text{softmax}(v)_i = \frac{e^{v_i}}{\sum_i e^{v_i}}$$

The following function acts like a real "soft max:"

$$\log(\sum_i e^{v_i})$$

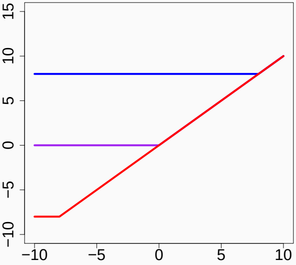
In the limit as scores become large, it reduces to max:

$$\lim_{\alpha \to \infty} \log(\sum_i e^{\alpha v_i}) = \max_i \alpha v_i$$

because the largest v_i dominates in the sum.

Hard Maximum

Why not use a hard max instead?



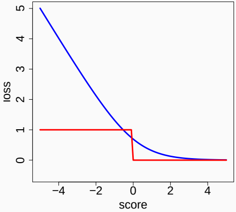
$\max(x, 8)$, $\max(x, 0)$, $\max(x, -8)$

Board work

Convert the **softmax** in CNLL to a **hard max** (3 min, on your own):

$$\left(\log \sum_{y' \in \mathcal{Y}} \exp \mathbf{w} \cdot f(\boldsymbol{x}, y')\right) - \mathbf{w} \cdot f(\boldsymbol{x}, y)$$

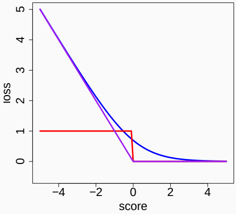
Discuss with a partner (2 min).



This is the Perceptron Loss!

$$\left(\max_{y' \in \mathcal{Y}} \mathbf{w} \cdot f(\boldsymbol{x}, y')\right) - \mathbf{w} \cdot f(\boldsymbol{x}, y)$$

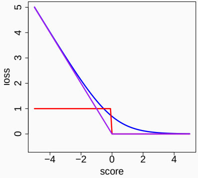
In the binary case:



Log Loss and Perceptron Loss for (x, y)

$$\text{log loss:} \quad \left(\log \sum_{y' \in \mathcal{Y}} \exp \mathbf{w} \cdot f(\boldsymbol{x}, y')\right) - \mathbf{w} \cdot f(\boldsymbol{x}, y)$$

$$\text{Perceptron loss:} \quad \left(\max_{y' \in \mathcal{Y}} \mathbf{w} \cdot f(\boldsymbol{x}, y')\right) - \mathbf{w} \cdot f(\boldsymbol{x}, y)$$



In purple is Perceptron loss, in blue is log loss; in red is "zero-one" loss (error). "Score" is the score of the correct label in binary case.

Comparision Between **Structured Perceptrons** and **Conditional Random Fields**

Last Quarter: Logistic Regression $\rightarrow$ CRFs

Logistic regression:

$$P(y|x) = \frac{\exp(\theta \cdot f(x, y))}{\sum_{y'} \exp(\theta \cdot f(x, y'))}$$

Trained to maximize conditional log probability of the data:

$$\sum_{i=1}^{N} \log P(y_i | x_i)$$

Predictions:

$$\hat{y} = \underset{y}{\operatorname{argmax}} P(y|x)$$

Last Quarter: Logistic Regression $\rightarrow$ CRFs

Logistic regression:

$$P(y|x) = \frac{\exp(\theta \cdot f(x,y))}{\sum_{y'} \exp(\theta \cdot f(x,y'))}$$

CRF:

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp(score(\boldsymbol{x},\boldsymbol{y}'))}{\sum_{\boldsymbol{y}'} \exp(score(\boldsymbol{x},\boldsymbol{y}'))}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are now **sequences**.
Trained to maximize conditional log probability of the data (same as logistic regression):

$$\sum_{i=1}^{N} \log P(\boldsymbol{y}_i|\boldsymbol{x}_i)$$

Predictions: $\hat{\boldsymbol{y}} = \mathrm{argmax}_{\boldsymbol{y}}\, P(\boldsymbol{y}|\boldsymbol{x})$ (found using Viterbi algorithm)

Last Quarter: CRFs for Sequence Labeling

Training: maximize $\sum_{i=1}^{N} \log P_\theta(\boldsymbol{y}_i|\boldsymbol{x}_i)$ (using gradient ascent)

$$\log(P_\theta(\boldsymbol{y}|\boldsymbol{x})) = score_\theta(\boldsymbol{x}, \boldsymbol{y}')) - \log(Z_\theta)$$

where $Z_\theta = \sum_{\boldsymbol{y}'} \exp(score_\theta(\boldsymbol{x}, \boldsymbol{y}'))$ is computed using the **Forward algorithm**.

The $score$ is a sum of “local parts”:

$$score_\theta(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} s_\theta(\mathbf{x}, i, y_i, y_{i-1})$$

Predictions: $\hat{\boldsymbol{y}} = \operatorname{argmax}_{\boldsymbol{y}} P_\theta(\boldsymbol{y}|\boldsymbol{x}) = \operatorname{argmax}_{\boldsymbol{y}} \log(P_\theta(\boldsymbol{y}|\boldsymbol{x}))$
$= \operatorname{argmax}_{\mathbf{y}} \sum_{i=1}^{n} s_\theta(\mathbf{x}, i, y_i, y_{i-1})$ (found using **Viterbi algorithm**)

Compare to Perceptron Algorithm

- $score_\theta(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=0}^{n} s_\theta(\boldsymbol{x}, i, y_i, y_{i+1})$
- Decoding algorithm = Viterbi algorithm (from last quarter)
- Loss function = Perceptron loss

$$L(\boldsymbol{x}, \boldsymbol{y}, \theta) = -score_\theta(\boldsymbol{x}, \boldsymbol{y}) + \max_{\boldsymbol{y}' \in \mathcal{Y}} score_\theta(\boldsymbol{x}, \boldsymbol{y}')$$

- Optimizer = stochastic subgradient descent (SSGD)

Can we use CRFs to train a dependency parser?

Can we use CRFs to train a dependency parser?

It turns out, yes! CRF Dependency Parsing

Last Quarter: CRF Training for Dependency Parsing

Need to compute the normalizer:

$$Z_\theta = \sum_{\boldsymbol{y}'} \exp(score_\theta(\boldsymbol{x}, \boldsymbol{y}'))$$

Can be computed using:

- **Non-projective parsing**: Matrix-tree theorem can compute normalizer (and marginals) (Koo et al. 2007)
- **Projective parsing**: Eisner algorithm with the $+, \times$ semiring can compute the normalizer (Eisner et al. 1996) (next week)

Applied to neural models in Ma et al. (2017)

Neural Parsing Methods

Training

We're training to predict the parser action
—Shift, RightArc(label), LeftArc(label)—
given the featurized configuration

Configuration features	Label
<stack1 = me, 1>, <stack2 = book, 1>, <stack1 POS = PRP, 1>, <buffer1 = the, 1>,	Shift
<stack1 = me, 0>, <stack2 = book, 0>, <stack1 POS = PRP, 0>, <buffer1 = the, 0>,	RightArc(det)
<stack1 = me, 0>, <stack2 = book, 1>, <stack1 POS = PRP, 0>, <buffer1 = the, 0>,	RightArc(nsubj)

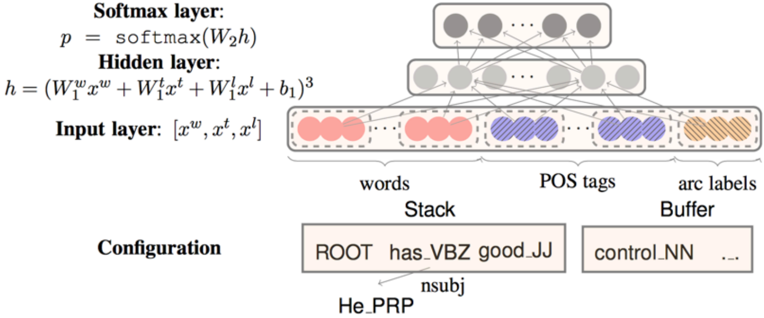
Neural Shift-Reduce Parsing

- We can train a neural shift-reduce parser by just changing how we:
  - represent the configuration
  - predict the label from that representation

- Otherwise training and prediction remains the same.

Chen and Manning (2014), “A Fast and Accurate Dependency Parser using Neural Networks”

Neural Shift-Reduce Parsing



Chen and Manning (2014), “A Fast and Accurate Dependency Parser using Neural Networks”

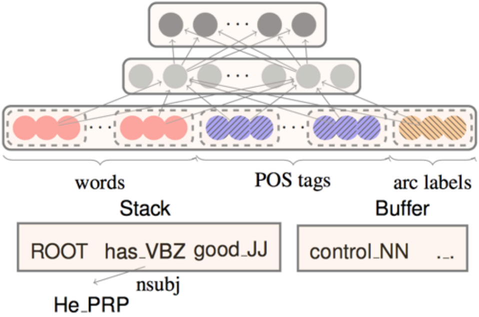
Neural Shift-Reduce Parsing

Representation for configuration:

- Embeddings for words/POS tags on top of stack
- Embeddings for words/POS tags at front of buffer
- Embeddings for existing arc labels at specific positions

Classifier:

- Feed-forward neural network (input representation has a fixed dimensionality)



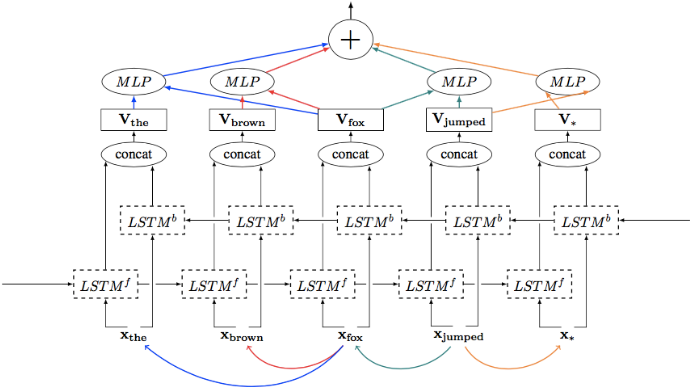
Chen and Manning (2014), “A Fast and Accurate Dependency Parser using Neural Networks”

Neural Models for Graph-based Parsing

Neural Feature Combinators
(Pei et al. 2015)

- Extract traditional features, let NN do feature combination
  - Similar to Chen and Manning (2014)'s transition-based model
- Use **cube + tanh** activation function
- Use **averaged embeddings of phrases**
- Use **second-order features**

BiLSTM Feature Extractors
(Kipperwasser and Goldberg 2016)



- Simpler and better accuracy than manual extraction

BiAffine Classifier

(Dozat and Manning 2017)

$$\mathbf{h}_i^{(arc\text{-}dep)} = \mathrm{MLP}^{(arc\text{-}dep)}(\mathbf{r}_i)$$
$$\mathbf{h}_j^{(arc\text{-}head)} = \mathrm{MLP}^{(arc\text{-}head)}(\mathbf{r}_j)$$

Learn specific representations for head/dependent for each word

$$\mathbf{s}_i^{(arc)} = H^{(arc\text{-}head)} U^{(1)} \mathbf{h}_i^{(arc\text{-}dep)} + H^{(arc\text{-}head)} \mathbf{u}^{(2)}$$

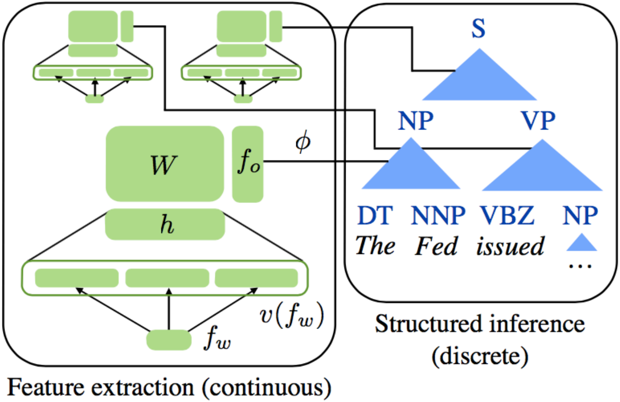
Calculate score of each arc

- Just optimize the likelihood of the parent, no structured training
  - This is a local model, with global decoding using MST at the end
- Best results (with careful parameter tuning) on universal dependencies parsing task

Neural Phrase-Structured Parsing Methods

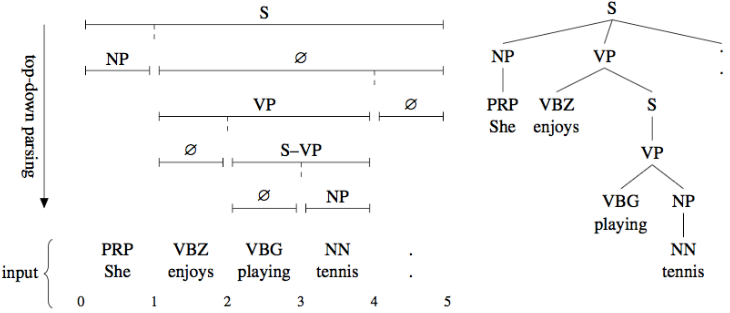
Neural CRF Parsing
(Durrett and Klein 2015)

- Predict score of each span using FFNN
- Do discrete structured inference using CKY, inside-outside



Span Labeling
(Stern et al. 2017)

- Simple idea: try to decide whether span is constituent in tree or not



(a) Execution of the top-down parsing algorithm. (b) Output parse tree.

- Allows for various loss functions (local vs. structured), inference algorithms (CKY, top down)

Stern et al 2017

Final Parsing Results on Penn Treebank

Parser	LR	LP	F1
Durrett and Klein (2015)	–	–	91.1
Vinyals et al. (2015)	–	–	88.3
Dyer et al. (2016)	–	–	89.8
Cross and Huang (2016)	90.5	92.1	91.3
Liu and Zhang (2016)	91.3	92.1	91.7
Best Chart Parser	90.63	92.98	91.79
Best Top-Down Parser	90.35	93.23	91.77

An Alternative:
Parse Reranking

An Alternative: Parse Reranking

- You have a nice model, but it’s hard to implement a dynamic programming decoding algorithm
- Try reranking!
    - Generate with an easy-to-decode model
    - Rescore with your proposed model

Examples of Reranking

- Inside-outside recursive neural networks (Le and Zuidema 2014)
- Parsing as language modeling (Choe and Charniak 2016)
- Recurrent neural network grammars (Dyer et al. 2016)