COMS 4705.H: Graph-Based Dependency Parsing

Karl Stratos

March 24, 2017

Transition- vs. Graph-Based

- We covered transition-based dependency parsing last time.
- We will now cover graph-based dependency parsing.
- Transition-based is simpler, faster, and sometimes even more accuate than graph-based.
- Regressive?

Not Regressive!

1. It's another important example of structured problems omnipresent in NLP with applications beyond parsing.

2. It will illustrate how classical structured NLP techniques can be naturally extended to neural networks.

Overview

Graph-Based Dependency Parsing

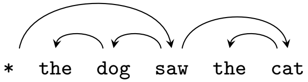
- Eisner's Algorithm
- Classical Parser with Feature Engineering

Very Quick Introduction to Neural Networks

Neural Extension of Classical Parser

Graph-Based Dependency Parsing

- Goal: Find a correct dependency tree for a given sentence. (Ignore arc labels for now.)



- As a **structured** problem: Given a sentence $x = x_1 \dots x_m$ with a set of possible dependency trees $T(x)$, find

$$y^* = \arg\max_{y \in T(x)} \ \mathsf{score}(x, y)$$

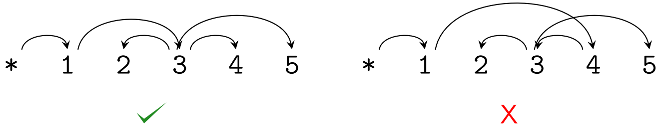
- **How can we efficiently search over $T(x)$?**

Assumptions for Efficient Search

1. Tree score **factorizes** into individual arc scores.

$$\text{score}(x, y) = \sum_{(i,j) \in y} \text{score}(x, i, j)$$

2. $T(x)$ only contains **projective** dependency trees.



A Bottom-Up Dynamic Programming Algorithm

- Under these assumptions, given score(x, i, j) for all i, j, we can compute in $O(m^3)$ time:

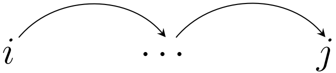
$$y^* = \arg\max_{y \in T(x)} \sum_{(i,j) \in y} \mathsf{score}(x, i, j)$$

- Similar to the CKY algorithm for context-free grammars, but needs an extra case analysis of dependency substructures.

Dependecy Substructure Cases

For any projective substructure y spanning nodes $i \ldots j$ where $i < k < j$ has collected all its children, either

1. **Direction** $\rightarrow$

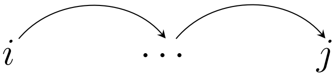


- **Complete** if j does not expect any more children.
- **Incomplete** if j expects more children.

Dependecy Substructure Cases

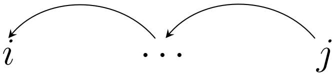
For any projective substructure y spanning nodes $i \ldots j$ where $i < k < j$ has collected all its children, either

1. **Direction** $\rightarrow$



- **Complete** if j does not expect any more children.
- **Incomplete** if j expects more children.

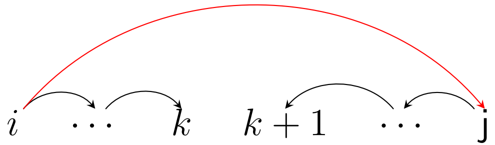
2. **Direction** $\leftarrow$



- **Complete** if i does not expect any more children.
- **Incomplete** if i expects more children.

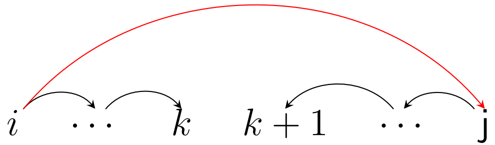
Substructure Spanning $i \ldots j$ with Direction $\rightarrow$: $i < k < j$ Has Collected All Children

- Incomplete? Formed by joining two complete arcs rooted at i and j with a right arc $i \rightarrow j$.

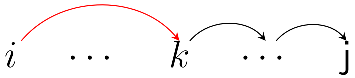


Substructure Spanning $i \ldots j$ with Direction $\rightarrow$: $i < k < j$ Has Collected All Children

- Incomplete? Formed by joining two complete arcs rooted at i and j with a right arc $i \rightarrow j$.



- Complete? Formed by gluing an incomplete arc rooted at i to a complete arc ending at j.



Chart

- Nodes $0 \leq i \leq j \leq m$
- Directions $d \in \{\leftarrow, \rightarrow\}$
- Completeness $c \in \{T, F\}$

Define a chart:

$$\pi(i, j, d, c) = \max_{\substack{y \in T(x_i \ldots x_j): \\ y.d = d,\ y.c = c}} \sum_{(s,t) \in y} \mathsf{score}(x, s, t)$$

Chart

- Nodes $0 \leq i \leq j \leq m$
- Directions $d \in \{\leftarrow, \rightarrow\}$
- Completeness $c \in \{T, F\}$

Define a chart:

$$\pi(i, j, d, c) = \max_{\substack{y \in T(x_i \ldots x_j): \\ y.d = d,\ y.c = c}} \sum_{(s,t) \in y} \mathsf{score}(x, s, t)$$

By definition, the score of an optimal complete projective tree is

$$\pi(0, m, \rightarrow, T)$$

Eisner's Algorithm (1996)

Input: score(x, i, j) for all $i, j \in \{0 \ldots m\}$
Output: score of an optimal complete projective tree

1. For $i = 0 \ldots m$:

2. For $l = 1 \ldots m$, for $i = 0 \ldots m - l$: set $j = i + l$ and

3. Return $\pi(0, m, \rightarrow, T)$.

Eisner's Algorithm (1996)

Input: score(x, i, j) for all $i, j \in \{0 \ldots m\}$
Output: score of an optimal complete projective tree

1. For $i = 0 \ldots m$:

$$\pi(i, i, \leftarrow, T) = \pi(i, i, \rightarrow, T) = \pi(i, i, \leftarrow, F) = \pi(i, i, \rightarrow, F) = 0$$

2. For $l = 1 \ldots m$, for $i = 0 \ldots m - l$: set $j = i + l$ and

3. Return $\pi(0, m, \rightarrow, T)$.

Eisner's Algorithm (1996)

Input: score(x, i, j) for all $i, j \in \{0 \ldots m\}$
Output: score of an optimal complete projective tree

1. For $i = 0 \ldots m$:

$$\pi(i, i, \leftarrow, T) = \pi(i, i, \rightarrow, T) = \pi(i, i, \leftarrow, F) = \pi(i, i, \rightarrow, F) = 0$$

2. For $l = 1 \ldots m$, for $i = 0 \ldots m - l$: set $j = i + l$ and

$$\pi(i, j, \rightarrow, F) = \max_{i \leq k < j} \pi(i, k, \rightarrow, T) + \pi(k + 1, j, \leftarrow, T) + \text{score}(x, i, j)$$

3. Return $\pi(0, m, \rightarrow, T)$.

Eisner's Algorithm (1996)

Input: score(x, i, j) for all $i, j \in \{0 \ldots m\}$
Output: score of an optimal complete projective tree

1. For $i = 0 \ldots m$:

$$\pi(i, i, \leftarrow, T) = \pi(i, i, \rightarrow, T) = \pi(i, i, \leftarrow, F) = \pi(i, i, \rightarrow, F) = 0$$

2. For $l = 1 \ldots m$, for $i = 0 \ldots m - l$: set $j = i + l$ and

$$\pi(i, j, \rightarrow, F) = \max_{i \leq k < j} \pi(i, k, \rightarrow, T) + \pi(k + 1, j, \leftarrow, T) + \text{score}(x, i, j)$$
$$\pi(i, j, \rightarrow, T) = \max_{i < k \leq j} \pi(i, k, \rightarrow, F) + \pi(k, j, \rightarrow, T)$$

3. Return $\pi(0, m, \rightarrow, T)$.

Eisner's Algorithm (1996)

Input: score(x, i, j) for all $i, j \in \{0 \ldots m\}$
Output: score of an optimal complete projective tree

1. For $i = 0 \ldots m$:

$$\pi(i, i, \leftarrow, T) = \pi(i, i, \rightarrow, T) = \pi(i, i, \leftarrow, F) = \pi(i, i, \rightarrow, F) = 0$$

2. For $l = 1 \ldots m$, for $i = 0 \ldots m - l$: set $j = i + l$ and

$$\pi(i, j, \rightarrow, F) = \max_{i \leq k < j} \pi(i, k, \rightarrow, T) + \pi(k+1, j, \leftarrow, T) + \text{score}(x, i, j)$$
$$\pi(i, j, \rightarrow, T) = \max_{i < k \leq j} \pi(i, k, \rightarrow, F) + \pi(k, j, \rightarrow, T)$$
$$\pi(i, j, \leftarrow, F) = \max_{i \leq k < j} \pi(i, k, \rightarrow, T) + \pi(k+1, j, \leftarrow, T) + \text{score}(x, j, i)$$
$$\pi(i, j, \leftarrow, T) = \max_{i \leq k < j} \pi(i, k, \leftarrow, T) + \pi(k, j, \leftarrow, F)$$

3. Return $\pi(0, m, \rightarrow, T)$.

Extracting the Optimal Parse

In step 2, also record a **backpointer**

$$\beta(i, j, \rightarrow, F) = \arg\max_{i \leq k < j} \; \pi(i, k, \rightarrow, T) + \pi(k+1, j, \leftarrow, T) + \mathsf{score}(x, i, j)$$

$$\beta(i, j, \rightarrow, T) = \arg\max_{i < k \leq j} \; \pi(i, k, \rightarrow, F) + \pi(k, j, \rightarrow, T)$$

$$\beta(i, j, \leftarrow, F) = \arg\max_{i \leq k < j} \; \pi(i, k, \rightarrow, T) + \pi(k+1, j, \leftarrow, T) + \mathsf{score}(x, j, i)$$

$$\beta(i, j, \leftarrow, T) = \arg\max_{i \leq k < j} \; \pi(i, k, \leftarrow, T) + \pi(k, j, \leftarrow, F)$$

Extracting the Optimal Parse

In step 2, also record a **backpointer**

$$\beta(i, j, \rightarrow, F) = \underset{i \leq k < j}{\arg\max}\ \pi(i, k, \rightarrow, T) + \pi(k+1, j, \leftarrow, T) + \mathsf{score}(x, i, j)$$

$$\beta(i, j, \rightarrow, T) = \underset{i < k \leq j}{\arg\max}\ \pi(i, k, \rightarrow, F) + \pi(k, j, \rightarrow, T)$$

$$\beta(i, j, \leftarrow, F) = \underset{i \leq k < j}{\arg\max}\ \pi(i, k, \rightarrow, T) + \pi(k+1, j, \leftarrow, T) + \mathsf{score}(x, j, i)$$

$$\beta(i, j, \leftarrow, T) = \underset{i \leq k < j}{\arg\max}\ \pi(i, k, \leftarrow, T) + \pi(k, j, \leftarrow, F)$$

Call **Backtrack**$(\beta, 0, m, \rightarrow, T, h)$ where

$$h(i) = \text{head/parent of node } i \qquad \forall i = 1 \dots m$$

will be populated.

Backtracking

Backtrack
Input: backpointer β, $0 \le i \le j \le m$, $d \in \{\leftarrow, \rightarrow\}$, $c \in \{T, F\}$, h

- If $i = j$: Return.
- Let $k = \beta(i, j, d, c)$.
- If $c = T$:
  - If $d = \rightarrow$: **Backtrack**$(i, k, \rightarrow, F, h)$, **Backtrack**$(k, j, \rightarrow, T, h)$
  - If $d = \leftarrow$: **Backtrack**$(i, k, \leftarrow, T, h)$, **Backtrack**$(k, j, \leftarrow, F, h)$
- If $c = F$:
  - If $d = \rightarrow$: Set $h(j) = i$.
  - If $d = \leftarrow$: Set $h(i) = j$.
  - **Backtrack**$(i, k, \rightarrow, T, h)$, **Backtrack**$(k + 1, j, \leftarrow, T, h)$

Overview

- Graph-Based Dependency Parsing
  - Eisner's Algorithm
  - Classical Parser with Feature Engineering
- Very Quick Introduction to Neural Networks
- Neural Extension of Classical Parser

Learning Problem

So far we have assumed score(x, i, j) for every arc $i \to j$.

- Then we can use Eisner's algorithm to predict

$$y^* = \arg\max_{y \in T(x)} \sum_{(i,j) \in y} \text{score}(x, i, j)$$

Goal. Learn a model that can **estimate the score of any arc**

$$i \qquad \cdots \qquad j$$

such that y^* corresponds to the true parse of x.

Classical Feature Representation of an Arc

- Design a **feature template** ϕ that extracts features from data, for instance

$$\phi_{511}(x, i, j) = \begin{cases} 1 & \text{if } x_i = \texttt{saw} \text{ and } x_j.\texttt{POS} = \texttt{NOUN} \\ 0 & \text{otherwise} \end{cases}$$

- Each arc is represented as a high-dimensional, sparse binary vector:

$$\phi(x, i, j) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \\ 1 \\ 0 \end{bmatrix} \in \{0, 1\}^{17324}$$

Example Activated Features for a Single Arc

[went]	[VBD]	[As]	[ADP]	[went]
[VERB]	[As]	[IN]	[went, VBD]	[As, ADP]
[went, As]	[VBD, ADP]	[went, VERB]	[As, IN]	[went, As]
[VERB, IN]	[VBD, As, ADP]	[went, As, ADP]	[went, VBD, ADP]	[went, VBD, As]
[ADJ, *, ADP]	[VBD, *, ADP]	[VBD, ADJ, ADP]	[VBD, ADJ, *]	[NNS, *, ADP]
[NNS, VBD, ADP]	[NNS, VBD, *]	[ADJ, ADP, NNP]	[VBD, ADP, NNP]	[VBD, ADJ, NNP]
[NNS, ADP, NNP]	[NNS, VBD, NNP]	[went, left, 5]	[VBD, left, 5]	[As, left, 5]
[ADP, left, 5]	[VERB, As, IN]	[went, As, IN]	[went, VERB, IN]	[went, VERB, As]
[JJ, *, IN]	[VERB, *, IN]	[VERB, JJ, IN]	[VERB, JJ, *]	[NOUN, *, IN]
[NOUN, VERB, IN]	[NOUN, VERB, *]	[JJ, IN, NOUN]	[VERB, IN, NOUN]	[VERB, JJ, NOUN]
[NOUN, IN, NOUN]	[NOUN, VERB, NOUN]	[went, left, 5]	[VERB, left, 5]	[As, left, 5]
[IN, left, 5]	[went, VBD, As, ADP]	[VBD, ADJ, *, ADP]	[NNS, VBD, *, ADP]	[VBD, ADJ, ADP, NNP]
[NNS, VBD, ADP, NNP]	[went, VBD, left, 5]	[As, ADP, left, 5]	[went, As, left, 5]	[VBD, ADP, left, 5]
[went, VERB, As, IN]	[VERB, JJ, *, IN]	[NOUN, VERB, *, IN]	[VERB, JJ, IN, NOUN]	[NOUN, VERB, IN, NOUN]
[went, VERB, left, 5]	[As, IN, left, 5]	[went, As, left, 5]	[VERB, IN, left, 5]	[VBD, As, ADP, left, 5]
[went, As, ADP, left, 5]	[went, VBD, ADP, left, 5]	[went, VBD, As, left, 5]	[ADJ, *, ADP, left, 5]	[VBD, *, ADP, left, 5]
[VBD, ADJ, ADP, left, 5]	[VBD, ADJ, *, left, 5]	[NNS, *, ADP, left, 5]	[NNS, VBD, ADP, left, 5]	[NNS, VBD, *, left, 5]
[ADJ, ADP, NNP, left, 5]	[VBD, ADP, NNP, left, 5]	[VBD, ADJ, NNP, left, 5]	[NNS, ADP, NNP, left, 5]	[NNS, VBD, NNP, left, 5]
[VERB, As, IN, left, 5]	[went, As, IN, left, 5]	[went, VERB, IN, left, 5]	[went, VERB, As, left, 5]	[JJ, *, IN, left, 5]
[VERB, *, IN, left, 5]	[VERB, JJ, IN, left, 5]	[VERB, JJ, *, left, 5]	[NOUN, *, IN, left, 5]	[NOUN, VERB, IN, left, 5]

Slide from Rush and Petrov (2012)

Linear Model

- Parameter $w \in \mathbb{R}^d$ defining the score of any arc $i \to j$ as

$$\text{score}(x, i, j) = w^\top \phi(x, i, j) = \sum_{k=1:\ \phi_k(x,i,j)=1}^{d} w_k$$

Linear Model

- Parameter $w \in \mathbb{R}^d$ defining the score of any arc $i \to j$ as

$$\text{score}(x, i, j) = w^\top \phi(x, i, j) = \sum_{k=1:\ \phi_k(x,i,j)=1}^{d} w_k$$

- **Learning problem.** Given a training dataset of N annotated sentences $(x^{(1)}, y^{(1)}) \dots (x^{(N)}, y^{(N)})$, find

$$w^* = \operatorname*{arg\,min}_{w \in \mathbb{R}^d} \sum_{i=1}^{N} L\left(x^{(i)}, y^{(i)} \middle| w\right)$$

where $L(x, y|w)$ is some "loss" on (x, y) under parameter w.

One Choice of Loss Function

Define an error-augmented score function ($[[S]] = 1$ if S is true, $[[S]] = 0$ if S is false):

$$\text{aug-score}^{y}(x, y') = \sum_{(i,j) \in y'} \text{score}(x, i, j) + [[(i, j) \notin y]]$$

One Choice of Loss Function

Define an error-augmented score function ($[[S]] = 1$ if S is true, $[[S]] = 0$ if S is false):

$$\text{aug-score}^{y}(x, y') = \sum_{(i,j) \in y'} \text{score}(x, i, j) + [[(i, j) \notin y]]$$

Structured hinge loss:

$$\boxed{L(x, y | w) = \max_{y' \in T(x)} \text{aug-score}^{y}(x, y') - \text{score}(x, y)}$$

One Choice of Loss Function

Define an error-augmented score function ($[[S]] = 1$ if S is true, $[[S]] = 0$ if S is false):

$$\text{aug-score}^{y}(x, y') = \sum_{(i,j) \in y'} \text{score}(x, i, j) + [[(i, j) \notin y]]$$

Structured hinge loss:

$$\boxed{L(x, y | w) = \max_{y' \in T(x)} \text{aug-score}^{y}(x, y') - \text{score}(x, y)}$$

Idea. Make the usual margin constraint to account for the *degree of structural difference*.

$$\text{score}(x, y) - \max_{y' \neq y} \text{score}(x, y') \geq \sum_{(i,j) \in y'} [[(i, j) \notin y]]$$

Calculating Optimal Error-Augmented Score

Input: score(x, i, j) for all $i, j \in \{0 \ldots m\}$, $\boxed{\text{reference tree } y}$

Output: $\max_{y' \in T(x)}$ aug-score$^{y}(x, y')$

1. For $i = 0 \ldots m$:

$$\pi(i, i, \leftarrow, T) = \pi(i, i, \rightarrow, T) = \pi(i, i, \leftarrow, F) = \pi(i, i, \rightarrow, F) = 0$$

2. For $l = 1 \ldots m$, for $i = 0 \ldots m - l$: set $j = i + l$ and

$$\pi(i, j, \rightarrow, F) = \max_{i \leq k < j} \pi(i, k, \rightarrow, T) + \pi(k+1, j, \leftarrow, T) + \text{score}(x, i, j) + [[(i, j) \notin y]]$$

$$\pi(i, j, \rightarrow, T) = \max_{i < k \leq j} \pi(i, k, \rightarrow, F) + \pi(k, j, \rightarrow, T)$$

$$\pi(i, j, \leftarrow, F) = \max_{i \leq k < j} \pi(i, k, \rightarrow, T) + \pi(k+1, j, \leftarrow, T) + \text{score}(x, j, i) + [[(j, i) \notin y]]$$

$$\pi(i, j, \leftarrow, T) = \max_{i \leq k < j} \pi(i, k, \leftarrow, T) + \pi(k, j, \leftarrow, F)$$

3. Return $\pi(0, m, \rightarrow, T)$.

Online Gradient-Based Training

For each sentence-tree instance (x, y),

1. Use the augmented Eisner's algorithm to predict

$$\hat{y} = \arg\max_{y' \in T(x)} \textsf{aug-score}^{y}(x, y')$$

under the current parameter w.

Online Gradient-Based Training

For each sentence-tree instance (x, y),

1. Use the augmented Eisner's algorithm to predict

$$\hat{y} = \arg\max_{y' \in T(x)} \text{aug-score}^{y}(x, y')$$

under the current parameter w.

2. If $\hat{y} \neq y$, let

$$L(x, y|w) = \text{aug-score}^{y}(x, \hat{y}) - \text{score}(x, y) > 0$$

and update w:

$$w \leftarrow w - \eta \nabla_w L(x, y|w)$$

At Test Time

- Error-augmented inference is no longer used.

- Given a new sentence, simply predict

$$y^* = \arg\max_{y \in T(x)} \sum_{(i,j) \in y} \mathsf{score}(x, i, j)$$

where scores are given by the trained w:

$$\mathsf{score}(x, i, j) = \sum_{k=1:\ \phi_k(x,i,j)=1}^{d} w_k$$

Takeaway

No need to understand all details, just remember:

1. Given arc scores, we can find an optimal projective tree in polynomial time using Eisner's algorithm.

Takeaway

No need to understand all details, just remember:

1. Given arc scores, we can find an optimal projective tree in polynomial time using Eisner's algorithm.

2. In a classical parser, we handcraft arc representations and score them with a linear model.

Takeaway

No need to understand all details, just remember:

1. Given arc scores, we can find an optimal projective tree in polynomial time using Eisner's algorithm.
2. In a classical parser, we handcraft arc representations and score them with a linear model.
3. One way to train the model is to optimize some loss on training data: here, structured hinge loss.

Takeaway

No need to understand all details, just remember:

1. Given arc scores, we can find an optimal projective tree in polynomial time using Eisner's algorithm.

2. In a classical parser, we handcraft arc representations and score them with a linear model.

3. One way to train the model is to optimize some loss on training data: here, structured hinge loss.

Now we extend this with neural nets.

Overview

Graph-Based Dependency Parsing

- Eisner's Algorithm
- Classical Parser with Feature Engineering

Very Quick Introduction to Neural Networks

Neural Extension of Classical Parser

“Very Quick” Edition

- We will cover just enough materials to do Assignment 3.
- We will revisit these topics in greater detail later.

What's a Neural Network?

Just a composition of linear/nonlinear functions.

$$f(x) = W^{(L)} \tanh\left(W^{(L-1)} \cdots \tanh\left(W^{(1)} x\right) \cdots\right)$$

What's a Neural Network?

Just a composition of linear/nonlinear functions.

$$f(x) = W^{(L)} \tanh\left(W^{(L-1)} \cdots \tanh\left(W^{(1)}x\right)\cdots\right)$$

More like a paradigm, not a specific model.

1. **Transform** your input $x \longrightarrow f(x)$.

2. Define **loss** between $f(x)$ and the target label y.

3. Train parameters by minimizing the loss.

You May Already Know Some Neural Networks...

Maximum entropy classifier ("maxent") is a neural network with 0 hidden layer and a softmax output layer:

$$p(y|x) := \frac{\exp([Wx]_y)}{\sum_{y'} \exp([Wx]_{y'})} = \mathsf{softmax}_y(Wx)$$

Get W by minimizing $L(W) = -\sum_i \log p(y_i|x_i)$.

You May Already Know Some Neural Networks...

Maximum entropy classifier ("maxent") is a neural network with 0 hidden layer and a softmax output layer:

$$p(y|x) := \frac{\exp([Wx]_y)}{\sum_{y'} \exp([Wx]_{y'})} = \mathsf{softmax}_y(Wx)$$

Get W by minimizing $L(W) = -\sum_i \log p(y_i|x_i)$.

Linear regression is a neural network with 0 hidden layer and the identity output layer:

$$f(x) := Wx$$

Get W by minimizing $L(W) = \sum_i (y_i - f_i(x))^2$.

Feedforward Network

Think: maxent with extra transformation

Feedforward Network

Think: maxent with extra transformation

With 1 hidden layer:

$$h^{(1)} = \tanh(W^{(1)}x)$$
$$p(y|x) = \mathsf{softmax}_y(h^{(1)})$$

Feedforward Network

Think: maxent with extra transformation

With 1 hidden layer:

$$h^{(1)} = \tanh(W^{(1)} x)$$
$$p(y|x) = \mathsf{softmax}_y(h^{(1)})$$

With 2 hidden layers:

$$h^{(1)} = \tanh(W^{(1)} x)$$
$$h^{(2)} = \tanh(W^{(2)} h^{(1)})$$
$$p(y|x) = \mathsf{softmax}_y(h^{(2)})$$

Again, get parameters $W^{(l)}$ by minimizing $-\sum_i \log p(y_i|x_i)$.

Feedforward Network

Think: maxent with extra transformation

With 1 hidden layer:

$$h^{(1)} = \tanh(W^{(1)}x)$$
$$p(y|x) = \mathsf{softmax}_y(h^{(1)})$$

With 2 hidden layers:

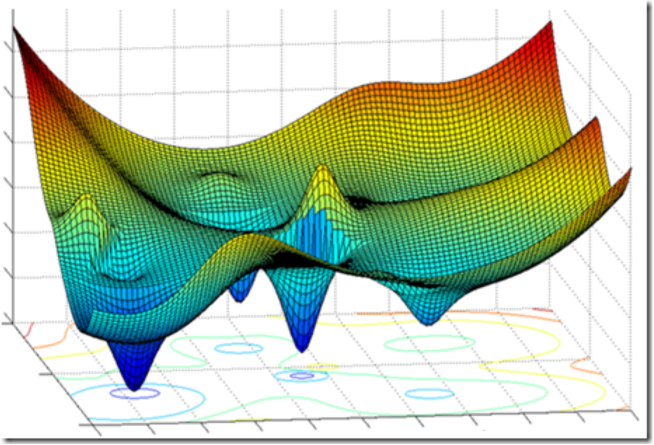
$$h^{(1)} = \tanh(W^{(1)}x)$$
$$h^{(2)} = \tanh(W^{(2)}h^{(1)})$$
$$p(y|x) = \mathsf{softmax}_y(h^{(2)})$$

Again, get parameters $W^{(l)}$ by minimizing $-\sum_i \log p(y_i|x_i)$.

- Q. What's the catch?

Nonconvex Loss Function

But we can still decrease any differentiable loss by following the gradient (*dismayed gasp*).



1. Differentiate the loss wrt. model parameters (backprop)
2. Take a gradient step

Recurrent Network (RNN)

Think: HMM (or Kalman filter) with extra transformation

Recurrent Network (RNN)

Think: HMM (or Kalman filter) with extra transformation

Input: sequence $x_1 \ldots x_m \in \mathbb{R}^d$

- For $i = 1 \ldots m$,

$$h_i = \tanh\left(W x_i + V {\color{red} h_{i-1}}\right)$$

Output: sequence $h_1 \ldots h_m \in \mathbb{R}^{d'}$

RNN $\approx$ Deep Feedforward

Unroll the expression for the last output vector h_m:

$$h_m = \tanh\left(Wx_m + V\left(\cdots + V\tanh\left(Wx_1 + Vh_0\right)\cdots\right)\right)$$

It's just a deep "feedforward network" with one important difference: **parameters are reused**

- (V, W) are applied m times

Training: do backprop on this unrolled network, update parameters

LSTM

- RNN produces a sequence of output vectors

$$x_1 \ldots x_m \quad \longrightarrow \quad h_1 \ldots h_m$$

- LSTM produces "memory cell vectors" along with output

$$x_1 \ldots x_m \quad \longrightarrow \quad c_1 \ldots c_m,\ h_1 \ldots h_m$$

- These $c_1 \ldots c_m$ enable the network to keep or drop information from previous states.

LSTM: Details

At each time step i,

- Compute a *masking vector* for the memory cell:

$$q_i = \sigma\left(U^q x + V^q h_{i-1} + W^i c_{i-1}\right) \in [0,1]^{d'}$$

LSTM: Details

At each time step i,

- Compute a *masking vector* for the memory cell:

$$q_i = \sigma\left(U^q x + V^q h_{i-1} + W^i c_{i-1}\right) \in [0,1]^{d'}$$

- Use q_i to keep/forget dimensions in previous memory cell:

$$c_i = (1 - q_i) \odot c_{i-1} + q_i \odot \tanh\left(U^c x + V^c h_{i-1}\right)$$

LSTM: Details

At each time step i,

- Compute a *masking vector* for the memory cell:

$$q_i = \sigma\left(U^q x + V^q h_{i-1} + W^i c_{i-1}\right) \in [0,1]^{d'}$$

- Use q_i to keep/forget dimensions in previous memory cell:

$$c_i = (1 - q_i) \odot c_{i-1} + q_i \odot \tanh\left(U^c x + V^c h_{i-1}\right)$$

- Compute *another masking vector* for the output:

$$o_i = \sigma\left(U^o x + V^o h_{i-1} + W^o c_i\right) \in [0,1]^{d'}$$

LSTM: Details

At each time step i,

- Compute a *masking vector* for the memory cell:

$$q_i = \sigma\left(U^q x + V^q h_{i-1} + W^i c_{i-1}\right) \in [0,1]^{d'}$$

- Use q_i to keep/forget dimensions in previous memory cell:

$$c_i = (1 - q_i) \odot c_{i-1} + q_i \odot \tanh\left(U^c x + V^c h_{i-1}\right)$$

- Compute *another masking vector* for the output:

$$o_i = \sigma\left(U^o x + V^o h_{i-1} + W^o c_i\right) \in [0,1]^{d'}$$

- Use o_i to keep/forget dimensions in current memory cell:

$$h_i = o_i \odot \tanh(c_i)$$

Recap

- A **neural network** is just a composition of linear and nonlinear functions.

Recap

- A **neural network** is just a composition of linear and nonlinear functions.

- A **neural paradigm** is to
  1. Transform input x via a network to obtain a prediction $\hat{y}$.
  2. Compute a loss $l(y, \hat{y})$ with respect to true output y.
  3. Learn network parameters by minimizing $l(y, \hat{y})$.

Recap

- A **neural network** is just a composition of linear and nonlinear functions.

- A **neural paradigm** is to
  1. Transform input x via a network to obtain a prediction $\hat{y}$.
  2. Compute a loss $l(y, \hat{y})$ with respect to true output y.
  3. Learn network parameters by minimizing $l(y, \hat{y})$.

- With a neural library like DyNet/Torch/TensorFlow/..., doing this is **embarrassingly easy**.
  1. Define your network ("feedforward here, RNN there").
  2. Define a loss function between prediction and true label.
  3. Give labeled data to the library and let it optimize parameters.

Overview

Graph-Based Dependency Parsing

- Eisner's Algorithm
- Classical Parser with Feature Engineering

Very Quick Introduction to Neural Networks

Neural Extension of Classical Parser

Basic Idea of Kiperwasser and Goldberg (2016)

We just need a model to give a score to any arc.

$$\begin{tikzcd} i \arrow[rr, bend left] & \cdots & j \end{tikzcd}$$

Previously score was given by linear model using handcrafted ϕ:

$$\text{score}(x, i, j) = w^\top \phi(x, i, j)$$

Now score will be given by neural network:

$$\text{score}(x, i, j) = \textbf{MyNetwork}(x, i, j)$$

Parameters

- Vector $e_x \in \mathbb{R}^{100}$ for every word x
- Vector $e_y \in \mathbb{R}^{25}$ for every POS tag y
- Forward LSTM $\mathbb{R}^{125} \rightarrow \mathbb{R}^{125}$
- Backward LSTM $\mathbb{R}^{125} \rightarrow \mathbb{R}^{125}$
- Feedforward parameters (W^1, W^2, b^1, b^2)

We will compute a transformed input vector for each word in

`the dog saw the cat`

Bidirectional LSTM Layer

Run LSTM on word-POS vectors forward:

$$\begin{bmatrix} e_{\text{the}} \\ e_{\text{D}} \end{bmatrix} \begin{bmatrix} e_{\text{dog}} \\ e_{\text{N}} \end{bmatrix} \begin{bmatrix} e_{\text{saw}} \\ e_{\text{V}} \end{bmatrix} \begin{bmatrix} e_{\text{the}} \\ e_{\text{D}} \end{bmatrix} \begin{bmatrix} e_{\text{cat}} \\ e_{\text{N}} \end{bmatrix} \longrightarrow$$

$$f_1 \;\; f_2 \;\; f_3 \;\; f_4 \;\; f_5 \in \mathbb{R}^{125}$$

Run LSTM on word-POS vectors backward:

$$\begin{bmatrix} e_{\text{cat}} \\ e_{\text{N}} \end{bmatrix} \begin{bmatrix} e_{\text{the}} \\ e_{\text{D}} \end{bmatrix} \begin{bmatrix} e_{\text{saw}} \\ e_{\text{V}} \end{bmatrix} \begin{bmatrix} e_{\text{dog}} \\ e_{\text{N}} \end{bmatrix} \begin{bmatrix} e_{\text{the}} \\ e_{\text{D}} \end{bmatrix} \longrightarrow$$

$$b_1 \;\; b_2 \;\; b_3 \;\; b_4 \;\; b_5 \in \mathbb{R}^{125}$$

Get a sentence-aware representation of `dog` as:

$$z_2 = \begin{bmatrix} f_2 \\ b_4 \end{bmatrix} \in \mathbb{R}^{250}$$

Arc Representation and Score

$$\phi(x, i, j) = \begin{bmatrix} z_i \\ z_j \end{bmatrix}$$

$$\begin{tikzcd} z_i \arrow[rr, bend left] & \cdots & z_j \end{tikzcd}$$

$$\mathsf{score}(x, i, j) = W^2 \tanh\left(W^1 \phi(x, i, j) + b^1\right) + b^2$$

Neural Parser

- Use the neural score function **exactly as before**.

$$\text{score}(x, y) = \sum_{(i,j) \in y} \text{score}(x, i, j)$$

$$\text{aug-score}^{y}(x, y') = \sum_{(i,j) \in y'} \text{score}(x, i, j) + [[(i, j) \notin y]]$$

Neural Parser

- Use the neural score function **exactly as before**.

$$\text{score}(x, y) = \sum_{(i,j) \in y} \text{score}(x, i, j)$$

$$\text{aug-score}^{y}(x, y') = \sum_{(i,j) \in y'} \text{score}(x, i, j) + [[(i, j) \notin y]]$$

- Training: Optimize loss over labeled data **exacty as before**.

$$\Theta^* = \arg\min_{\Theta \in \mathbb{R}^d} \sum_{i=1}^{N} L\left(x^{(i)}, y^{(i)} \middle| \Theta\right)$$

where Θ now refers to all parameters of the neural network.

Neural Parser

- Use the neural score function **exactly as before**.

$$\text{score}(x, y) = \sum_{(i,j) \in y} \text{score}(x, i, j)$$

$$\text{aug-score}^{y}(x, y') = \sum_{(i,j) \in y'} \text{score}(x, i, j) + [[(i, j) \notin y]]$$

- Training: Optimize loss over labeled data **exacty as before**.

$$\Theta^* = \arg\min_{\Theta \in \mathbb{R}^d} \sum_{i=1}^{N} L\left(x^{(i)}, y^{(i)} \middle| \Theta\right)$$

where Θ now refers to all parameters of the neural network.

- Test time: Predict parses **exacty as before**.

Critical Differences from Classical Model

- The arc represenations $\phi(x, i, j)$ are **learned!**
  - Word vectos e_x and POS vectors e_y are model parameters.

Critical Differences from Classical Model

- The arc represenations $\phi(x, i, j)$ are **learned!**
  - Word vectos e_x and POS vectors e_y are model parameters.

- They are learned **specifically** to decrease parsing loss.
  - Learn whatever representations that reduce this loss.

Critical Differences from Classical Model

- The arc represenations $\phi(x, i, j)$ are **learned!**
  - Word vectos e_x and POS vectors e_y are model parameters.
- They are learned **specifically** to decrease parsing loss.
  - Learn whatever representations that reduce this loss.
- The model is **nonlinear**.
  - Obvious advantage over linear models.

Additional Pieces

See Kiperwasser and Goldberg (2016) for details on

- **Stacking bidirectional LSTMs.** Run another round of forward/backward LSTMs.
- **Labeled parsing.** Add an additional feedforward to predict arc labels.
- **Transition-based neural parser.** Swap the representation of parser configuration with neural representations.