Natural Language Processing

Instructor: Ryan Cotterell

Lecture 1: Introduction to NL

NLP is the backbone of many tech companies: Siri, search engines, Alexa, etc.

The set of grammatical sentences is infinite even if we have a fine lexicon.
NL is not context-free (e.g., Swiss-German).

Linguistics: the structure of human language.
NLP: engineer systems to solve problems.

NLP is a set of methods and algorithms for making natural language accessible to computers.

Lecture 2: Backpropagation

Backpropagation is a linear-time dynamic program to calculate derivatives (not chain rule).
The chain rule: $\frac{\partial z_k}{\partial x_i} = \sum_j \frac{\partial z_k}{\partial y_j}\frac{\partial y_j}{\partial x_i}$.
From composite function to computation graph.
NP is linear in the number of edges.

Automatic Differentiation

1. Write composite function as hypergraph (a variable may be a function of more than one intermediate variable) with variables as nodes and hyperedges labeled with functions.
2. Perform forward propagation for a set of inputs to get the function value.
3. Run BP on the graph using stored forward values.

$$
\begin{aligned}
&\text{forward} - \text{propagate}(f, x \in \mathbb{R}^n)\text{:}\\
&\quad v_i \leftarrow \begin{cases} x_i \text{ if } i \le n \\ 0 \text{ otherwise} \end{cases}\\
&\quad \text{for } i = n+1, \dots, N\text{:}\\
&\quad\quad v_i \leftarrow p_i\big(\langle v_{Pa(i)}\rangle\big)\\
&\quad \text{return } [v_1, \dots, v_N]
\end{aligned}
$$

N is the number of nodes and Pa = Parent.

$$
\begin{aligned}
&\text{back} - \text{propagate}(f, x \in \mathbb{R}^n)\text{:}\\
&\quad v \leftarrow \text{forward} - \text{propagate}(f, x)\\
&\quad \frac{\partial f}{\partial v_i} \leftarrow 0, \forall i \in \{1, \dots, N\}\\
&\quad \text{for } i = N, \dots, 1\text{:}\\
&\quad\quad \frac{\partial f}{\partial v_i} \leftarrow \textstyle\sum_{j: i \in Pa(j)} \frac{\partial f}{\partial v_j}\frac{\partial}{\partial v_i} p_j\big(\langle v_{Pa(j)}\rangle\big)\\
&\quad \text{return } \left[\frac{\partial f}{\partial v_1}, \dots, \frac{\partial f}{\partial v_N}\right]
\end{aligned}
$$

we have a set of primitives and their derivatives.
Three types of differentiation:

- Symbolic: made by hand (calculations can be redundant).
- Numerical: the finite-difference approx. (so much slower).
- Automatic: backpropagation.

Lecture 3: Log-Linear Modeling (Meet the Softmax)

Random variables are about interactions between different properties of elements of sample space Ω (independence, correlation, etc.).
Example:
Sample Space Ω: set of all possible outcomes, e.g., $\Omega = \{1,2,3,4,5,6\}$ for a dice.
Event Space E: set of potential results of the experiment (set of subsets of Ω).
Probability Function: $p(e \in E) \in [0,1]$.

Log-linear Modeling

Inputs: $x \in \mathcal{X}$
Output label: $y \in \mathcal{Y}$
Feature function: $f: \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^K$
Parameters: $\theta \in \mathbb{R}^K$

$$p(y|x,\theta) = \frac{1}{Z(\theta)}\exp\big(\theta \cdot f(x,y)\big)$$

where $Z(\theta) = \sum_{y' \in \mathcal{Y}} \exp\big(\theta \cdot f(x,y')\big)$
Log-linear because $\log p(y|x,\theta) = \theta \cdot f(x,y) + C$

Feature Engineering: design f

- Preprocessing: tokenization, lower casing, stemming, stop word removal, etc.
- Feature Design: n-grams, one-hot encoding, bag of words, word embeddings, etc.

$$f(x,y) = \begin{pmatrix} \text{CountOf}(\text{money}, x) \wedge y = 1 \\ \text{CountOf}(\text{bank}, x) \wedge y = 1 \\ \dots \\ \text{CountOf}(\text{money}, x) \wedge y = 0 \\ \text{CountOf}(\text{bank}, x) \wedge y = 0 \end{pmatrix}$$

Estimating the parameters

Training Data: $\{(x_n, y_n)\}_{n=1}^{N}$

Log-likelihood: $L(\theta) = \sum_n \log p(y_n|x_n, \theta)$: convex

MLE estimation: $\theta_{MLE} = \arg\max_{\theta\in\Theta} L(\theta)$

The gradient of a Log-Linear Model

$\frac{\partial L}{\partial \theta_k} = \sum_n f_k(x_n, y_n) - \sum_n \sum_{y'} p(y'|x_n, \theta) f_k(x_n, y')$ (important).

= observed feature count − expected feature count.

Softmax

The default way of building probabilistic models using neural networks.

$\text{softmax}(h, y, T) = \frac{\exp\frac{h_y}{T}}{\sum_{y'} \exp\frac{h_{y'}}{T}}$ and $h_y = \theta \cdot f(x, y)$

Why Softmax?

$$\lim_{T\to 0} T \log\left[\exp\frac{x}{T} + \exp\frac{y}{T}\right] = \max(x, y)$$

Gradient of the Softmax

$$\log \text{softmax}(h, y) = h_y - \log \sum_{y'} \exp h_{y'}$$

$$\frac{\partial \log \text{softmax}(h, y)}{\partial h_i} = \delta_{yi} - \text{softmax}(h, i)$$

Exponential Family

A family of probability distribution (more general than softmax) of the form

$$p(x|\theta) = \frac{1}{Z(\theta)} h(x) \exp \theta \cdot \Phi(\mathrm{x})$$

$Z(\theta)$: the partition function

$h(x)$ determines the support

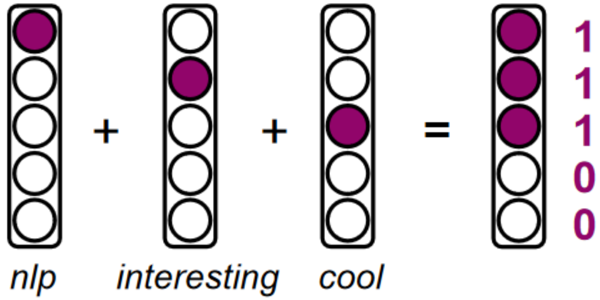
θ: the canonical parameters

$\Phi(x)$ are the sufficient statistics

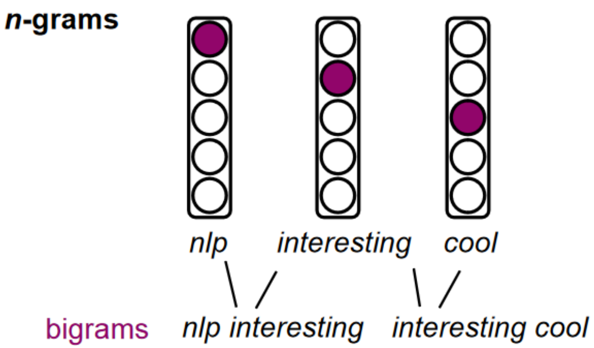
Lecture 4: Sentiment Analysis with Multi-layer Perceptrons

How to encode words?

- One-hot encoding



- N-grams



Skip-gram

<u>Preprocessing</u>: get pairs of word (w, c_w) for every word w and every context word of w, i.e., c_w. Context is a window of size k.

<u>The model</u>: $p(w|c) = \frac{1}{Z(c)} \exp\left(e_{wrd}(w) \cdot e_{ctx}(c)\right)$ where e is the embedding function.

<u>Estimation</u>: maximize the log-likelihood by computing the gradient wrt $e_{wrd}(w)$ and $e_{ctx}(w)$

$\sum_n \log p\left(w^{(n)}|c^{(n)}\right)$

$= \sum_n \left(e_{wrd}\left(w^{(n)}\right) \cdot e_{ctx}\left(c^{(n)}\right) - \log Z\left(c^{(n)}\right)\right)$

The output: two collections of word embeddings $\{e_{wrd}(w)\}_{w\in V}$ and $\{e_{ctx}(w)\}_{w\in V}$

<u>Evaluate Word Embeddings</u>

Cosine Similarity: $\cos(u_i, v_i) = \frac{u_i \times v_i}{\|u_i\| \times \|v_i\|}$

Sentiment Analysis

Sentiment Analysis is the NLP task of classifying utterances according to how they make the interlocutor feel.

term frequency–inverse document frequency: we look for words that are frequent in the considered document but not frequent in other documents.

SA Pipeline: embedding → pooling → softmax → backpropagation.

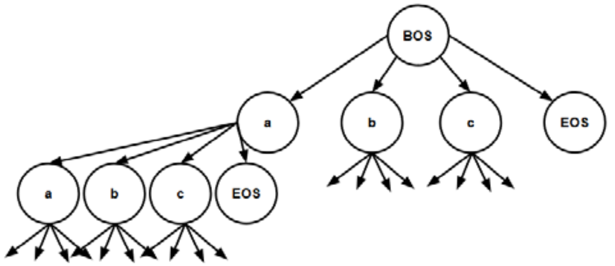
Lecture 5: Language Modeling with n-grams and RNNS

Structured Prediction

Predict structured objects (strings, trees) rather than scalar values ($|\mathcal{Y}| = 2^n$ for part-of-speech tagging!).

Given a vocabulary $V = \{a, b, c\}$, the task is modeling the distribution over sequences over V^* (all possible outputs, i.e., $\{a, b, c, aa, ab, ac, \ldots\}$). Without any prior assumption, $|V^*| \to \infty$.

A language model is a weighting of the prefix tree.



How to normalize?

$p(y) = \frac{1}{Z}\prod_{t=1}^{|y|} \theta_{y_{\leq t}}$ and $Z = \sum_{y' \in V^*} \prod_{t=1}^{|y'|} \theta_{y'_{\leq t}}$

Global Normalization: find an efficient algorithm to compute Z.

Local Normalization

Choose the weights θ strategically such that $Z = 1$: the probability of all children given their parent is 1.

Conditional Language Modeling

$$p(y|x) = \frac{\exp \text{score(y,x)}}{\sum_{y' \in V^*} \exp \text{score(y',x)}}$$

x can be source text (translation), signal (speech recognition), long text (summarization).

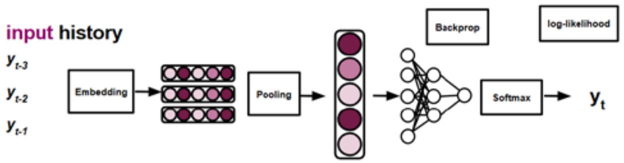
y is the target text.

n-gram Models

key idea: we enforce a finite number of histories to make modeling easier.

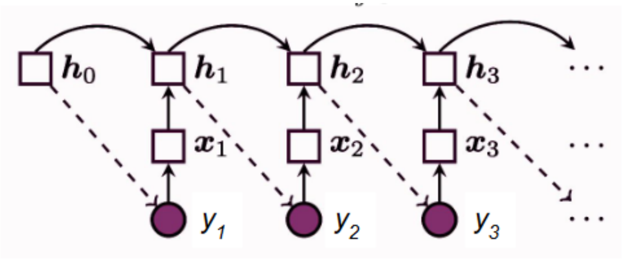
$$p(y_t|y_{<t}) = p(y_t|y_{t-1}, \dots, y_{t-n+1})$$

Condition on only the last $n-1$ words.



RNNs

y encodes the token and h for the entire context.



Backpropagation Through Time

Perform backpropagation after unfolding the network.

Exploding/vanishing gradient.

Lecture 6: Part-of-Speech Tagging

Assign each word in a sentence to a grammatical category.

Setup: score(t,w) where t is a tag sequence and w is a word sequence (sentence).

Condition Random Fields

$$p(t|x) = \frac{\exp \text{score(t,x)}}{\sum_{t' \in \mathcal{T}^N} \exp \text{score(t',x)}}$$

$N = |w|$: the length of the sentence → runs in $O(|\mathcal{T}|^N)$.

Score function (anything!)

Linear: $\text{score}(t, w) = \theta \cdot f(t, w)$

Non-linear: $\text{score}(t, w) = \text{NN}_\theta(t, w)$

To reduce computations, we assume a structure.

$\text{score}(t, w) = \sum_n \text{score}(< t_{n-1}, t_n >, w)$: bigram

Calculate the normalizer:

$\sum_{t_1 \in \mathcal{T}} \exp \text{score}(< t_0, t_1 >, w) \times \dots \times$

$\sum_{t_N \in \mathcal{T}} \exp \text{score}(< t_{N-1}, t_N >, w)$

$\beta(\mathbf{w}, t_N) \leftarrow 1$

for $n \leftarrow N-1, \dots, 0$:

$\quad \beta(\mathbf{w}, t_n) \leftarrow \sum_{t_{n+1} \in \mathcal{T}} \exp\{\text{score}\,\langle t_n, t_{n+1}\rangle, \mathbf{w})\} \times \beta(\mathbf{w}, t_{n+1})$

Semiring $R =< A, \bigoplus, \bigotimes, \bar{0}, \bar{1} >$

1. $(A, \bigoplus, \bar{0})$: commutative monoid.
2. $(A, \bigotimes, \bar{1})$: monoid (no inverse).
3. $\bigotimes$ distributes over $\bigoplus$.
4. $\bar{0}$ is an annihilator.

CRF as Softmax

To estimate the parameters, we maximize the log-likelihood:

$$\sum \left(\text{score}(t^{(i)}, w^{(i)}) - T \log \sum_{t' \in \mathcal{T}^N} \exp \frac{\text{score}(t', w^{(i)})}{T} \right)$$

When $T \to 0$, we get:

$$\sum \left(\text{score}(t^{(i)}, w^{(i)}) - \max_{t' \in \mathcal{T}^N} \text{score}(t', w^{(i)}) \right)$$

<u>Lexical semantics</u> is the study of meaning of words.

<u>Compositional semantics</u> is the study of the meaning of utterance (neutral term for chunk of word).

Meaning: we know the meaning of an utterance u iff we know all the situations where us is true.

Skip-gram is a probabilistic model $p(w|c) = \frac{1}{Z(c)}\exp\{e[w]^T e[c]\}$ to predict a word given its context, where $Z(c) = \sum_{w' \in V} \exp\{e[w']^T e[c]\}$.

Skip-gram objective: $\sum_n e[w_n]^T e[c_n] - \log Z(c_n)$
A word $w \in V$, is the concatenation $[e[w]; e[w]]$, i.e., word and context vector.

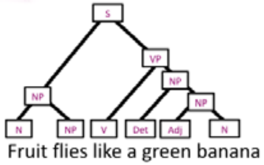
Lecture 7: Context-Free Parsing with CKY

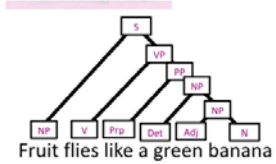
Syntactic Constituency

The mathematical study of structure of sentences (word order).
Constituent: multiple words functioning as a unit
How to check if a set of words is a constituent? Pronoun substitution …

Context-Free Grammars

Grammar: rules to describe how to form sentences from words.
Context-free grammar: rules can be applied regardless of the context.
Example: every node is a constituent





Probabilistic CFGs: assign a probability to each production locally normalized).

The Parsing Problem

Get a tree given a sentence.
The probability of a tree given a sentence:

$$p(t|s) = \frac{1}{Z(s)}\exp \operatorname{score}(t)$$

$$Z(s) = \sum_{t' \in \mathcal{T}(s)} \exp \operatorname{score}(t')$$

Where $\mathcal{T}(s)$ is the set of trees that yields s.

Chomsky Normal Form

- $N_1 \rightarrow N_2 N_3$ where N_i are non-terminals.
- $N \rightarrow a$ where a is terminal.

$\rightarrow$ a finite number of trees given a sentence s.

The CKY Algorithm

An efficient dynamic program to compute the normalizer of the parser in a CNF.
(see slides for algorithm)

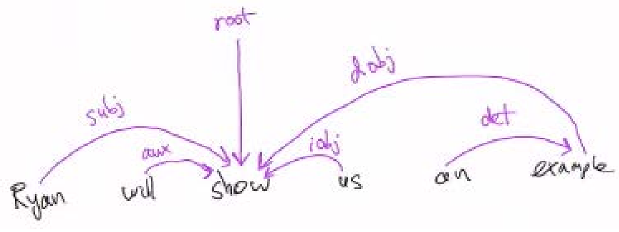
Lecture 8: Dependency Parsing with the Matrix-Tree Theorem

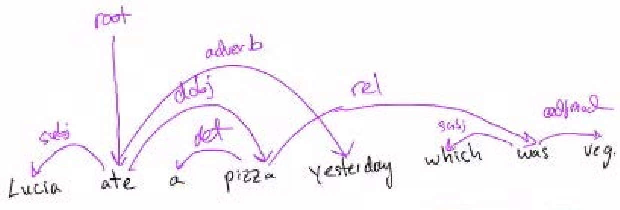
Dependency Parsing

Dependency Grammar is an alternative to constituency grammar: link every word with its syntactic head.

Dependency Parsing: construct a tree relating words with syntactic relations: directed & labeled

Projective Trees: no overlapping arcs.
Non-Projective Tree: overlapping arcs.





Probability Distributions over Non-Projective Trees

Now $\mathcal{T}(w)$ is non-projective spanning tree set $\rightarrow O(n^n)$.

For simplicity, the edge-factored scoring function is used, where $(i \rightarrow j)$ is an edge:

$$p(t|w) = \frac{1}{Z}\prod_{(i\rightarrow j)\in t} \exp \operatorname{score}(i,j,w) \exp \operatorname{score}(r,w)$$

$$Z = \sum_{t' \in \mathcal{T}(w)} \prod_{(i\rightarrow j)\in t'} \exp \operatorname{score}(i,j,w) \exp \operatorname{score}(r,w)$$

Adjacency Matrix

$$A_{ij} = \exp \operatorname{score}(i,j,w)$$

$$\rho_j = \exp \operatorname{score}(j,w)$$

Number of undirected spanning trees: $Z = \det L$

$$L = D - A$$

$$L_{ij} = \begin{cases} -A_{ij} \text{ if } i \neq j \\ \sum_{k \neq i} A_{kj} \text{ otherwise} \end{cases}$$

Revisited

$$\boxed{L_{ij} = \begin{cases} \rho_j & \textbf{if } i = 1 \\ \sum\limits_{i'=1, i' \neq j}^{n} A_{i'j} & \textbf{if } i = j \\ -A_{ij} & \textbf{otherwise} \end{cases}}$$

Decoding Non-Projective Trees

Find maximum-weight spanning tree

$$\arg\max_{t \in \mathcal{T}} \sum_{(i \to j) \in t} \text{score}(i, j, w)$$

Kruskal's Algorithm: add the highest-score edge that does not create a cycle: $O(E \log E)$.

Lecture 9: Transliteration with WFSTs

Map strings between character sets.
Weighted Finite-State Transducers: there are a finite number of states in our model of language.
Weighted: transition probabilities.

Construct a conditional distribution $p(y|x)$. Its structure is given by a WFST T:

$$\text{score}(\pi) = \sum_{n=1}^{|\pi|} \text{score}(\tau_n) = \sum_{n=1}^{|\pi|} w(\tau_n)$$

where π is a path and τ is a transition.

Decompose Score Function

$$p(y|x) = \frac{1}{Z} \exp \text{score}(y, x)$$
$$= \frac{1}{Z} \sum_{\pi \in \Pi(x,y)} \exp \sum_{n=1}^{|\pi|} \text{score}(\tau_n)$$

In an unambiguous WFST, the first sum can be dropped because we have 0 or 1 path.
The normalizer: $Z = \sum_{y' \in \Omega^*} \exp \text{score}(y', x)$.
How to compute?

Floyd-Warshall Algorithm

Find shortest paths in a weighted graph with positive or negative edge weights.

Pseudocode
for each vertex v:
  $\text{dist}_{vv} = 0$
for k,i,j:
  if $d_{ij} > d_{ik} + d_{kj}$:
    $d_{ij} \leftarrow d_{ik} + d_{kj}$

Generalization to any semiring

```
let dist be a N × N array of minimum distances initialized to 0 (infinity)
for each edge (u, v) do
  dist[u][v] ← W[u][v]  // This corresponds to W¹
for each vertex v do
  dist[v][v] ← W[v][v] // This corresponds to W⁰
for k from 1 to N
  for i from 1 to N
    for j from 1 to N
      dist[i][j] ←dist[i][j] ⊕ (dist[i][k] ⊗ dist[k][k]* ⊗ dist[k][j])
```

$$Z = \alpha^T \left(\sum_{\omega \in \Omega \cup \{\epsilon\}} W^{(\omega)} \right)^* \beta$$

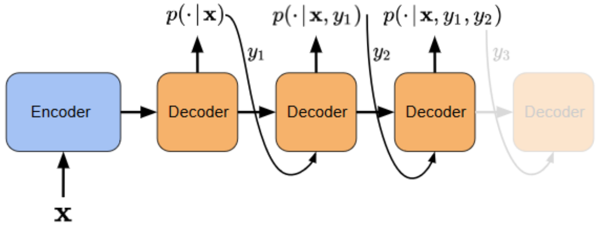
Lecture 10: Machine Translation with Transformers

Sequence-to-Sequence Models

Model the probability distribution $p(y|x)$: what's the most likely translation y of x.

$$p(x|y) = \prod_{t=1}^{T} p(y_t | x, y_1, \dots, y_{t-1})$$

Inference



The Attention Mechanism

Use different context vector to represent the input sequence depending on where we are in output generation.

___ FIN ___