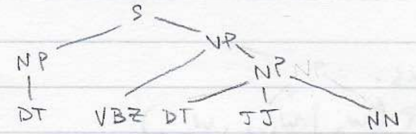
to revise:
- notes
- labs
- revision guide
- Piazza

FNLP
final exam notes

(1) <u>intro</u>

- some terminology:



} syntax

→ part of speech

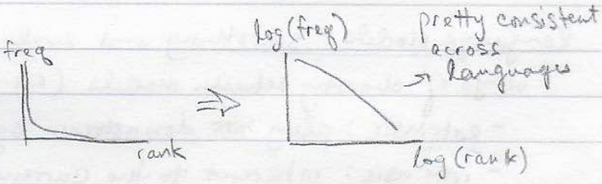
This is a simple sentence → words

be, 3sg, present | SIMPLE1 having few parts | SENTENCE1 string of words satisfying the grammatical rules of a language → morphology, semantics

But it is an instructive one → discourse

why is NLP hard?

- NLP is hard b/c of ambiguity
  - word senses: bank (finance or river?)
  - part of speech: chair (noun or verb?)
  - syntactic structure: I saw a man with a telescope
  - quantifier scope: Every child loves some movie
  - multiple: I saw her duck.
- data is sparse due to Zipf's law: freq vs. rank



  $f \cdot r \approx k$ where f = freq of word, r = rank (by f), k = constant

  $\Rightarrow \log f = \log k - \log r$ $\quad y = b + mx$
- variation: think WSJ vs. social media
- expressivity: can express same thing many different ways
- context dependence and unknown representation: need world knowledge (sometimes/usually), don't know how to represent "meaning"

(2) <u>text corpora</u>

- good if naturally-occurring, includes metadata, or has linguistic annotations
- for sentiment analysis, good to have some sort of ratings/labels
- problems w/ counting positive/negative words: sense ambiguity, sarcasm/irony, (other) context
- tokenization: adding logical boundaries between separate word/punctuation tokens not already separated by spaces
  - only one part of preprocessing/normalization
- "domain" of corpus includes mode of communication (e.g., speech, writing), topic (e.g., politics, physics), genre (e.g., news, tweet), audience (formality, politeness, complexity)
- domain adaption corrects for when training data and test data don't come from the same source

(3) <u>n-gram language models</u>

- this lecture talks about sentence probabilities
- language model gives approx. for true prob. of an arbitrary sequence of words
- uses of a language model: spelling correction, automatic speech recognition, machine translation
  - in each case, get possibilities from some model (e.g., error model, acoustic model, translation model) and use language model to choose among alternatives
- unrealistic to do MLE on complete sentences; instead, multiply probabilities of sub-parts
- n-gram model:

$$P(w_1, \dots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_1, w_2, \dots, w_{i-1}) \approx P(w_i \mid w_{i-2}, w_{i-1}) \text{ for trigram model}$$
$$\approx P(w_i \mid w_{i-1}) \text{ for bigram}$$
$$\approx P(w_i) \text{ for unigram}$$

- estimation w/ trigram model:

$$P_{MLE}(w_i \mid w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})}$$

n-gram language models (cont'd)

- practical implementation of tri-grams:

$$P(\vec{w}) = P(w_1)P(w_2|w_1)\prod_{i=3}^{n} P(w_i | w_{i-2}, w_{i-1})$$

  - but this doesn't capture the fact that some word strings are more/less likely to start/end a sentence
  - can assume $w_{-1} = w_0 = \langle s \rangle$ and $w_{n+1} = \langle /s \rangle$:

$$P(\vec{w}) = \prod_{i=1}^{n+1} P(w_i | w_{i-2}, w_{i-1})$$

  - typically use negative log probs since probs can get small

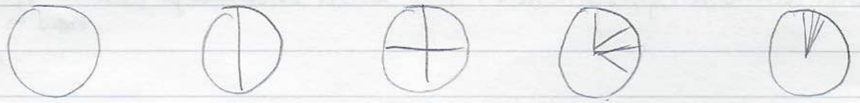
(4)

language models: smoothing and evaluation

- ways of choosing between models (e.g., bigram and trigram)
  - extrinsic: plug into downstream system and (somehow) evaluate that system
  - intrinsic: inherent to the current task; usually quicker/easier
- one intrinsic measure: entropy:

$$H(x) = \sum_x -P(x)\log_2 P(x) = \mathbb{E}[-\log_2 P(X)]$$

  - this measures uncertainty / disorder / surprise



$H(x)=0$ $H(X)=1$ $H(x)=2$ $H(X)=1.4$ $H(X)=0.2$ → or, # yes/no questions

  - intuition: average number of bits needed to encode $X \geq$ entropy of X
- a good model assigns high probability to sequences that actually have high probability; put another way, our model should have low uncertainty (entropy) about which word comes next
  - ⇒ this can be measured using cross-entropy
  - note: cross-entropy $\geq$ entropy
  - computing per-word cross entropy: $H_M(w_1, \ldots, w_n) = -\frac{1}{n}\log_2 P_M(w_1, \ldots, w_n)$ ← average across words
- perplexity: $2^{(\text{cross-entropy})}$
  - interpretation: the average branching factor at each decision point, if our dist. were uniform
- the "goodness" of different values of these measures depends on the corpus. e.g., if corpus is "meow, meow, meow, ..." then we'd hope cross-entropy would be very low (0).
- smoothing: fixes flaw of MLE, which is that it estimates probs that make training data maximally probable by making everything else (unseen data) minimally probable
  - add-one (Laplace) smoothing: just pretend you've seen everything one more time than you did

$$P_{+1}(w_i | w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i) + 1}{C(w_{i-2}, w_{i-1}) + V} \quad \text{where } V = \text{vocab size}$$

    - large vocabs make it so that way too much prob mass is stolen from seen events
  - add-α (Lidstone) smoothing improves things:

$$P_{+\alpha}(w_i | w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i) + \alpha}{C(w_{i-2}, w_{i-1}) + \alpha V} \quad \text{where } \alpha < 1$$

    - to find α: test α on different validation sets to see which gives lowest cross-entropy

FNLP
final exam notes

n-gram language models

- good-Turing
  - previous methods change the denominator, which can have big effects on frequent events; Good-Turing changes the numerator
    - usually, we have $P_{ML} = \frac{c}{n}$ → # times you see n-gram

    but Good-Turing uses adjusted counts c^*: $P_{GT} = \frac{c^*}{n}$ where $c^* = (c+1)\frac{N_{c+1}}{N_c}$,

    N_c = # n-grams that appear exactly c times in corpus

    N_0 = # unseen n-grams

  - summary:
    - add-1 and add-α are simple, but not very good
    - Good-Turing more sophisticated and better, but there are even better ones

more smoothing and the noisy channel model

- Good-Turing says these have same prob: "Scottish beer drinkers" and "Scottish beer eaters"
- solution: use info from lower-order N-grams "beer drinkers" and "beer eaters"
- two ways: interpolation and backoff
- interpolation: combine lower order N-gram models since they have different strengths/weaknesses
  - just a weighted average of probs from different models
  - called a mixture model
  - weights λ_i are interpolation parameters or mixture weights
- back-off: trust the highest order language model that contains N-gram
- but diversity of histories matters: think P(York | New) is high but P(York | !New) is low
- Kneser-Ney smoothing takes diversity of histories into account
  - replace raw counts with counts of histories:

$$P_{ML}(w_i) = \frac{c(w_i)}{\sum_w c(w)} \Rightarrow P_{KN}(w_i) = \frac{N_{1+}(\bullet w_i)}{\sum_w N_{1+}(\bullet w)}$$

$$\text{where } N_{1+}(\bullet w_i) = \left|\{w_{i-1} : c(w_{i-1}, w_i) > 0\}\right|$$

  - best smoothing for n-grams
- quick summary:
  - uniform probs: add-α, Good-Turing
  - probs from lower-order n-grams: interpolation, backoff
  - prob of appearing in new contexts: Kneser-Ney
- now dealing with word similarity, e.g., knowing that P(salmon | caught two) tells us something about P(swordfish | caught two)
- can use embeddings
- noisy channel model: $\underset{\text{language model}}{P(Y)} \Rightarrow \underset{\text{noise model}}{P(X|Y)} \Rightarrow \underset{\text{resulting dist}}{P(X)}$

Application	Y	X
Speech recog.	spoken words	acoustic signal
Machine translation	words in L_1	words in L_2
Spelling correction	intended words	typed words

More smoothing and the noisy channel model (cont'd)

- we want $\arg\max_y P(y|x)$

$$\arg\max_y P(y|x) = \arg\max_y \frac{P(x|y)\,P(y)}{P(x)} = \arg\max_y P(x|y)\,P(y)$$

Spelling correction, edit distance, and EM

- spelling correction:
  - we don't know $P(y)$, so need to make constraints:
    - words must differ by one character from x
  - then can compute $P(x|y)\,P(y)$
  - assume substitutions, e.g., o → e, has one prob instead of a prob conditional on context (surrounding letters)

    so:

$$P(x|y) = \prod_{i=1}^{n} P(x_i|y_i)$$

$$\text{e.g., } P(no|not) = P(n|n)\,P(o|o)\,P(-|t)$$

data needed: subs probs

  - can estimate subs probs with data (confusion matrix)
- alignments and edit distance
  - want to find optimal character alignment b/t two words (fewest changes: min edit distance MED)

```
STALL
TALL
TABL
TABLE
```

MED(stall, table) = 3

also: (as alignment)

```
STALL -
d | | s | i
- TABLE
```

- exponential number of possibilities: how to choose?
- use dynamic programming (memoization) algorithms: Viterbi, CKY
- intuition: D(stall, table) must be min of:
  - D(stall, tabl) + cost (ins)
  - D(stal, table) + cost (del)
  - D(stal, tabl) + cost (sub)

  assume cost (ins) = cost (del) = 1, cost (sub) = 2
- chart stores two things: MED and backpointers

fill like this: start ↓↑↓↑↓↑ etc.

		T	A	B	L	E
	0	←1	...			
S	↑1	↖↑2	...			
T	↑1	↖1				
A	↑1				⋮	⋮
L	↑1	⋮		...	↖2	←3
L	↑1			...	↖3	↖↑4

cost of going from S to empty string

cost of going from ST to empty string

cost of going from S to T

min cost: follow arrows to get possible paths back

- chicken-and-egg: want to use costs from noise model, but need costs to estimate noise model
- solution: EM algorithm
  1. initialize params to arbitrary values (e.g., all costs = 1)
  2. using these params, compute optimal values (run MED to get alignments)
  3. using alignments, recompute params
  4. repeat 2 and 3 until params stop changing

FNLP
final exam notes

spelling correction, edit distance, and EM (cont'd)

- EM just described for spelling correction is "hard" EM, which converges (like soft EM) but is not nicely defined mathematically (doesn't converge to optimum of likelihood function like soft EM does); but it probably works fine in practice (easier to compute)

  to something, i.e., doesn't diverge or oscillate
- likelihood function:

$$P(\text{data} \mid \theta) = \prod_{i=1}^{n} P(x_i \mid y_i) = \overbrace{\sum_a \prod_{i=1}^{n} P(x_i \mid y_i, a)}$$

  if alignments a are latent
- neither hard nor soft EM guaranteed to converge to global optimum; soft maybe not even local

text classification

- have y values like spam/not spam, sentiment, topic
- bag of words models: word order doesn't matter
- Naive Bayes

$$\hat{c} = \underset{c \in C}{\arg\max}\ P(\underset{\text{category}}{c} \mid \underset{\text{document}}{d}) = \underset{c \in C}{\arg\max}\ P(d \mid c) P(c)$$

  likelihood

  model $P(d|c)$ as set of features (words) it contains: $P(d|c) = P(f_1, f_2, \dots, f_n | c)$

$$\approx P(f_1|c)P(f_2|c)\dots P(f_n|c)$$

  (naive Bayes assumption)

  prior

  so: $$\hat{c} = \underset{c \in C}{\arg\max}\ P(c) \prod_{i=1}^{n} P(f_i | c)$$

  - can estimate $P(f_i|c)$ with simple smoothing
  - alternative feature values and feature sets:
    - use only binary values for f_i
    - only use subset of vocab for F
    - use more complex features (e.g., bigrams, syntactic features)
    - for sentiment analysis: whether word is +/-

  data: need features (can be extracted?)
  - quick and easy model
  - but independence assumption may be strong/unrealistic
- MaxEnt better if you have enough data
  - a.k.a. multinomial logistic regression

$$\hat{c} = \underset{c \in C}{\arg\max}\ P(c|d) \quad \text{(same as NB)}$$

  ... but we model $P(c|d)$ directly
  - features are functions of both observations $\vec{x}$ and class c

$$P(c|\vec{x}) = \frac{1}{Z} \exp\left(\sum_i w_i f_i(\vec{x}, c)\right)$$

  (logistic)

  ⇒ as if just running c separate regressions... and using softmax
  - training:

$$\hat{w} = \underset{\vec{w}}{\arg\max} \sum_j \log P(c^{(j)} | x^{(j)})$$

  can take some time (gradient ascent): conditional MLE (CMLE)

Part-of-speech tagging and HMMs

- first step toward syntactic analysis
- deciding on correct label depends on word to be labeled and labels of surrounding words
- parts of speech: two main classes: open class ("content" words) and closed class (function words)
- commonly 40-100 different tags
- hard b/c of ambiguity, sparsity
- relationship to previous models:

	sequences	hidden vars
n-gram	✓	×
NB	×	✓
HMM	✓	✓

  - n-gram model: a model for sequences that also makes a Markov assumption, but has no hidden vars
  - Naive Bayes: a model w/ hidden vars (classes) but no sequential dependencies
  - HMM: a model for sequences with hidden variables
- HMM:

$$P(T) = \prod_i P(t_i \mid t_{i-1})$$

$$P(S \mid T) = \prod_i P(w_i \mid t_i)$$

$$\arg\max_T \; p(T \mid S) = \arg\max_T \; p(S \mid T)\, p(T)$$

$$\text{where } \underbrace{P(S \mid T)\, p(T)}_{= P(S,T)} = \prod_i P(w_i \mid t_i)\, P(t_i \mid t_{i-1})$$

  - too many tag sequences T to enumerate

    ⇒ use Viterbi again (see next lecture)

algorithms for HMMs

- change of notation:
  - A: transition matrix w/ possible states $q^1, q^2, \dots, q^N$ for N tags
  - B: output matrix w/ possible outputs $o^1, o^2, \dots, o^V$ for V words
  - $\lambda = (A, B)$: params of HMM
  - Q: state sequence $q_1, q_2, \dots, q_T$
  - O: output sequence $o_1, o_2, \dots, o_T$

    } T: time
- using new notation: (w_i, t_i, t_i, t_{i-1} ← old notation; S, T)

$$P(O, Q \mid \lambda) = \prod_{t=1}^{T} P(o_t \mid q_t)\, P(q_t \mid q_{t-1})$$

$$= \prod_{t=1}^{T} b_{q_t}(o_t)\, a_{q_{t-1} q_t}$$

- example grid:

	$o_1 = x$	$o_2 = z$	$o_3 = y$
q^1	0.6	0.126	0.00882
q^2	0	$\boxed{0.036}$	0.02646

best: $q^1 q^1 q^2$

$$v(2,2) = \max \begin{cases} v(1,1) \cdot a_{12} \cdot b_2(z) \\ v(2,1) \cdot a_{22} \cdot b_2(z) \end{cases}$$

value from q^1_{t-1}; transition from q^1 to q^2; output prob

- implementation / efficiency:
  - enumeration is $O(N^T)$, $O(T)$ space
  - Viterbi is $O(N^2 T)$, $O(NT)$ space (much better on run time)
  - use neg log probs
- may also want to compute likelihood: $P(O \mid \lambda) = \sum_Q P(O, Q \mid \lambda)$
  - use Viterbi (w/ a slightly different update), sum last column for $P(O \mid \lambda)$
- learning $\lambda = (A, B)$: use EM if unsupervised (no tag labels), using expected counts in E step

  → i.e., a language model $P(q^j \mid q^i) = \dfrac{c(q^i \to q^j)}{c(q^i)}$
- forward-backward algo: avoid enumerating all possible sequences ⇒ an instance of EM

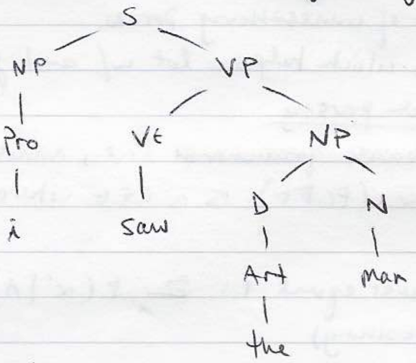
FNLP
final exam notes

Methods in annotation and evaluation

- Annotation not easy, even for humans because of ambiguity and gray areas (both syntax and semantics)
- Penn Treebank has 36 POS tags (excluding punctuation) w/ extensive guidelines for annotators
  - even so, there will still be errors
- inter-annotator agreement (IAA)
  - have multiple people annotate, compute raw agreement rate
  - very low rates ⇒ examine sources of disagreement and consider rewriting guidelines
  - agreement rate can be considered upper bound (human ceiling) on accuracy of a system evaluated on that dataset
  - can also weight errors by severity
- can make hypotheses about linguistic objects (e.g., author), output of a language system, and human beings
- can use cross-validation to choose hyperparams (e.g., for smoothing)
- measures of performance:
  - precision: $P = \frac{TP}{TP+FP}$ ⇒ fixes for cases where you over-predict 1 when there are a lot of 1's
  - recall: $R = \frac{TP}{TP+FN}$ ⇒ fixes for cases where you over-predict 0 when there are few 0's
  - F_1-score: $F_1 = \frac{PR}{P+R}$
- use significance tests to determine whether measurements are different

Syntax and parsing

- theory of syntax should explain which sentences are well-formed (grammatical); well-formed distinct from meaningful
  - two theories: context-free grammar and dependency grammar
- context-free grammar (CFG)
  - two types of grammar symbols: terminals (t): words; and non-terminals (NT): phrasal categories like S, NP, VP, PP
  - has rules of the form NT → β, where β is any string of NT's and t's
  - e.g.,



  - structural ambiguity: caused by POS ambiguity; e.g., I saw her duck
  - attachment ambiguity: e.g., I saw the man with the telescope.
- all parsers have two fundamental properties
  - directionality: sequence in which structures are constructed (top-down, bottom-up, mixed)
  - search strategy: the order in which the search space of possible analyses is explored: depth-first, breadth-first, best-first
  - example: the dog bit (recursive descent strategy) → top-down, depth-first

step	op	subgoals	input
0		S	the dog bit
1	E	NP VP	"
2	E	DT NN VP	"
3	E	the NN VP	"
4	M	NN VP	dog bit
5	E	bit VP	"
6	B4	NN VP	"

operations:
E = expand
M = match
Bn = backtrack to step n

- problems: can be many backtracks or infinite loops (NP → NP PP)

data: need POS tags and rules

operations:
S = shift
R = reduce
Bn

step	op	stack	input
0			the dog bit
1	S	the	dog bit
2	R	DT	dog bit
3	S	DT dog	bit
4	R	DT V	bit
5	S	DT V bit	
6	R	DT V V	
7	BS	DT V bit	

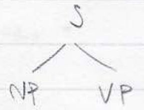
<u>Syntax and parsing</u> (cont'd)

- shift-reduce: depth-first, bottom-up
  - e.g., ⟶
- depth-first parsers very efficient for unambiguous structures but massively inefficient when faced with local ambiguity
  - but can use probabilistic model to learn which choices to make (next lesson...)
- breadth-first search using dynamic programming
  - avoid re-analyzing any substring because its analysis is independent of the rest of the parse
  - memoized in chart, a.k.a., well-formed substring table (WFST).
  - CKY algorithm: bottom-up, breadth-first
    - takes $O(Gn^3)$, where G is the number of grammar rules

	1	2	3	4
0	Pro, NP ①			S ⑤
1		Vt, Vp, N	②	VP ④
2			Pro, PosPro, D	NP ③
3				N, Vi

$_0$ he $_1$ saw $_2$ her $_3$ duck $_4$

① NP → Pro
② D → PosPro
③ NP → D N
④ VP → Vt NP
⑤ S → NP VP



- start with all POS allowed for word
- ignore lower diagonal: would be backward strings
- to fill cell (i,j), we use cell from row i and a cell from column j
  ⇒ so need to fill cells down and left of (i,j) before filling (i,j)
- even though avoids re-computing substructures, much more efficient than depth-first (in worst case)
- but still may be a lot of unnecessary parses.
- next: statistical parsing, which helps a lot w/ ambiguity and efficiency

<u>CKY parsing, treebanks and Statistical parsing</u>

- for probabilistic parsing, use treebank grammars (i.e., annotated sentences)
- a probabilistic context-free grammar (PCFG) is a CFG where each rule $A \to \alpha$ is assigned a probability $P(\alpha | A)$
  - sum over expansions of A must equal 1: $\sum_{\alpha'} P(\alpha' | A) = 1$
  - use MLE for probs (w/ smoothing)
  - a <u>generative model</u>, like with HMMs
  - prob of parse t is product of all rules of parse: $P(t) = \prod_{A \to \alpha \in t} A \to \alpha$

    i.e. each non-terminal node has a prob; just multiply these together
  - we also have a <u>language model</u> since t implicitly includes words $\vec{w}$: $P(t) = P(t, \vec{w})$
    - sentence prob is then obtained by summing over $T(\vec{w})$, the set of valid parses of $\vec{w}$:

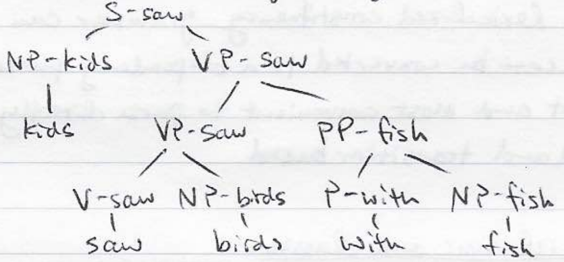
$$P(\vec{w}) = \sum_{t \in T(\vec{w})} P(t, \vec{w}) = \sum_{t \in T(\vec{w})} P(t)$$

  - straightforward to extend CKY parsing to probabilistic case
    - goal: return highest prob parse of sentence (analogous to Viterbi)
  - best-first parsing help by not having to do exhaustive parsing
    - constituents have <u>scores</u> and are added to <u>agendas</u>, which are ordered by scores
    - scores computed as an average over words to normalize tree sizes
- not-so-great consequence: words w/ same POS will give same scores / probs even though we know this isn't true

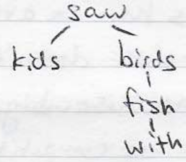
FNLP
final exam notes

heads, dependency parsing

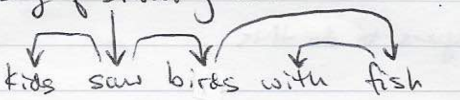
- ways to fix PCFGs
  1. automatically create new categories that include the old category as the ~~parent~~ parent
     - parent annotation → e.g., an NP in subject position becomes NP^S
  2. lexicalization: create new categories by adding the lexical head of the phrase
     - e.g.,



- but this leads to huge grammar blowup and very sparse data
- evaluating parse accuracy
  - output considered correct if there's a gold constituent that spans the same sentence positions
  - use precision / recall / F_1 score
- dependencies
  - transforming constituency ⇒ dependency parse
    - start w/ lexicalized constituency parse, remove phrasal categories, remove duplicated terminals, and collapse chains of duplicates
    - so the lexicalized constituency parse above becomes:



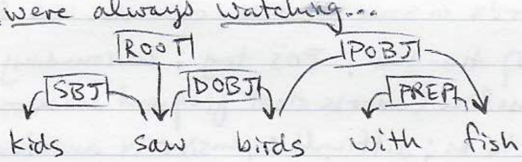
    - another way of showing this:



head → modifier

every word has exactly one parent

  - some treebanks prefer content heads, while others functional heads
    - e.g., ...were always watching... (functional: were; content: watching)
  - edge labels:



  - projectivity: parse said to be projective if every subtree (node and all its descendents) occupies a contiguous span of the sentence
    - i.e., no crossing edges
      - nonprojectivity is rare in English, but quite common in other languages
  - constituency → dependency parse (again): but how do we find each phrase's head in the first place?
    - use head rules: designate one RHS nonterminal as containing the head
- direct dependency parsing:
  - transition-based parsing
    - adapts shift-reduce methods: stack and buffer (note: shift-reduce more efficient than CKY)
    - idea: train classifier to predict next action (SHIFT, REDUCE, ATTACH-LEFT, ATTACH-RIGHT), and proceed left-to-right through sentence. O(n) time complexity!
    - only finds projective trees
  - graph-based parsing
    - global algorithm: from the fully connected directed graph of all possible edges, choose the best ones that form a tree
    - e.g., maximum spanning tree algorithm ($O(n^2)$)

heads, dependency parsing (cont'd)

- conversion-based parser: first constituency-parse, then convert to dependencies
- Summary:
  - while constituency parses give hierarchically nested phrases, dependency parses represent syntax with trees whose edges connect words in a sentence
  - head rules govern how a lexicalized constituency grammar can be extracted from a treebank, and how a constituency parse can be converted to a dependency parse
  - for English, often fastest and most convenient to parse directly to dependencies; two main paradigms: graph-based and transition-based

lexical semantics: word senses, relations, and classes

- why it's hard: different senses, synonyms (vacation/holiday), hyponym/hypernym (subset/superset) (animal, polar bear), similarity and gradation (good and great), need for inference
- WordNet a hand-built resource w/ 117,000 synsets: sets of synonymous words
  - although this doesn't solve issues w/ homonyms (bank: river/finance), polysemes (multiple senses)
  - synsets organized into a network by several kinds of relations, including:
    - hypernymy (is-a): hyponym {ambulance} is a kind of hypernym car
    - meronymy (part-whole): meronym {air bag} is a part of holonym car
  - nouns have on average 1.24 senses, verbs 2.17
  - incomplete: doesn't have all multiword phrases (stress out), neologisms (facepalm), names (Microsoft)
- word sense disambiguation (WSD)
  - there are competitions held every 1-3 years to do this
  - data-driven methods do well
    - e.g., Naive Bayes, decision lists, decision trees
    - as features, can use content words in some window; can also use syntactically related words, syntactic role in sense, topic of the text, POS tag, surrounding POS tags
  - evaluation: extrinsic, intrinsic, baseline (choose most frequent sense - sometimes hard to beat)
  - issues: not clear how fine-grained to be; difficult expensive to annotate w/ fine-grained; classifiers must be trained separately for each word
- named entity recognition (for extracting proper names) and supersense tagging: categories like person, location, artefact
  - supersenses (e.g., attribute, food, object, quantity, motion, social) are language-neutral

distributional lexical semantics

- distributional hypothesis: a word is the company it keeps
- use word-context matrices (like in NLU): each word is a vector of context word frequencies/probs
- two kinds of co-occurence between two words: first order (near each other), second order (similar neighbors)
- for syntactic similarity, use small context windows; for semantic, use larger (sometimes whole document)
- collocation: occuring unusually often in the context of word w
  - can use $PMI(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$
    - b/c uses MLE, it's over-sensitive to chance co-occurence of infrequent words
  - alternatives: t-test, χ^2, others...
  - or can improve PMI: positive PMI (PPMI) = max(PMI, 0)
- measuring similarity: cosine: $\cos(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|}$
- sparse → dense: SVD/LSA, skip-grams, etc.
  - e.g., brown clusters: binary tree

FNLP
final exam notes

distributional lexical semantics (cont'd)

- class-based language model: $P(w_i \mid w_{i-1}) = P(c_i \mid c_{i-1}) P(w_i \mid c_i)$

Semantic role labeling and argument structure

- can't just use syntax to trivially infer semantic roles
- two computational datasets/approaches that describe sentences in terms of semantic roles:
  - PropBank: simpler, more data
  - FrameNet: richer, less data
- PropBank has structure and its roles (Arg0, Arg1, ...)
  - abstracts away from syntax to predicate-argument structures
  - lexicon and annotations on full WSJ PTB corpus and other data
  - limitation: roles (Arg0, ...) are predicate-specific
  - originally verbs only, but now has other POS
- FrameNet
  - no longer (primarily) verb-based, but frame-based
    like PropBank
  - FrameNet parsing: SEMAFOR
    - pipeline similar (but not identical) to traditional SRL

machine translation [non-examinable]