CS4248 Finals Cheatsheet v1.0 (2020-11-22)
by Aadit Rahul Kamat, page 1 of 2

1 Regular Expressions and Finite State Automata

1.1 Finite State Automaton
- Formal definition: (Q, Σ, q_0, F, $\delta(q,i)$) where:
  - Q: set of states
  - σ : input alphabet
  - q_0 : the start state
  - F: the set of final state
  - $\delta(q,i)$: transition function between the states

1.2 Deterministic Finite State Automaton
- Accepts an input string if we run out of input and it is in an accepting state
- Exactly one transition for a given symbol from one state to another

1.3 Non-Deterministic Finite State Automaton
- Accepts an input string if there is at least some path to an accepting state that exhaust the input string
- Could have more than one transition for the same input symbol or transition to another empty state given an empty symbol

1.4 Convert from NFSA to DFSA
- Keep track of all states that can be transitioned from NFSA
- A state in converted DFSA denotes a combination of states in NFSA

2 Words, Spelling Errors and Edit Distance

2.1 Porter Stemming Algorithm
- ATIONAL -> ATE
- ING -> ϵ
- SSES -> SS

2.2 Bayesian Classification

$$\hat{c} = argmax_{c \in C} P(c|o) = argmax_{c \in C} \frac{P(o|c).P(c)}{P(o)} = argmax_{c \in C} P(o|c).P(c)$$

where P(o|c) is the likelihood and P(c) is the prior

2.3 Minimum Edit Distance

```
function MIN-EDIT-DISTANCE(source, target) returns min-distance
  m ← LENGTH(source)
  n ← LENGTH(target)
  Create a matrix distance[m + 1, n + 1]
  distance[0, 0] ← 0
  for each row i from 1 to m do
      distance[i, 0] ← distance[i – 1, 0] + del-cost(source_i)
  for each column j from 1 to n do
      distance[0, j] ← distance[0, j – 1] + ins-cost(target_j)
  for each row i from 1 to m do
    for each column j from 1 to n do
      distance[i, j] ← MIN(distance[i – 1, j] + del-cost(source_i),
                          distance[i – 1, j – 1] + subst-cost(source_i, target),
                          distance[i, j – 1] + ins-cost(target_j))
return distance[m, n]
```

3 N-grams

3.1 N-gram approximation
- $P(w_k|w_1,...,w_{k-1}) \approx P(w_k|w_{k-(n-1)},...,w_{k-2},{}^2{k-1})$
- MLE: $P(w_k|w_{k-(n-1)},...,w_{k-2},w_{k-1},w_k) = \frac{C(w_{k-(n-1)},...,w_{k-2},w_{k-1},w_k)}{C(w_{k-(n-1)},...,w_{k-2},w_{k-1})}$
- bigram: n = 2 and trigram: n = 3

3.2 Perplexity
- $PP = m(w_1,...,w_n)^{-\frac{1}{n}} = \frac{1}{\sqrt[n]{m(w_1,...,w_n)}}$
- For bigrams, $m(w_1,...,w_n) = \prod_{k=1}^{n} P(w_k|w_{k-1})$
- Weighted average number of choices a random variable has to make (weighted average branching factor of a language)

3.3 Smoothing
- Add k smoothing: $P(w|w_0) = \frac{C(w_0 w)+k}{C(w_0)+kV}$
- Discount: $\frac{C*(w_0 w)}{C(w_0 w)}$
- Witten-Bell Smoothing: $P(w|w_0) = \begin{cases} \frac{C(w_0 w)}{C(w_0)+T(w_0)} & \text{if } C(w_0 w) > 0 \\ \frac{T(w_0)}{Z(w_0)(C(w_0)+T(w_0))} & \text{if } C(w_0 w) = 0 \end{cases}$
- Kneser-Ney Smoothing for Bigram:

$$P(w|w_0) = \begin{cases} \frac{C(w_0 w)-D}{C(w_0)} & \text{if } C(w_0 w) > 0 \\ \frac{\alpha(w).|\{w':C(w' w_0)>0\}|}{\sum_w |\{w':C(w' w)>0\}|} & \text{if } C(w_0 w) = 0 \end{cases}$$

3.4 Entropy
- $H(x) = -\sum_{x \in X} p(x) log_2 p(x)$
- Entropy for a language: $H(L) = lim_{n->inf} \frac{-1}{n} log_2 p(w_1,...,w_n)$
- Cross Entropy: $H(p,m) = lim_{n->inf} \frac{-1}{n} \sum_{w \in L} p(w_1,...,w_n) log_2 m(w_1,...,w_n)$
- Relation between Perplexity and Entropy:

$$Perplexity = 2^H = 2^{\frac{-1}{n} log_2 m(w_1,...,w_n)} = m(w_1,...,w_n)^{\frac{1}{n}}$$

4 POS Tagging

4.1 Stochastic POS Tagging
- $T \hat{=} argmax_{t_1,...,t_T} \prod_{i=1} TP(t_i|t_{i-1}).P(w_i|t_i).P(</s>|t_T)$
- Time Complexity: $O(T.N^T)$

4.2 Viterbi (Dynamic Programming)
- $v_t(j) = max_{i=1} N a_{ij} b_j(o(t))$ where a_{ij} is the transition probability, b_j is the observation likelihood of observation $o(t)$
- Time Complexity: $O(N^2)$

4.3 Penn Treebank POS Tags

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	*and, but, or*	SYM	symbol	*+,%, &*
CD	cardinal number	*one, two, three*	TO	"to"	*to*
DT	determiner	*a, the*	UH	interjection	*ah, oops*
EX	existential 'there'	*there*	VB	verb, base form	*eat*
FW	foreign word	*mea culpa*	VBD	verb, past tense	*ate*
IN	preposition/sub-conj	*of, in, by*	VBG	verb, gerund	*eating*
JJ	adjective	*yellow*	VBN	verb, past participle	*eaten*
JJR	adj., comparative	*bigger*	VBP	verb, non-3sg pres	*eat*
JJS	adj., superlative	*wildest*	VBZ	verb, 3sg pres	*eats*
LS	list item marker	*1, 2, One*	WDT	wh-determiner	*which, that*
MD	modal	*can, should*	WP	wh-pronoun	*what, who*
NN	noun, sing. or mass	*llama*	WP$	possessive wh-	*whose*
NNS	noun, plural	*llamas*	WRB	wh-adverb	*how, where*
NNP	proper noun, singular	*IBM*	$	dollar sign	*$*
NNPS	proper noun, plural	*Carolinas*	#	pound sign	*#*
PDT	predeterminer	*all, both*	"	left quote	*' or "*
POS	possessive ending	*'s*	"	right quote	*' or "*
PRP	personal pronoun	*I, you, he*	(	left parenthesis	*[, (, {, <*
PRP$	possessive pronoun	*your, one's*	)	right parenthesis	*],), }, >*
RB	adverb	*quickly, never*	,	comma	*,*
RBR	adverb, comparative	*faster*	.	sentence-final punc	*. ! ?*
RBS	adverb, superlative	*fastest*	:	mid-sentence punc	*: ; ... – -*
RP	particle	*up, off*			

5 Neural Networks

5.1 Multi Layer Perceptron

$$NN_{MLP2}(x) = (g^2(g^1(xW^1 + b^1)W^2 + b^2))W^3 + b^3$$
$$h^1 = g^1(xW^1 + b^1)$$
$$h^2 = g^2(xW^2 + b^2)$$
$$y = h^2 W^3 + b^3$$

5.2 Activation Functions
- RELU(x) = $g(x) = max(0,x)$
- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$, $\sigma'(x) = \sigma(x)(1-\sigma(x))$
- Tanh: $tanh(x) = \frac{e^x+e^{-x}}{e^x-e^{-x}}$, $tanh'(x) = 1-(tanh(x))^2$
- HardTanh: $hardtanh(x) = \begin{cases} -1 & x < -1 \\ 1 & x > 1 \\ x & \text{otherwise} \end{cases}$

5.3 Loss Functions
- Cross Entropy Loss: $L_{logistic}(\hat{y}, y) = -ylog_2(\hat{y}) - (1-y)log_2(1-\hat{y})$ where $y \in \{0,1\}$ for binary classification
- For multi class classification (C > 2), $L_{logistic}(\hat{y}, y) = \sum_{t}^{C} t_i log_2(\hat{y_t})$
- Softmax loss: $L_{softmax}(\hat{y}, y) = \frac{e^{\hat{y}[i]}}{\sum_j e^{\hat{y}[j]}}$, squishes range of values to (0, 1), a probability distribution
- Squared (quadratic) loss: $L_{squared}(y/hat, y) = 0.5 * (\hat{y} - y)^2$

5.4 Gradient Descent

$$w_i = w_i - \alpha \frac{\delta L}{w_i} \frac{\delta L}{wm} = \sum_{i=1} N \frac{\delta L}{\delta x_i} \frac{\delta x_i}{\delta wm}$$

6 Word Embeddings

6.1 Word2Vec

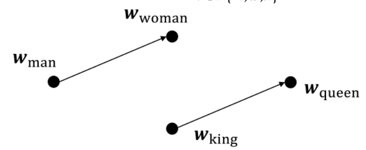
$$s(w,c) = w.c$$
$$P(+|t,c) = \frac{1}{1+e^{-t.c}}$$
$$P(-|t,c) = 1 - P(+|t,c) = \frac{e^{-t.c}}{1+e^{-t.c}}$$

6.2 Choosing noise words
- $P_\alpha(w) = \frac{count(w)^\alpha}{\sum_w' count(w')^\alpha}$ where count is the unigram frequency. In practice, $alpha$ value of 0.75 works quite well.
- Maximize objective function:
  $\sum_{(w,c) \in D} log_2 P(+|t,c) + \sum_{(w,c) \in D} log_2 P(-|t,c)$

6.3 CBOW
- $c = \sum_i c_i$
- $logP(+|w, c_{1:k}) = log \frac{1}{1+e^{(\sum_{i=1..k} w.c_i)}}$

6.4 Skip-gram
- $P(+|w, c_{1..k}) = \prod_{i=1}^{k} \frac{1}{1+e^{-w.c_i}}$
- $P(+|w, c_{1..k}) = \sum_{i=1}^{k} log \frac{1}{1+e^{-w.c_i}}$

6.5 Cosine similarity
- $sim_{cos}(u,v) = \frac{\mathbf{u}.\mathbf{v}}{|||\mathbf{u}|||_2 |||\mathbf{v}|||_2}$
- $analogy(m : w -> k : ?) = argmax_{v \in V \{m,w,k\}} cos(v, k - m + w)$



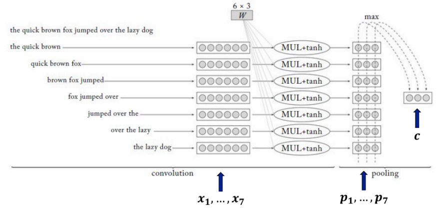
7 CNN

7.1 Why is it used?
- Identify informative ngrams
- Consider local ordering patterns

7.2 Process
- Apply non linear learned function (filter) over each k-word sliding window
- Apply l filters to get l-dimensional vector
- Combine vectors from different windows using pooling into single l-dimensional vector
- Feed single l-dimensional vector into neural network for prediction

7.3 Convolution
- Narrow convolution (no padding): m = n - (k - 1)
- Wide convolution (padding k - 1 to each side) : m = n + (k - 1)

7.4 Pooling
- Max pooling: $c_j = max_{1<=i<=m} p_{i_{[j]}} \forall j \in [1, l]$
- Average pooling: $\frac{1}{m} \sum_{i=1}^{m} p_i$



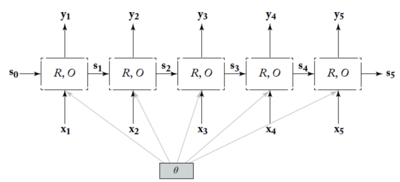
Narrow convolution, $n = 9, d = 2, k = 3, l = 3, m = 7$

8 RNN

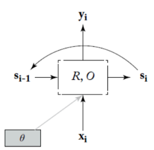
8.1 Why is it used?
- Capture subtle patterns and regularities in sequences
- Model non-Markovian dependencies

8.2 RNN Abstraction

$$RNN^*(x_{1:n} : s_0) = y_{1..n}$$
$$s_i = R(s_{i-1}, x_i)$$
$$y_i = o(s_i)$$



Recursive view:



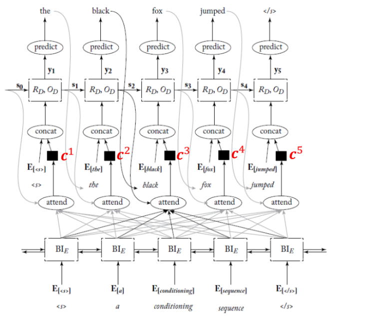
8.3 Elman RNN
Sensitive to the order of the words

$$y_i = O_R NN(s_i)$$
$$x_i \in R^{d_x}, S_i \in R^{d_s}, W \in R^{(d_x+d_s).d_s}, b \in R_s^d$$

8.4 CBOW RNN
Does not take into account order of words

$$s_i = R_{CBOW}(s_{i-1}, x_i) = s_{i-1} + x_i$$
$$y_i = O_{CBOW}(s_i)$$

9 Seq2Seq



$$s_i = R_{dec}(s_{j-1}, [\hat{t_j}; c^j])$$
$$y_j = O_{dev}(s_j)$$
$$P(t_{j+1}|\hat{t_{1:j}}, x_{1:n}) = softmax(MLP^{out}(y_j))$$
$$c_{1:n} = biRNN^*_{enc}(x_{1:n})$$
$$\alpha^j_{[i]} = v.tanh([s_{j-1}; c_i]U + b)$$
$$\alpha^j_{[i]} = softmax(\alpha^j_{[1]}, ..., \alpha^j_{[n]})$$
$$c^j = \sigma_{i=1}^{n} \alpha_{[}i]^j.c_i(\text{attend})$$

10 Grammars

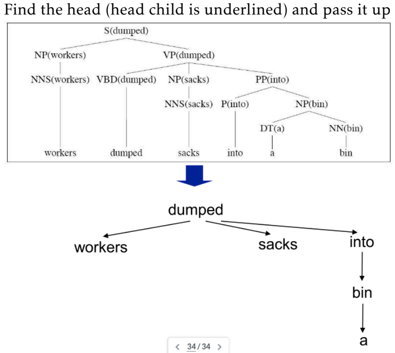
10.1 CFG
$G = (N, \Sigma, P, S)$
- N - terminal symbols
- Σ - non-terminal symbols
- P - a -> A where $a \in N, A \in (\Sigma \cup N)^*$

10.2 CNF
- A -> BC or A -> a
- No *epsilon*

10.3 Equivalence
- Strong equivalence: $L(G1) = L(G2)$ and same phrase structure
- Weak equivalence: $L(G1) = L(G2)$ but different phrase structures

10.4 Convert to CNF
- Copy all conforming rules to the new grammar unchanged
- Convert terminals within rules to dummy non-terminals
- Convert unit-productions
- Binarize all rules and add to new grammar

10.5 Converting Phrase Structure to Untyped Dependency
- Find the head (head child is underlined) and pass it up the tree



10.6 Parsing
- Top down parsing: Goal directed, begin from start symbol and then derive parse tree for the given sentence consisting of terminals

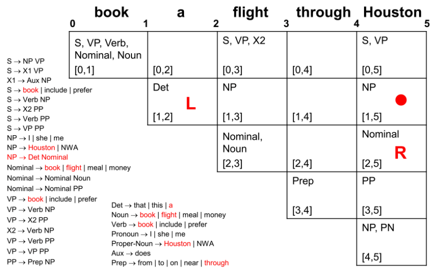
- Bottom up parsing: Data directed, start form the terminals in a sentence and derive parse tree by tracing up to the start symbol.

10.7 Ambiguity

- Attachment Ambiguity: Ambiguity on how a verb phrase is split up based on the rules in the CFG (VP -> V NP PP or VP -> V PP for example)
- Coordination Ambiguity: Ambiguity depending on how the Noun Phrase is split
- Noun Phrase Bracketing Ambiguity: Ambiguity depends on how the Noun Phrase is split

10.8 Parsing Algorithms

10.8.1 CKY algorithm



- Bottom-up dynamic programming
- Grammar must be in CNF
- Time Complexity: $O(n^3)$, Space Complexity: $O(n^2)$
- Probabilistic version: most probable parse stored based on taking the maximum of probabilities calculated using the assigned probabilities for each production in the Grammar for regular CKY

10.8.2 Earley Algorithm

- Top-down dynamic programming
- Grammar need not be in CNF
- Scan sentence from left to right
- Predictor: Create new states from original states using grammar (e.g. Given S -> .VP [0, 0], add VP -> .Verb [0, 0] and VP ->.Verb NP [0, 0])
- Scanner: Use predicted POS to incorporate the next word (e.g. Given the state VP -> .Verb NP [0, 0], add Verb -> book. [0, 1])
- Completer: Find and advanced previously created states looking for non-terminal at this position (e.g. Given stated NP -> Det Nominal.[1, 3] and VP -> Verb.NP [0, 1] and add the new state VP -> Verb NP. [0, 3])
- Time Complexity: $O(n^3)$ maximum but tends to perform better, Space Complexity: $O(n)$

11 Statistical Parsing

11.1 PCFG

- $P(T,S) = \prod_{t \in T}$ p (r(n)) where r(n) represents the rules of CFG applicable to the parse trees
- $\hat{T(S)} = argmaxP(T)$

11.2 Parse Tree Scoring

- $s_tree(T) = \sigma_{(i,j,l) \in T} s(i,j,l)$
- Feed sentence through bidirectional LSTM and then a Dense NN and score based on loss
- Dynamic Programming:

$$s_{best}(i,j) = max_l[s(i,j,l)] if j - i = 1$$

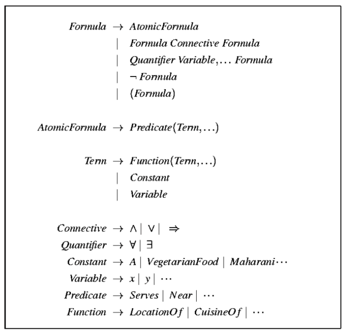
$$s_{best}(i,j) = max_l[s(i,j,1)] + max_k[s_{best}(i,k) + s_{best}(k,j)]$$

- Best scored tree: $T \hat{=} argmax_T[s_{tree}(T)]$

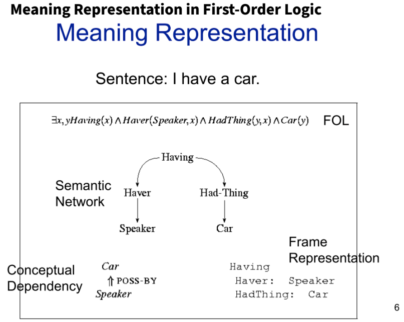
12 Semantics

12.1 First Order Logic

FOL



12.2 Meaning Representation in First-Order Logic



12.3 Syntax Driven Semantic Analysis