Natural Language Understanding and Generation

Introduction to Natural Language Processing

Challenges

Syntax & Semantics
- L = Words + rules + exceptions..
- Ambiguity at all levels..

Variety
- We speak different languages..
- And language is a cultural entity..
- So they are not equivalent..
- Highly systematic but also complex..

- Keeps changing.. New words, New rules and New exceptions..

Source :
Electronic texts / Printed texts / Acoustic Speech Signal..
- they are noisy..
- Language looks obvious to us.. But it is a Big Deal

Types of Ambiguities:

Structural Ambiguities
- Namrata thinks she understands me.
- She thinks Namrata understands me.
- Visiting relatives can be nuisance. (tw meanings)

Grammatical Ambiguities
- I (feminine or masculine) go.
- Can- Noun = container, Can – Modal(auxiliary verb), Can-verb = to can means to pack

Lexical Ambiguities:
- Polysemy Ex: "understand" (I get it)
- Ex:going to the bank/savings account in a bank (BUILDIN <-> ORGANIZATION)
- Homonymy Ex: Bank= river, financial bank

Lexical analysis is dividing the whole chunk of text into paragraphs, sentences, and words. Concerns with listing of words and categorizing them
word and word meanings - the lexicon

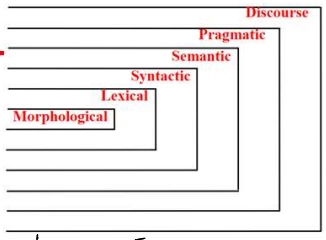
Lexicon of a language means the collection of words and phrases in a language. In linguistics, a lexicon is a language's inventory of lexemes

friendful or beautyship → Lexically Incorrect
friendship and beautiful is correct → Lexically Correct

Semantic knowledge is the aspect of language knowledge that involves word meanings/vocabulary.
Concerns what words mean and how these meanings – combine in sentences to form **sentence meanings**.
Semantics concerns the **meaning of linguistic expressions** independently of the context in which they occur.

"Green ideas have large noses" → Syntactically Correct But Semantically Incorrect

Different Levels of Language Analysis



Morphological Analysis
Lexical Analysis
Syntactic Analysis
Semantics Analysis
Pragmatic Analysis
Discourse Knowledge

FireHouse = Fire + House each mophene has unique meaning ∴ Free morphenes

Morphology concerns how words are constructed from basic meaning units called morphemes. A morpheme is the primitive unit of meaning in a language

cats → morphological parser → cat + s → No meaning ⇒ bound; → some meaning ⇒ morphenes

Two broad classes of morphemes
stems—the central morpheme of the word, supplying the main meaning
affixes—adding "additional" meanings of various kinds

Syntax analysis checks the text for meaningfulness comparing to the rules of formal language. Concerns how words can be put together to form correct sentences and determines what **structural role** each word plays in the sentence and what phrases are subpart of other phrases

"Large have green ideas nose" → Lexically Correct But Syntactically Incorrect

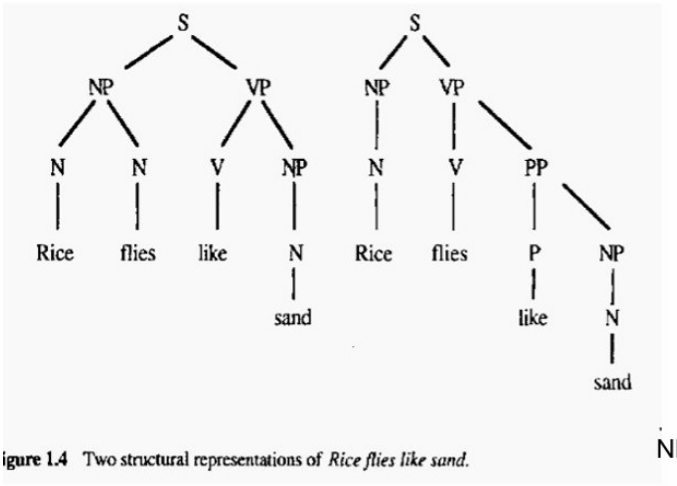
Pragmatics is meaning in use / context. Concerns how sentences are used in **different situations** how use affects the **interpretation** of sentence and whether it produces a useful meaning.

"She cuts banana with a pen" → Semantically Correct But Pragmatically Incorrect

Discourse analysis concerns how the immediately preceding sentences affect the interpretation of next sentence

"NLP is a very interesting subject. IT is liked by students. They generally do well in exams." → Pragmatically Correct But Incorrect at a Discourse Level

- **Morphological** analysis: This is the analysis of the structure and formation of words. For example, the word 'unhappy' consists of two morphemes: the prefix 'un-' and the root 'happy'. The prefix 'un-' changes the meaning of the root 'happy' to its opposite. A small example of morphological analysis is to identify the morphemes and their functions in a word like 'reopened'. A counterexample to the statement "Every word consists of one or more morphemes" is the word 'a', which is not a morpheme by itself, but a phonetic reduction of the morpheme 'an'.
- **Lexical** analysis: This is the analysis of the meaning and usage of words and phrases. For example, the word 'bank' can have different meanings depending on the context: it can mean a financial institution, a slope of land, or a set of similar things. A small example of lexical analysis is to explain the difference between the words 'affect' and 'effect'. A counterexample to the statement "Every word has a fixed meaning" is the word 'bat', which can mean a flying mammal, a wooden club, or a verb meaning to strike something.
- **Syntactic** analysis: This is the analysis of the structure and rules of sentences and clauses. For example, a sentence consists of a subject and a predicate, and a clause consists of a subject, a verb, and an object or complement. A small example of syntactic analysis is to identify the parts of speech and the functions of the words in a sentence like 'She gave him a book'. A counterexample to the statement "Every sentence has a subject and a predicate" is the sentence 'Go!', which has only an implied subject and a predicate.
- **Semantic** analysis: This is the analysis of the meaning and interpretation of sentences and texts. For example, a sentence can have different meanings depending on the context, the tone, or the intention of the speaker or writer. A small example of semantic analysis is to explain the difference between the sentences 'He is cold' and 'He is a cold person'. A counterexample to the statement "Every sentence has a literal meaning" is the sentence 'It's raining cats and dogs', which has a figurative meaning that implies heavy rain, not animals falling from the sky.
- **Pragmatic** analysis: This is the analysis of the use and effect of language in specific situations and purposes. For example, language can be used to persuade, inform, entertain, or request. A small example of pragmatic analysis is to explain the difference between the utterances 'Can you pass me the salt?' and 'Pass me the salt'. A counterexample to the statement "Every utterance has a direct meaning" is the utterance 'Can you open the window?', which has an indirect meaning that requests the hearer to open the window, not a question about their ability.
- **Discourse** analysis: This is the analysis of the structure and coherence of larger units of language, such as paragraphs, conversations, or texts. For example, discourse analysis can examine how different speakers or writers organize their ideas, how they relate to each other, and how they achieve their goals. A small example of discourse analysis is to identify the topic, the main points, and the transitions in a paragraph. A counterexample to the statement "Every paragraph has a topic sentence" is the paragraph 'I like apples. They are sweet and crunchy. You can make pies and jams with them. They are good for your health.', which has no clear topic sentence, but several supporting sentences.



igure 1.4 Two structural representations of *Rice flies like sand.*

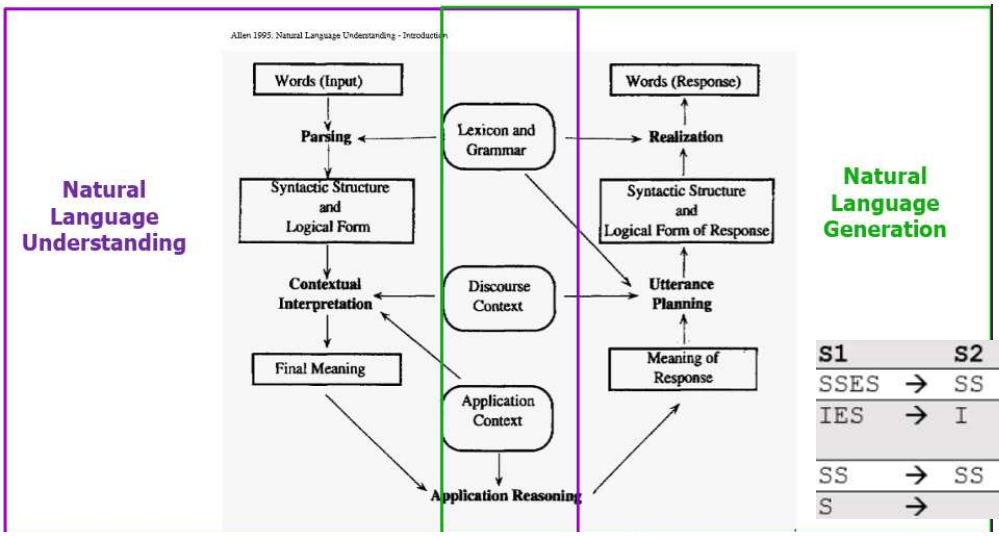
CFG Rules
S -> NP VP
NP -> N N
NP -> N
VP -> V NP
VP -> V PP
PP -> P NP

Lexicon
Rice : N
Flies : N, V
Like : V, P
Sand : N

NP – Noun Phrases

VP – Verb Phrases
PP – Prepositional Phrases

1.4 Two structural representations of "*Rice flies like sand*".



S1		S2	word		stem
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
			ties	→	ti
SS	→	SS	caress	→	caress
S	→		cats	→	cat

cats → cat Seusss → Seuss

Stemming is a crude heuristic process that chops off the ends of words, often includes the removal of derivational affixes

"Seusss", "cat", "hat", "different", "other", "cats", "Seuss", "perfectionist", "always", "organized", "Being", "firm", "believer", "democracy", "seuss", "ardent", "fan", "democratic" party", "nation"

break, breaks, broke, broken and breaking are forms of the same **lexeme**, with break as the **lemma** by which they are **indexed**.

Lemma is the canonical form, dictionary form, or citation form of a set of word forms.

Lemma: same stem, part of speech, rough word sense
cat and cats = same lemma cat
am, are, and is : lemma be

Lemmatization is the task of determining that two words have the same root despite their surface differences.

Wordform: the full inflected surface form
cat and cats = different wordforms

The word "**walk**" is the base form for the word "**walking**", and hence this is matched in both stemming and lemmatization.
"**meeting**" : base form of a noun
: or a form of a verb ("**to meet**")
e.g., "in our last meeting" or "We are meeting again tomorrow"

"Seuss", "cat", "hat", "different", "other", "cat", "Seuss", "perfectionist", "always", "organized", "Being", "firm", "believer", "democracy", "seuss", "ardent", "fan", "democratic" party", "nation"

N-Gram Language Modelling

Language model computes:
Probability of the sentence
P(W) or $P(w_1,w_2 \ldots w_{n-1}w_n)$

P("its water is so transparent")
= P(its) × P(water | its) × P(is | its water)
× P(so | its water is)
× P(transparent | its water is so)

Simplifying assumption:
limit history of fixed number of words, n

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

or

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

$$P(w1\ w2 \ldots \ldots wn) = \prod_{k=1}^{n} P(w_k|w_1^{k-1})$$

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1})$$

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

In other words, we approximate each component in the product

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-k} \ldots w_{i-1})$$

$P(\text{I} \mid \text{<s>}) = \frac{2}{3} = .67$ $P(\text{Sam} \mid \text{<s>}) = \frac{1}{3} = .33$ $P(\text{am} \mid \text{I}) = \frac{2}{3} = .67$
$P(\text{</s>} \mid \text{Sam}) = \frac{1}{2} = 0.5$ $P(\text{Sam} \mid \text{am}) = \frac{1}{2} = .5$ $P(\text{do} \mid \text{I}) = \frac{1}{3} = .33$

N=1 → Unigram language model $P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$; $P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$

P("its water is so transparent") = P(its) × P(water) × P(is) × P(so) × P(transparent)

N=2 → Bigram language model $P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$

P("I want to eat healthy food") =
P(I | <start>) * P(want | I) * P(to | want) * P(eat | to) * P(healthy | eat) * P(food | healthy) * P(<end> | food)

Perplexity is a measure of how well a language model predicts a sequence of words. It is defined as the exponentiated average negative log-likelihood of a sequence, calculated with exponent base e [1]. In other words, perplexity is the inverse probability of the sequence, normalized by its length. A lower perplexity means a higher probability, and a higher perplexity means a lower probability.

Perplexity tells us how surprised or uncertain a language model is about the next word in the sequence. A good language model should assign high probabilities to well-formed and coherent sentences, and low probabilities to ill-formed and incoherent sentences. Therefore, a good language model should have a low perplexity on the test data. Perplexity is one way to evaluate the quality and performance of language models in NLP [2].

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$

unigram $$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i)}}$$

bigram $$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

$$\log(p_1 \times p_2 \times p_3 \times p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4$$

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

$$P_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{\sum_w (C(w_{n-1}w) + 1)} = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

$$P^*_{\text{Add-k}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + kV}$$

If we are trying to compute $P(w_n|w_{n-2}w_{n-1})$ but we have no examples of a particular trigram $w_{n-2}w_{n-1}w_n$, we can instead estimate its probability by using the bigram probability $P(w_n|w_{n-1})$. Similarly, if we don't have counts to compute $P(w_n|w_{n-1})$, we can look to the unigram $P(w_n)$.

In other words, sometimes using **less context** is a good thing, helping to generalize more for contexts that the model hasn't learned much about. There are two ways to use this n-gram "hierarchy". In **backoff**, we use the trigram if the evidence is sufficient, otherwise we use the bigram, otherwise the unigram. In other words, we only "back off" to a lower-order n-gram if we have zero evidence for a higher-order n-gram. By contrast, in **interpolation**, we always mix the probability estimates from all the n-gram estimators, weighting and combining the trigram, bigram, and unigram counts.

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n|w_{n-2}w_{n-1})$$

$$\sum_i \lambda_i = 1$$

Question 2. N-gram language models [5 Marks]

"Erring is human. To err is human. Seeing is believing. To see is to believe."

Given above training data, use bigram language model to compare the likelihood of below test sentences. **Note:** To handle zero probability, use interpolation with weights 0.3 & 0.7 for unigram and bigram respectively. Don't account for punctuation ie., "." for the modelling and predictions.

a) Test Sentence 1: Seeing is human
b) Test Sentence 2: To believe is human

Cleaned text: "<S> erring is human to err is human seeing is believing to see is to believe <E>"

unigrams

$P(\text{erring}) = \frac{1}{15}$

$P(\text{is}) = \frac{4}{15}$

$P(\text{human}) = \frac{2}{15}$

$P(\text{to}) = \frac{3}{15} = \frac{1}{5}$

$P(\text{err}) = \frac{1}{15}$

$P(\text{believing}) = \frac{1}{15}$

$P(\text{believe}) = \frac{1}{15}$

$P(\text{see}) = \frac{1}{15}$

$P(\text{seeing}) = \frac{1}{15}$ & $P(\text{<S>}) = P(\text{<E>}) = \frac{1}{15}$

bigrams

$P(\text{erring} \mid \text{<S>}) = \frac{1}{1}$

$P(\text{is} \mid \text{erring}) = \frac{1}{1}$

$P(\text{human} \mid \text{is}) = \frac{2}{4} = \frac{1}{2}$

$P(\text{to} \mid \text{human}) = \frac{1}{2}$

$P(\text{err} \mid \text{to}) = \frac{1}{3}$

$P(\text{is} \mid \text{err}) = \frac{1}{1}$

$P(\text{seeing} \mid \text{human}) = \frac{1}{2}$

$P(\text{is} \mid \text{seeing}) = \frac{1}{1}$

$P(\text{believing} \mid \text{is}) = \frac{1}{4}$

$P(\text{to} \mid \text{believing}) = \frac{1}{1}$

$P(\text{see} \mid \text{to}) = \frac{1}{2}$

$P(\text{is} \mid \text{see}) = \frac{1}{1}$

$P(\text{to} \mid \text{is}) = \frac{1}{4}$

$P(\text{believe} \mid \text{to}) = \frac{1}{3}$

$P(\text{<E>} \mid \text{believe}) = \frac{1}{1}$

Test sentence 1: seeing is human

$$P(\text{"seeing is human"}) = \hat{P}(\text{seeing} \mid \text{<S>}) \times \hat{P}(\text{is} \mid \text{seeing}) \times \hat{P}(\text{human} \mid \text{is}) \times \hat{P}(\text{<E>} \mid \text{human})$$

$$(0.3 \times P(\text{seeing}) + 0.7 \times P(\text{seeing} \mid \text{<S>})) \times (0.3 \times P(\text{is}) + 0.7 \times P(\text{is} \mid \text{seeing})) \times (0.3 \times P(\text{human}) + 0.7 \times P(\text{human} \mid \text{is})) \times (0.3 \times P(\text{<E>}) + 0.7 \times P(\text{<E>} \mid \text{human}))$$

$$= (0.3 \times 1/15 + 0.7 \times 0) \times (0.3 \times 4/15 + 0.7 \times 1) \times (0.3 \times 2/15 + 0.7 \times 1/2) \times (0.3 \times 1/15 + 0.7 \times 0) = 0.00012168 \text{ or } 1.2168e^{-4}$$

```
In [11]: (0.3/15)*(0.3*4/15 + 0.7)*(0.3*2/15 + 0.7/2)*(0.3/15)
Out[11]: 0.00012167999999999999
```

Question 4. Vector semantics [1.5+1.5=3 Marks]

Answer the following: with respect to Training Corpus D:
Training Corpus D
document - d1: Today is sunny day! I like sunny days
document - d2: I do not like sunny days. I like chocolate
document - d3: I like both sunny and rainy day

Use the TF-IDF to decide on :

The word “sunny” is discriminative in identifying documents in corpus D.
The word “chocolate” is discriminative in identifying documents in corpus D. Justify your answer.

TF is calculated by dividing the number of times a word appears in the document by the total number of words in that document.

TF	D1	D2	D3
Sunny	2/8=0.25	1/9=0.1111	1/7=0.1429
Chocolate	0	1/9=0.1111	0

IDF is calculated by taking the logarithm of the ratio of the total number of documents (3) to the number of documents that contain word

	IDF
sunny	$\log_{10}(3/3) = 0$
chocolate	$\log_{10}(3/1) = 0.47712125471966244$

TF-IDF is calculated by multiplying the TF and IDF values

TF-IDF	D1	D2	D3
Sunny	0	0	0
Chocolate	0	$(1/9)*\log_{10}(3/1)=0.0530134727466291$	0

a) The word "sunny" is discriminative in identifying documents in corpus D. (No)
b) The word "chocolate" Is discriminative In identifying documents in corpus D. (Yes)

Since TF-IDF value of "chocolate" in D2 is more than that of "sunny"

```
d1 = "today is sunny day I like sunny days"
d2 = "I do not like sunny days I like chocolate"
d3 = "I like both sunny and rainy day"
```

```
corpus = [d1,d2,d3]
```

```
import pandas as pd
import numpy as np

words_set = set()

for doc in  corpus:
    words = doc.split(' ')
    words_set = words_set.union(set(words))

print('Number of words in the corpus:',len(words_set))
print('The words in the corpus: \n', words_set)
```

```
Number of words in the corpus: 13
The words in the corpus:
 {'today', 'day', 'both', 'do', 'chocolate', 'not', 'rainy', 'days', 'and', 'is', 'like', 'sunny', 'I'}
```

```
n_docs = len(corpus)         #·Number of documents in the corpus
n_words_set = len(words_set) #·Number of unique words in the

df_tf = pd.DataFrame(np.zeros((n_docs, n_words_set)), columns=list(words_set))

# Compute Term Frequency (TF)
for i in range(n_docs):
    words = corpus[i].split(' ') # Words in the document
    for w in words:
        df_tf[w][i] = df_tf[w][i] + (1 / len(words))

df_tf
```

	today	day	both	do	chocolate	not	rainy	days	and	is	like	sunny	I
0	0.125	0.125000	0.000000	0.000000	0.000000	0.000000	0.000000	0.125000	0.000000	0.125	0.125000	0.250000	0.125000
1	0.000	0.000000	0.000000	0.111111	0.111111	0.111111	0.000000	0.111111	0.000000	0.000	0.222222	0.111111	0.222222
2	0.000	0.142857	0.142857	0.000000	0.000000	0.000000	0.142857	0.000000	0.142857	0.000	0.142857	0.142857	0.142857

```
print("IDF of: ")

idf = {}

for w in words_set:
    # no.of documents in the corpus that contain this word
    k = 0

    for i in range(n_docs):
        if w in corpus[i].split():
            k += 1

    idf[w] =  np.log10(n_docs / k)
    print(w,'is log({}/{})'.format(n_docs,k),f'= {idf[w]}' )
```

```
IDF of:
today is log(3/1) = 0.47712125471966244
day is log(3/2) = 0.17609125905568124
both is log(3/1) = 0.47712125471966244
do is log(3/1) = 0.47712125471966244
chocolate is log(3/1) = 0.47712125471966244
not is log(3/1) = 0.47712125471966244
rainy is log(3/1) = 0.47712125471966244
days is log(3/2) = 0.17609125905568124
and is log(3/1) = 0.47712125471966244
is is log(3/1) = 0.47712125471966244
like is log(3/3) = 0.0
sunny is log(3/3) = 0.0
I is log(3/3) = 0.0
```

```
df_tf_idf = df_tf.copy()

for w in words_set:
    for i in range(n_docs):
        df_tf_idf[w][i] = df_tf[w][i] * idf[w]

df_tf_idf
```

	today	day	both	do	chocolate	not	rainy	days	and	is	like	sunny	I
0	0.05964	0.022011	0.00000	0.000000	0.000000	0.000000	0.00000	0.022011	0.00000	0.05964	0.0	0.0	0.0
1	0.00000	0.000000	0.00000	0.053013	0.053013	0.053013	0.00000	0.019566	0.00000	0.00000	0.0	0.0	0.0
2	0.00000	0.025156	0.06816	0.000000	0.000000	0.000000	0.06816	0.000000	0.06816	0.00000	0.0	0.0	0.0

Skip Gram

Given a training corpus: "I live in Bangalore, India", use the skip-gram negative sampling method and answer the following: The initial embedding matrix has dimensions |V| x 3 and is given as follows:

a)Generate the training dataset for an input word 'I' and context window of 1 next word and hyper parameter value k=3 for the negative sampling task. Use the information available in the question.

b)Calculate the error for the above dataset during the first iteration of skip-gram training.

I	0.03	-0.504	0.28
Live	0.1	-0.009	-0.8
In	-0.56	0.745	0.002
Bangalore	-0.592	-0.81	23
India	-0.728	-0.06	0.499

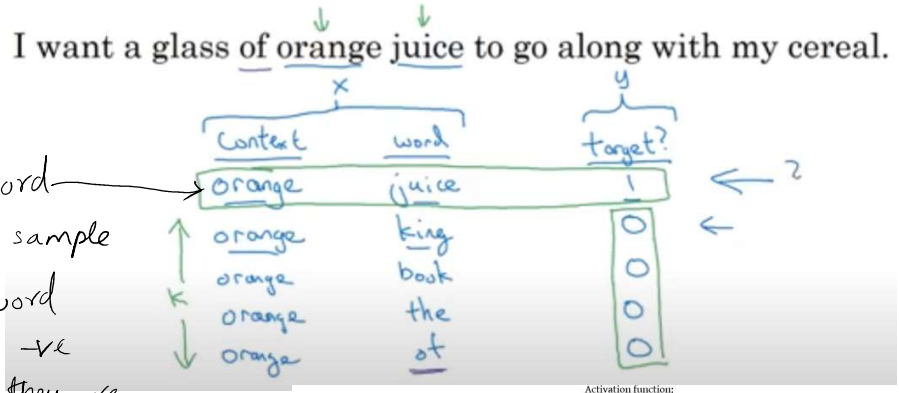
The initial context matrix has |V| x 3 dimension and given as follows:

I	-0.572	-0.588	-0.501
Live	0.22	0.83	-0.72
In	-0.84	0.501	0.246
Bangalore	-0.422	0.911	0.55
India	-0.087	-0.45	-0.07

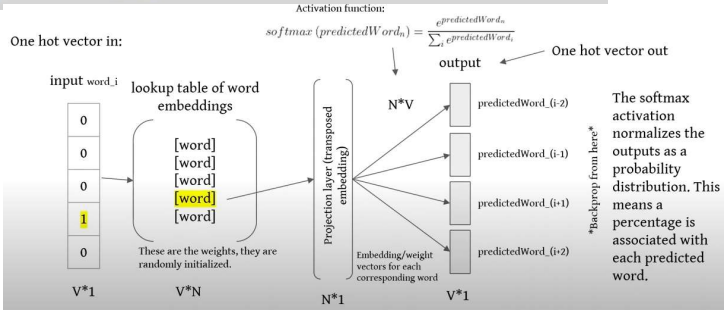
Negative Sampling for a target word

1.) pick 1 context word and mark it as +ve sample

2.) pick 'K' context word and mark them as -ve sample even though they are in context window



a.) for word "I" given K=3 imediate word is live



context	target
live	1
in	0
banglore	0
india	0

picking remaining words K-3 as -ve

b.) since we got context words as "live", "in", "banglore", "India" we will pick context vectors of these context words and pick embedding vector of "I" as those are the weights that we want to learn.

"I"

$$\begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix} \times \begin{bmatrix}0.03 & -0.504 & 0.28\\0.1 & -0.009 & -0.8\\-0.56 & 0.745 & 0.002\\-0.592 & -0.81 & 23\\-0.728 & -0.06 & 0.499\end{bmatrix} \rightarrow [0.03 \quad -0.504 \quad 0.28]_{1\times3} \times \begin{bmatrix}0.22 & 0.83 & -0.72\\-0.84 & 0.501 & 0.246\\-0.422 & 0.911 & 0.55\\-0.087 & -0.45 & -0.07\end{bmatrix}_{4\times3}$$

Transpose

$$[0.03 \quad -0.504 \quad 0.28]_{1\times3} \times \begin{bmatrix}0.22 & -0.84 & -0.422 & -0.087\\0.83 & 0.501 & 0.911 & 0.55\\-0.72 & 0.246 & 0.55 & -0.07\end{bmatrix}_{3\times4}$$

```
(0.03*0.22)+(-0.504*0.83)+(0.28*-0.72)
-0.61332

(0.03*-0.84)+(-0.504*0.501)+(0.28*0.246)
-0.208824

(0.03*-0.422)+(-0.504*0.911)+(0.28*0.55)
-0.317804

(0.03*-0.087)+(-0.504*-0.45)+(0.28*-0.07)
0.20459
```

$= [-0.61332, -0.208824, -0.3178, 0.20459]$

$0.03 \times 0.22 + -0.504 \times 0.83 + 0.28 \times -0.72$

$0.03 \times -0.84 + -0.504 \times 0.501 + 0.28 \times 0.246$

$0.03 \times -0.422 + -0.504 \times 0.911 + 0.28 \times -0.55$

$0.03 \times -0.087 + -0.504 \times -0.45 + 0.28 \times 0.07$

$\sigma(x) = \frac{1}{1+e^{-x}}$ apply $\sigma(x)$ over $= [0.351302, 0.447983, 0.42121, 0.555097] = V$

$V - k = \text{error}$

POS tagging [4 Marks]

By manual tagging I know

Tokenize and tag the following sentences using the Brown Corpus table given,

Sentence 1: "he will chair the session"

Sentence 2: "It is a chair"

Sentence 3: "He will race the car."

Sentence 4: "It is a race."

Write the best sequence of tags for sentence 1, sentence 2, sentence 3 and sentence.

S "he will chair the session" E (PRP MD VB DT NN)

S "It is a chair" E (PRP VBZ DT NN)

S "He will race the car." E (PRP MD VB DT NN DOT)

S "It is a race." E (PRP VBZ DT NN DOT)

Sr.No	Examples	Tag	Description
1	He, she, It	PRP	Personal Pronoun
2	will	MD	modal
3	Chair, race	VB	verb
4	is	VBZ	Verb, 3rd person singular
5	a, the	DT	determiner
6	chair, session, race, car	NN	Noun, singular or mass
7	.	DOT	DOT

Emission counts

	PRP	MD	VB	VBZ	DT	NN	DOT
he	2						
she							
it	2						
will		2					
chair			1			1	
race			1			1	
is				2			
a					2		
the					2		
session						1	
car						1	
.							2
Total	4	2	2	2	4	4	2

Emission probabilities

	PRP	MD	VB	VBZ	DT	NN	DOT
he	2/4 = 0.5						
she							
it	2/4 = 0.5						
will		2/2 = 1					
chair			1/2=0.5			1/4=0.25	
race			1/2=0.5			1/4=0.25	
is				2/2=1			
a					2/4=0.5		
the					2/4=0.5		
session						1/4=0.25	
car						1/4=0.25	
.							2/2=1

Transition counts

	PRP	MD	VB	VBZ	DT	NN	DOT	<END>	Total
<STRT>	4								4
PRP		2		2					4
MD			2						2
VB					2				2
VBZ					2				2
DT						4			4
NN							2	2	4
DOT								2	2

Transition probabilities

	PRP	MD	VB	VBZ	DT	NN	DOT	<END>
<STRT>	4/4=1							
PRP		2/4=0.5		2/4=0.5				
MD			2/2=1					
VB					2/2=1			
VBZ					2/2=1			
DT						4/4=1		
NN							2/4=0.5	2/4=0.5
DOT								2/2=1

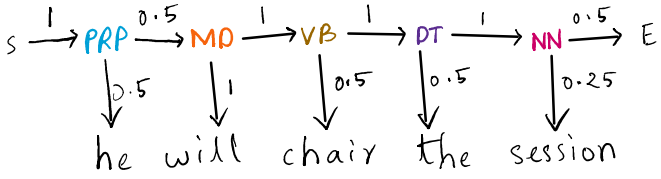
from the given tag table the words "chair" & "race" have multiple tags and using training sentences we've built Emission & Transition probabilities which inturn can be used in HMM tagger to most find most probable tags

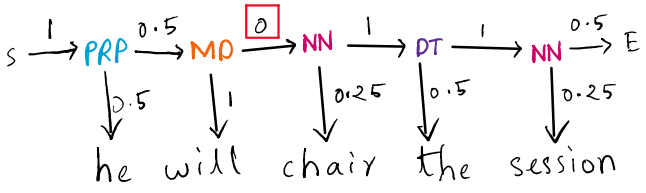
Sentence 1:

PRP MD VB DT NN
he will chair the session

or

PRP MD NN DT NN
he will chair the session





$1 \times 0.5 \times 0.5 \times 1 \times 1 \times 0.5 \times 1 \times 0.5 \times 1 \times 0.25 \times 0.5$

$= 0.0078125$

$= 0$

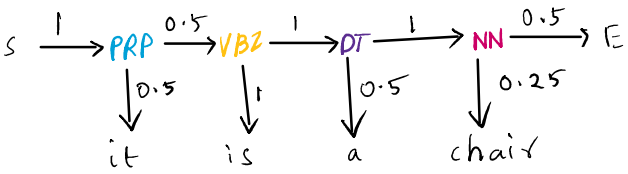
since $p(\text{PRP MD VB DT NN}) > p(\text{PRP MD NN DT NN})$ most probable tags are PRP MD VB DT NN
he will chair the session

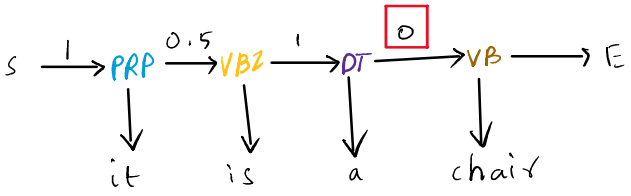
sentence 2:

PRP VBZ DT NN
it is a chair

or

PRP VBZ DT VB
it is a chair





$= 1 \times 0.5 \times 0.5 \times 1 \times 1 \times 0.5 \times 1 \times 0.25 \times 0.5$

$= 0.015625$

$= 0$

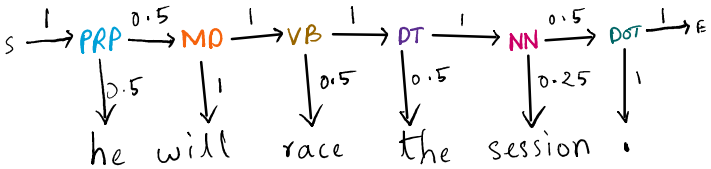
since $P(PRP, VBZ, DT, NN) > P(PRP, VBZ, DT, VB)$ most probable tags are PRP VBZ DT NN
it is a chair

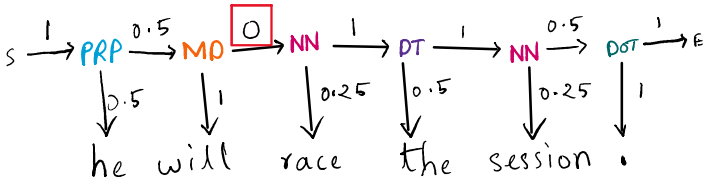
sentence 3:

PRP MD VB DT NN DOT
he will race the session .

or

PRP MD NN DT NN DOT
he will race the session .





$1 \times 0.5 \times 0.5 \times 1 \times 1 \times 0.5 \times 1 \times 0.5 \times 1 \times 0.25 \times 0.5 \times 1$

$= 0.0078125$

$= 0$

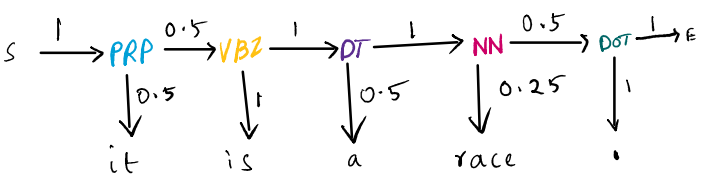
since $P(PRP\ MD\ VB\ DT\ NN) > P(PRP\ MD\ NN\ DT\ NN)$ most probable tags are PRP MD VB DT NN
he will race the session

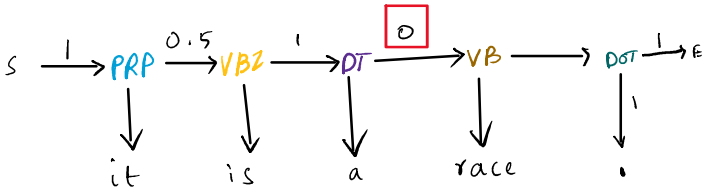
sentenu:

PRP VBZ DT NN DOT
it is a race .

or

PRP VBZ DT VB DOT
it is a race .





$= 1 \times 0.5 \times 0.5 \times 1 \times 1 \times 0.5 \times 1 \times 0.25 \times 0.5$

$= 0.015625$

$= 0$

since $P(PRP, VBZ, DT, NN, DOT) > P(PRP, VBZ, DT, VB, DOT)$ most probable tags are PRP VBZ DT NN DOT
it is a race .

function VITERBI(*observations* of len T,*state-graph* of len N) **returns** *best-path*, *path-prob*

create a path probability matrix *viterbi[N,T]*
for each state s **from** 1 **to** N **do** ; initialization step
  $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
  $backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$; termination step
$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$; termination step
bestpath $\leftarrow$ the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*

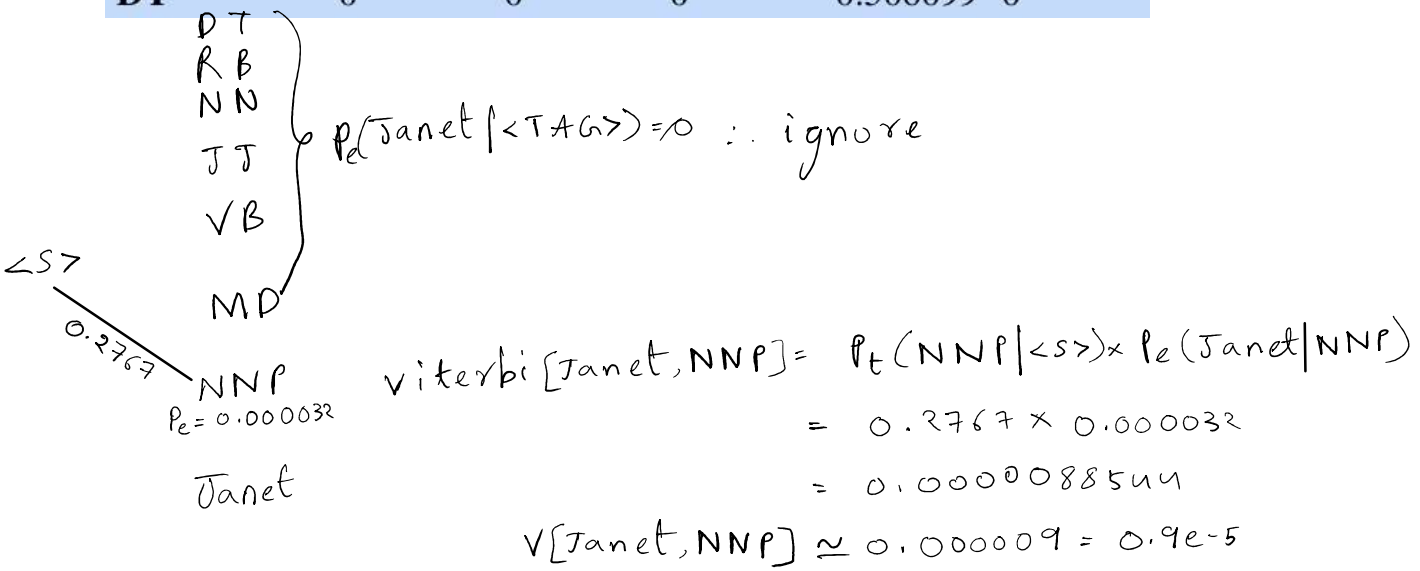
transition probabilities $P(t_i|t_{i-1})$

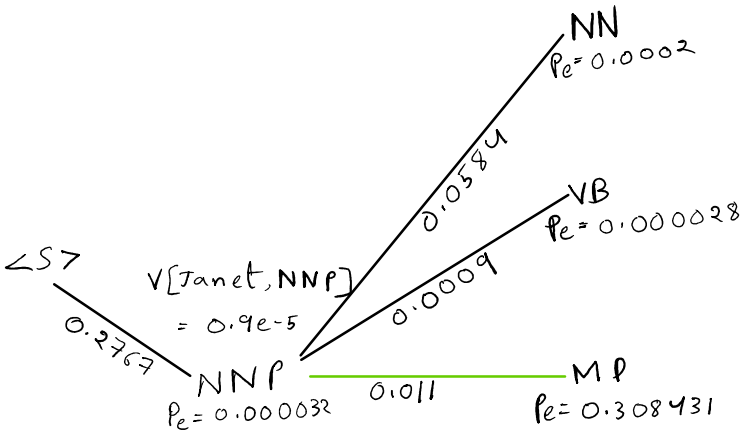
	NNP	MD	VB	JJ	NN	RB	DT
<s>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

Observation likelihoods B

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

row sum should be 1 but here its not thus it is only a subset





$V[will, NN] = V[Janet, NNP] \times P_t(NN|will) \times P_e(will|NN)$

$= 0.9e-5 \times 0.0584 \times 0.2e-3$

$= 1.0512e-10$

$V[will, VB] = V[Janet, NNP] \times P_t(VB|NNP) \times P_e(will|VB)$

$= 0.9e-5 \times 0.9e-3 \times 0.28e-4$

$= 2.268e-13$

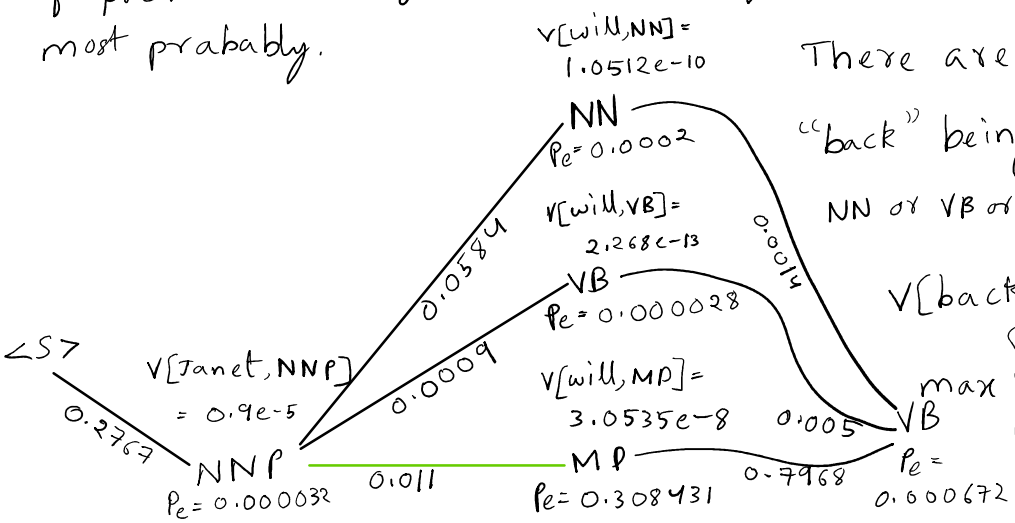
$V[will, MD] = V[Janet, NNP] \times P_t(MD|NNP) \times P_e(will|MD)$

$= 0.9e-5 \times 0.011 \times 0.308431$

$= 3.0534669e-8$

$\simeq 3.0535e-8$

since out of {V[will,NN], V[will,VB], V[will,MD]} V[will,MD] is highest. this indicates in given sentence word "will" belongs to MD tag if previous word ("janet") was NNP tag most prabably.



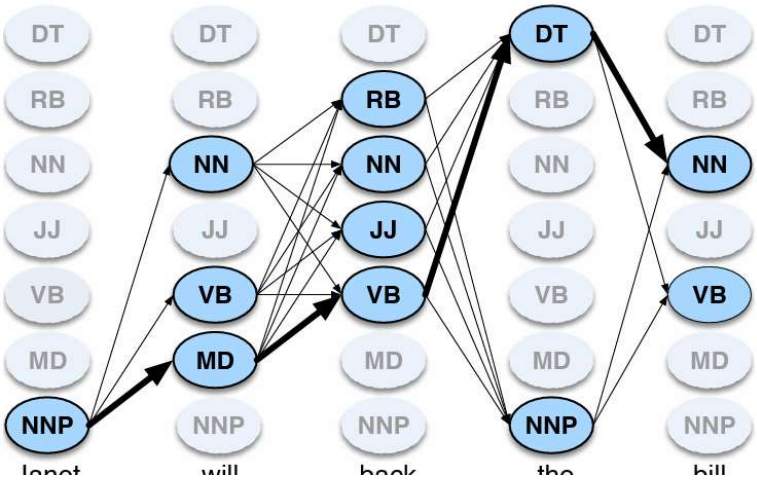
There are 3 possibilities for "back" being VB given "will" was NN or VB or MD

$V[back, VB] = \max \begin{cases} V[will, NN] \times P_t(VB|NN) \times P_e(back|VB) \\ V[will, VB] \times P_t(VB|VB) \times P_e(back|VB) \\ V[will, MD] \times P_t(VB|MD) \times P_e(back|VB) \end{cases}$

$= \max \begin{cases} 1.0512e-10 \times 0.0014 \times 0.000672 \\ 2.268e-13 \times 0.005 \times 0.000672 \\ 3.0535e-8 \times 0.7968 \times 0.000672 \end{cases} = \max \begin{cases} 9.89e-17 \\ 7.62048e-19 \\ 1.635e-11 \end{cases}$

$V[back, VB] = 1.635e-11$

similarly we can calculate V[back,JJ], V[back,NN], V[back,RB] and other viterbi values and finally we will get beside network of paths. Key idea is every time we move from one word to another we eliminate some possible paths



Question 7. HMM [4 Marks]

By using Viterbi Algorithm, find whether the word "drive" is VB or NN through computing their probabilities for the sentence, "I love to drive". The transition probabilities and the observation likelihood for this corpus are as follows:

	VB	TO	NN	PRP
<S>	0.019	0.0043	0.041	0.067
VB	0.0038	0.035	0.047	0.0070
TO	0.83	0	0.0047	0
NN	0.0040	0.016	0.087	0.0045
PRP	0.23	0.00079	0.0012	0.00014

	I	Love	to	drive
VB	0	0.0082	0	.07
TO	0	0	0.88	0
NN	0	0.0074	0	0.057
PRP	0.48	0	0	0

Assume the values for the third column of the Viterbi table which corresponds to observation 3 (word is "to") and they are VB: 0, TO: 0.018, NN: 0, PPSS: 0.
(Hint: Show the computations of the last column of the Viterbi table).

VB			0	v[4,4] = v[3,3]*P(VB|TO)*P(drive|VB) 0.018*0.83*0.07 = 0.0010458
TO			0.018 = v[3,3]	v[4,3] = v[3,3]*P(TO|TO)*P(drive|TO) 0.018*0*0 = 0
NN			0	v[4,2] = v[3,3]*P(NN|TO)*P(drive|NN) 0.018*0.0047*0.057 = 0.0000048222
PRP			0	v[4,1] = v[3,3]*P(PRP|TO)*P(drive|PRP) 0.018*0*0 = 0
	I	Love	To	Drive

v[4,4] = P(drive = VB) = 0.0010458 and v[4,2] = P(drive = NN) = 0.0000048222
Since P(drive = VB) > P(drive = NN) the word drive is VB in the given sentence

Top Down Parser

1. S → NP VP
2. NP → ART N
3. NP → ART ADJ N
4. VP → V
5. VP → V NP

1 The 2 old 3 man 4 cried 5

the: ART
old: ADJ, N
man: N, V
cried: V

Step	Current State	Backup States	Comment
1.	((S) 1)		
2.	((NP VP) 1)		S rewritten to NP VP
3.	((ART N VP) 1)	((ART ADJ N VP) 1)	NP rewritten producing two new states
4.	((N VP) 2)	((ART ADJ N VP) 1)	
5.	((VP) 3)	((ART ADJ N VP) 1)	the backup state remains
6.	((V) 3)	((V NP) 3) ((ART ADJ N VP) 1)	
7.	(( ) 4)	((V NP) 3) ((ART ADJ N VP) 1)	
8.	((V NP) 3)	((ART ADJ N VP) 1)	the first backup is chosen
9.	((NP) 4)	((ART ADJ N VP) 1)	
10.	((ART N) 4)	((ART ADJ N) 4) ((ART ADJ N VP) 1)	looking for ART at 4 fails
11.	((ART ADJ N) 4)	((ART ADJ N VP) 1)	fails again
12.	((ART ADJ N VP) 1)		now exploring backup state saved in step 3
13.	((ADJ N VP) 2)		
14.	((N VP) 3)		
15.	((VP) 4)		
16.	((V) 4)	((V NP) 4)	
17.	(( ) 5)		success!

The algorithm starts with the initial state ((S) 1) and no backup states.

1. Select the current state: Take the first state off the possibilities list and call it C. If the possibilities list is empty, then the algorithm fails (that is, no successful parse is possible).
2. If C consists of an empty symbol list and the word position is at the end of the sentence, then the algorithm succeeds.
3. Otherwise, generate the next possible states.
   3.1. If the first symbol on the symbol list of C is a lexical symbol, and the next word in the sentence can be in that class, then create a new state by removing the first symbol from the symbol list and updating the word position, and add it to the possibilities list.
   3.2. Otherwise, if the first symbol on the symbol list of C is a non-terminal, generate a new state for each rule in the grammar that can rewrite that nonterminal symbol and add them all to the possibilities list.

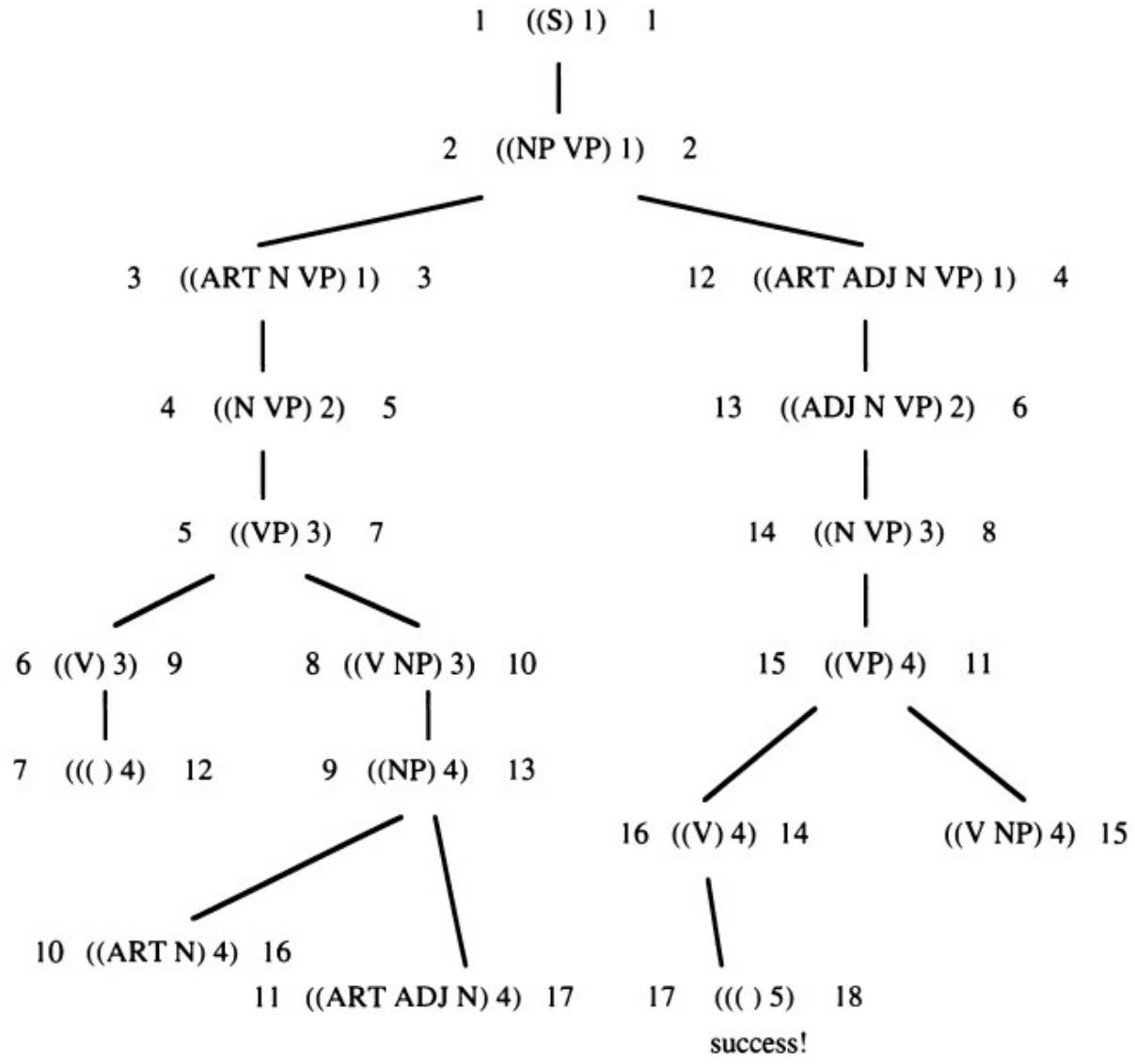


Figure 3.7 Search tree for two parse strategies (depth-first strategy on left; breadth-first on right)

Top Down Parser (More examples)

People laugh

1 2 3

These are positions

Lexicon:
People - N, V
Laugh - N, V

This indicate that both Noun and Verb is possible for the word "People"

- Grammar – S → NP VP
  NP → DT N | N
  VP → V ADV | V

	State	Backup State	Action
1.	((S) 1)	-	-
2.	((NP VP)1)	-	-
3a.	((DT N VP)1)	((N VP) 1)	Consume "People"
3b.	((N VP)1)	-	-
4.	((VP)2)	-	Consume "People"
5a.	((V ADV)2)	((V)2)	Consume "laugh"
6.	((ADV)3)	-	Consume "laugh"
5b.	((V)2)	-	Consume "laugh"
6.	((.)3)	-	Consume "laugh"

search for 1st phrase

Position of input pointer

there is only 1 production for S → NP VP

since current index word doesn't match current phrase NP != N or V expand NP

search phrase VP current word "laugh" ∈ {N,V} VP not in it ∴ expand VP

word at index 1 i.e 'people' is it Noun phrase(NP)?

"People" ∈ {N,V} by given Lexicon it's a Noun(N) or Verb(V)

DT & ∴ next state

search phrase = N & current index word "people" ∈ {N,V}

∴ Match found

∴ remove matched search phrase & index++

search phrase V is in ((V)2)

current "laugh" {N,V} ((V)2)

∴ remove search phrase V & index++

at index 3 reached end i.e empty string & it doesn't have any productions or Lexicon match so backup state is picked

backup state moved to index 2 "laugh" & search phrase(SP) V match ∴ remove V & move index++

Termination Condition : All inputs over. No symbols remaining.

Note: Input symbols can be pushed back.

"$_1$ The $_2$ old $_3$ man $_4$ cried $_5$"

1. S → NP VP
2. NP → ART N
3. NP → ART ADJ N
4. VP → V
5. VP → V NP

cried: V
old: ADJ, N
man: N, V
the: ART

Step	Current State	Backup States	Comment
1.	((S) 1)		initial position
2.	((NP VP) 1)		S rewritten to NP VP
3.	((ART N VP) 1)	((ART ADJ N VP) 1)	NP rewritten by rules 2&3
4.	((N VP) 2)	((ART ADJ N VP) 1)	
5.	((VP) 3)	((ART ADJ N VP) 1) ((V NP) 3)	
6.	((V) 3)	((ART ADJ N VP) 1) ((V NP) 3)	VP rewritten by rules 4&5
7.	(( ) 4)	((ART ADJ N VP) 1)	
8.	((V NP) 3)	((ART ADJ N VP) 1)	the first backup is chosen
9.	((NP) 4)	((ART ADJ N VP) 1)	
10.	((ART N) 4)	((ART ADJ N) 4) ((ART ADJ N VP) 1)	looking for ART fails
11.	((ART ADJ N) 4)	((ART ADJ N VP) 1)	fails again
12.	((ART ADJ N VP) 1)		exploring backup state saved in step 3
13.	((ADJ N VP) 2)		
14.	((N VP) 3)		
15.	((VP) 4)		
16.	((V) 4)	((V NP) 4)	
17.	(( ) 5)		success!

Top Down Parser

Ref https://gawron.sdsu.edu/compling/course_core/lectures/simple_top_down.htm

Think of this as an algorithm that takes one particular view of how to do a derivation. Suppose we are trying to parse the sentence *John likes Richard M. Nixon* with respect to the grammar:

- S -> NP VP | VP
- VP -> V NP | V PP
- NP -> Art (Adj) N | PN
- VP -> is/are Adj | V NP
- PP -> P NP
- PN -> Richard M Nixon | John
- V -> likes | runs | eats
- P -> in | on | under
- Art -> the | a
- Adj -> red | big
- N -> house | garden | car

Non-terminals are symbols that occur on the LHS of a rule. The non-terminals in this grammar

S, NP, VP, PP, P, Art, Adj, N, V, PN

Terminals are symbols that occur ONLY on the RHS of a rule. The non-terminals in this grammar

is are Richard M Nixon John likes runs
eats in on under the a red big

The **pre-terminals** are symbols that occur on the LHS of a rule introducing ONLY terminals. The pre-terminals are:

N Art Adj PN V P

Step	Curr. State	State Stack	Rewrite/ Found
1.	((S) (John likes R.M.N.))		initial state
2.	((NP VP) (John likes R.M.N.))	((VP) (John likes R.M.N.))	S -> NP VP (current state) S -> VP (state stack)
3.	((Art N VP) (John likes R.M.N.))	((Art Adj N VP) (John likes R.M.N.)) ((PN VP) (John likes R.M.N.)) ((VP) (John likes R.M.N.))	NP -> Art N (current state) NP -> Art Adj N (state stack) NP -> PN (state stack)
4.	((Art Adj N VP) (John likes R.M.N.))	((PN VP) (John likes R.M.N.)) ((VP) (John likes R.M.N.))	Popped state stack!! "John" is not an Art
5.	((PN VP) (John likes R.M.N.))	((VP) (John likes R.M.N.))	Popped state stack!! "John" is not an Art
6.	((VP) (likes R.M.N.))	((VP) (John likes R.M.N.))	Lexical pop! "John" is a PN
7.	((V NP) (likes R.M.N.))	((V PP) (likes R.M.N.)) ((VP) (John likes R.M.N.))	VP -> V NP VP -> V PP
8.	((NP) (R.M.N.))	((V PP) (likes R.M.N.)) ((VP) (John likes R.M.N.))	Lexical pop! "likes" is a V
9.	((Art N) (R.M.N.))	((Art Adj N) (R.M.N.)) ((PN) (R.M.N.)) ((V PP) (likes R.M.N.)) ((VP) (John likes R.M.N.))	NP -> Art N (current state) NP -> Art Adj N (backup state) NP -> PN (backup state)
10.	((Art Adj N) (R.M.N.))	((PN) (R.M.N.)) ((V PP) (likes R.M.N.)) ((VP) (John likes R.M.N.))	Popped state stack!! "Richard" is not an Art
11.	((PN) (R.M.N.))	((V PP) (likes R.M.N.)) ((VP) (John likes R.M.N.))	Popped state stack!! "Richard" is not an Art
12.	(() ())	((V PP) (likes R.M.N.)) ((VP) (John likes R.M.N.))	Lexical pop! "R.M.N." is a PN Done!!

Bottom up Chart Parser

For rule NP -> ART ADJ N In a bottom-up parser you use the rule to take a sequence ART ADJ N that you have found and identify it as an NP. a simple bottom-up parser would be expensive, as the parser would tend to try the same matches again and again, thus duplicating much of its work unnecessarily

To avoid this problem, a data structure called a **chart** is introduced that allows the parser to store the partial results of the matching it has done so far so that the work need not be reduplicated.

Sentence:

"The Large can can hold the water"

Lexicons:

The -> ART
Large -> ADJ
Can -> N, V, AUX
Hold -> N, V
Water -> N, V

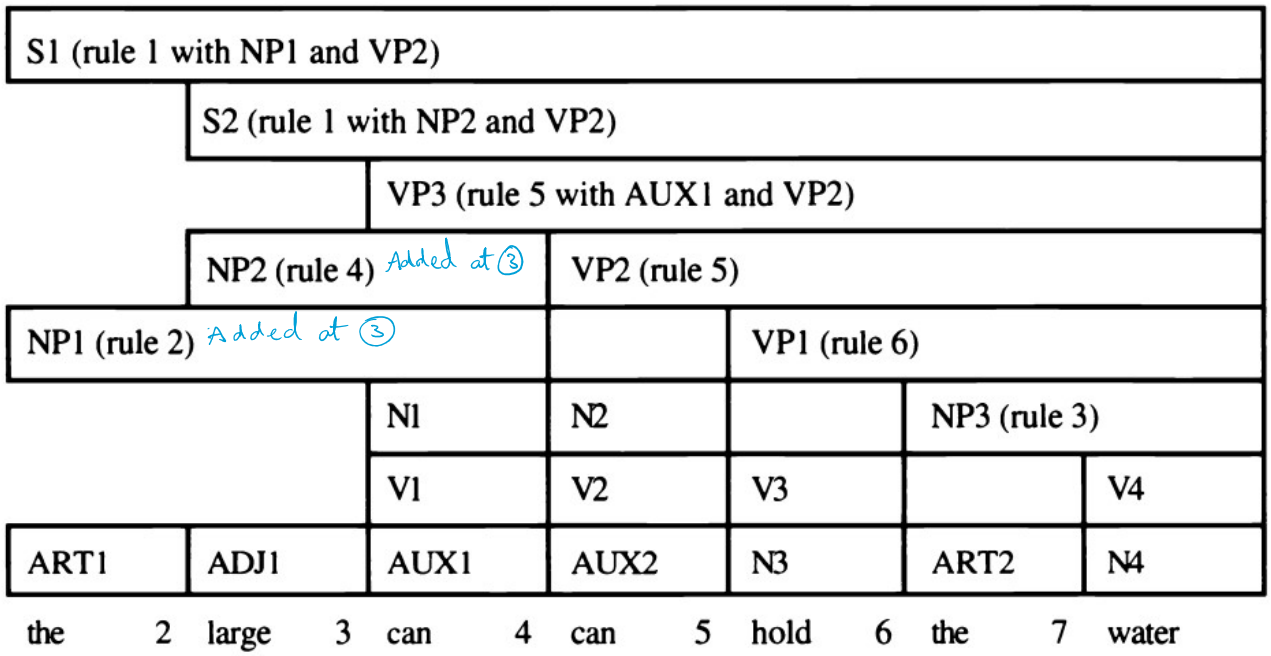
Grammar Rule (G_Rule):

1. S → NP VP
2. NP → ART ADJ N
3. NP → ART N
4. NP → ADJ N
5. VP → AUX VP
6. VP → V NP

```
INITIALIZE:
set Agenda = list of all possible categories of each input word
             (in order of input);
set n = length of input;
set Chart = ();

ITERATE:
loop
    if Agenda = () then
        if there is at least one S constituent from 0 to n then
            return SUCCESS ☺
        else
            return FAIL ☹
        end if
    else ...
    Set C_i,j = First(Agenda);  /* Remove first item from agenda. */
        /* C_i,j is a completed constituent of type C from position i to position j */
    Add C_i,j to Chart;

    ARC UPDATE:
        a. BOTTOM-UP ARC ADDITION (PREDICTION):
            for each grammar rule X → C X1 ... XN do
                Add arc X → C • X1 ... XN, from i to j, to Chart;
        b. ARC EXTENSION (FUNDAMENTAL RULE):
            for each arc X → X1 ... • C ... XN, from k to i, do
                Add arc X → X1 ... C • ... XN, from k to j, to Chart;
        c. ARC COMPLETION:
            for each arc X → X1 ... XN C • added in step (a) or step (b) do
                Move completed constituent X to Agenda;
    end if
end loop
```



① "the" is a ART pick grammar production starting with ART i.e rule 2 & 3 ADD new ARCs

A1: NP → ART • ADJ N (1 → 2) • indicates that lexicons to left are seen/encountered and the ones to right are yet to be seen.

A2: NP → ART • N (1 → 2)

- No ARCs can be joined & No ARCs have completed so continue loop

---

② "large" is an ADJ, pick production starting with ADJ i.e rule 4 & ADD a new ARC

A3: NP → ADJ • N (2 → 3) In this ARC an ADJ is seen and one of previous ARC is searching for an ADJ so these two ARC's can be joined

A4: A1+A3 (from A1 ← NP → ART ADJ • N → from A3) (1 → 3) No ARC completed so continue loop

---

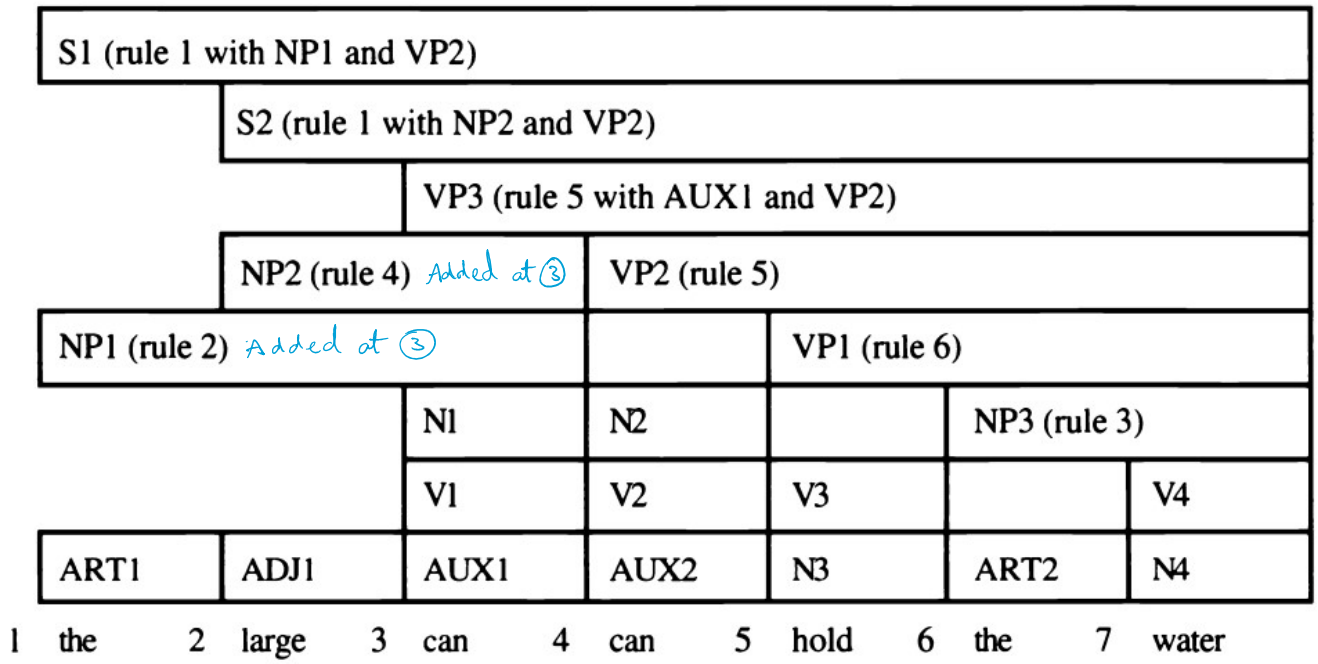
③ "can" is an N, V, AUX so production rules 5 & 6 are picked & ADD new ARCs

Also previous ARC's A3 & A4 also looking for lexicon N till current index 3 since "can" is N

new A4: NP → ART ADJ N • (1 → 3) since • pointer reached end means entire production is seen & its LHS can been added to agenda for respective indexes

new A3: NP → ADJ N •

Bottom up Chart Parser (Contd.)



A1: NP → ART ∘ ADJ N (1 → 2)

A2: NP → ART ∘ N (1 → 2)

A3: NP → ADJ ∘ N (2 → 3)

A4: A1 + A3 (from A1, from A3) NP → ART ADJ ∘ N (1 → 3)

new A4: NP → ART ADJ N ∘ (1 → 3) ∘ reached end ∴ add LHS to agenda

new A3: NP → ADJ N ∘ (2 → 4) ∘ reached end ∴ add LHS to agenda

continue ③

NP1 + NP2 {NP at core}

since new lexicons have been added to agenda we have to look for productions starting with NP

A5: S → NP ∘ VP (1 → 4)

A6: S → NP ∘ VP (2 → 4)

also add ARC's for original lexicons for "can"

A7: VP → AUX ∘ VP (3 → 4)

A8: VP → V ∘ NP (3 → 4)

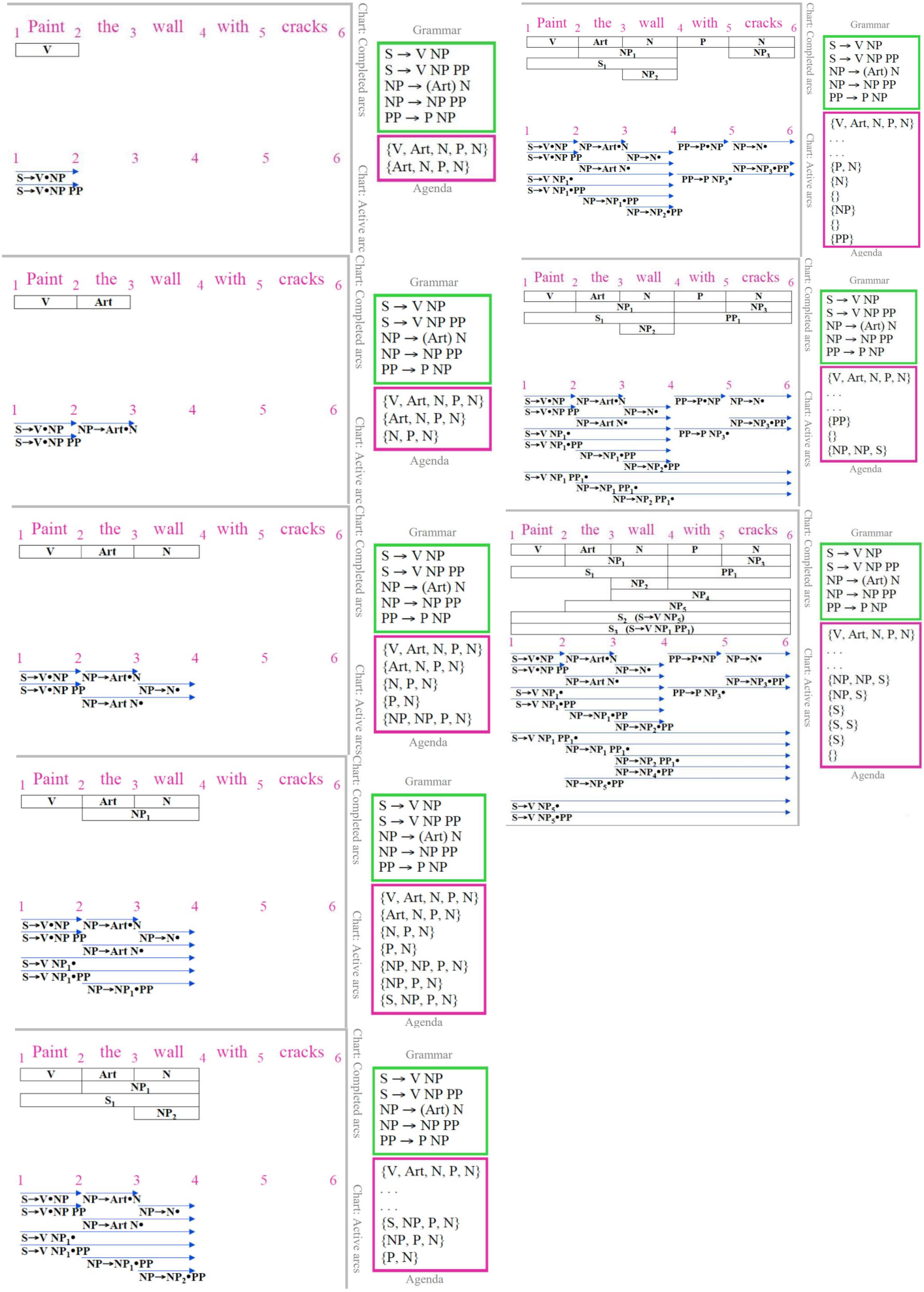
these ARC's didn't reach end nor can be joined so continue

---

④ .... continue ARC Addition
ARC extension
& ARC completion

until starting non-terminal 'S' is added to agenda spaning from start index to end index

Bottom up Chart Parser (more examples)



Top Down Chart Parsing/ Earley Parsing

```
function EARLEY-PARSE(words,grammar) returns chart

    ENQUEUE((γ → •S, [0,0]),chart[0])
    for i ← from 0 to LENGTH(words) do
        for each state in chart[i] do
            if INCOMPLETE?(state) and NEXT-CAT(state) is not POS then
                PREDICTOR(state)
            elseif INCOMPLETE?(state) and NEXT-CAT(state) is POS then
                SCANNER(state)
            else
                COMPLETER(state)
        end
    end
    return(chart)
```

A → aA | ε

Non-terminal symbols; these are symbols that may be replaced (further expanded).

Terminal symbols; these are symbols that may not be replaced.

R7 Det → that | this | a | the
R8 Noun → book | flight | meal | money
R9 Verb → book | include | prefer

Rule (R)

1 S → NP VP
2 S → VP
3 NP → Det Nominal
4 Nominal → Noun
5 VP → Verb
6 VP → Verb NP

• Book that flight.

Chart	State	Rule	Start, End	Added By
0	S0	γ → • S	0, 0	Start State
0	S1	S → • NP VP	0, 0	Predictor R1
0	S2	S → • VP	0, 0	Predictor R2
0	S3	NP → • Det Nominal	0, 0	Predictor R3
0	S4	VP → • Verb	0, 0	Predictor R5
0	S5	VP → • Verb NP	0, 0	Predictor R6

both pick R9 } Verb → • book

Book • that flight.

if pos right of • then get all rules starting with that pos

1	S6	Verb → book •	0, 1	Scanner
1	S7	VP → Verb •	0, 1	Completer
1	S8	VP → Verb • NP	0, 1	Completer
1	S9	S → VP •	0, 1	Completer
1	S10	NP → • Det Nominal	1, 1	Predictor

this also moves • in parents as well → gets Det → • that

Book that • flight.

Since "book" is not Pos tag but word so scanner moves • one position ahead

2	S11	Det → that •	1, 2	Scanner
2	S12	NP → Det • Nominal	1, 2	Completer
2	S13	Nominal → • Noun	2, 2	Predictor

Book that flight. •

keep reapting above steps until • hasn't reached end of sentance

3	S14	Noun → flight •	2, 3	Scanner
3	S15	Nominal → Noun •	2, 3	Completer
3	S16	NP → Det Nominal •	1, 3	Completer
3	S17	VP → Verb NP •	0, 3	Completer
3	S18	S → VP •	0, 3	Completer

Top Down Chart Parsing/ Earley Parsing

```
S   → NP VBD
S   → NP VP
NP  → DT NN
VP  → VBZ NN
```

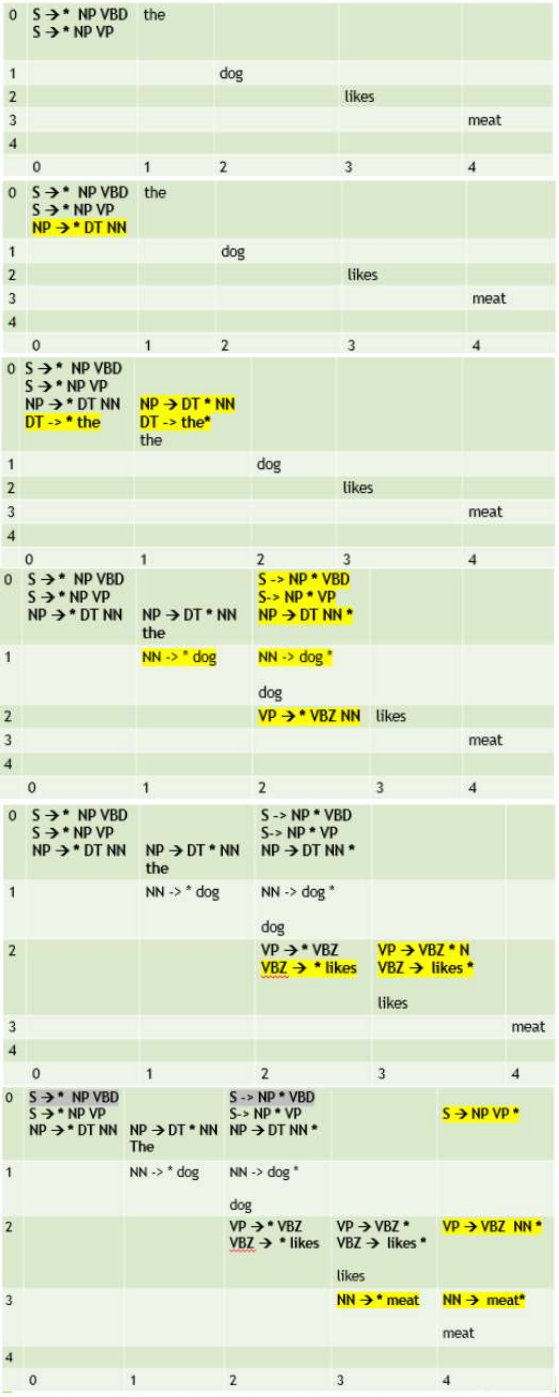
Top-down initialization: When category S is the root category of the grammar, then for every rule with S on the LHS, create an empty edge at position [0:0] with a dot leftmost on the RHS. That is, whenever S -> A B, for some A, add an edge [0:0] S -> * A B

Top-down prediction rule: When category X is expected (i.e., just to the right of a dot) then for all rules where X is on the LHS, add a new edge where the dot is leftmost on the RHS with a span of length zero at the current end point. (This rule will be repeated multiple times until no new predictions can be made). That is, whenever ([i:k] A -> W * X Y) and X -> S T, for some X, add a new edge [k:k] X -> * S T. (Note: W can be empty.)

Fundamental rule: When a new category is seen or nonterminal created, then for all active rules where the category is leftmost on the RHS of the dot, and the spans are adjacent, create a new edge with the dot moved to the left of that category and combine the spans. That is, whenever ([i:j], A -> X * Y Z and [j:k] Y -> S T * , for some j and Y, add a new edge [i:k], A -> X Y * Z (Note: X and Z can be empty.)

```
NN  → dog | meat
VBD → barked
VBZ → likes
DT  → the
```

"the dog likes meat".



[0:0] S → * NP VBD
[0:0] S → * NP VP

For each of the sentence rules, make a top-down prediction to create an empty, active edge.

[0:0] NP → * DT NN

Make more top-down predictions to create active edges for each of the nonterminal categories just to the right of the dot.

[0:1] DT → the *
[0:1] NP → DT * NN

Predict *the* and use the fundamental rule to create new edges where the dot in the DT rule and in NP rule move to the right.

[1:1] NN → * dog
[1:2] NN → dog *
[0:2] NP → DT NN *
[0:2] S → NP * VBD
[0:2] S → NP * VP
[2:2] VP → * VBZ NN

Predict *dog;* apply fundamental rule and then make a top down prediction for a VP (using the second S rule.)

[2:2] VBZ → * likes
[2:3] VBZ → likes *
[2:3] VP → VBZ * NN

Predict *likes;* apply the fundamental rule to create VBZ-> likes *, and again, to add VP -> VBZ * NN

[3:3] NN → * meat
[3:4] NN → meat*
[2:4] VP → VBZ NN *
[0:4] S → NP VP *

Predict *meat* apply the fundamental rule for meat as a noun (NN) and again to extend the active VP edge, to get VP -> VBZ NN *. Finally, use the fundamental rule to extend the active S edge, which is now complete.

CFG to CNF

Requires input grammar based on Chomsky Normal Form
- A CNF grammar is a Context-Free Grammar in which:
  - Every rule LHS is a non-terminal
  - Every rule RHS consists of either a single terminal or two non-terminals.
  - Examples:
    - ~~A → BC~~
    - NP → N PP
    - A → a
    - Noun → man
  - But not:
    - NP → the N
    - S → VP

To convert a CFG to CNF, we need to deal with three issues:
1. Rules that mix terminals and non-terminals on the RHS
   - E.g. NP → *the* Nominal
2. Rules with a single non-terminal on the RHS (called unit productions)
   - E.g. NP → Nominal
3. Rules which have more than two items on the RHS
   - E.g. NP → Det Noun PP

8

1. Rules that mix terminals and non-terminals on the RHS
   - E.g. NP → *the* Nominal
   - Solution:
     - Introduce a dummy non-terminal to cover the original terminal
       - E.g. Det → the
     - Re-write the original rule:
       - NP → Det Nominal

2. Rules with a single non-terminal on the RHS (called unit productions)
   - E.g. NP → Nominal
   - Solution:
     - Find all rules that have the form Nominal → ...
       - Nominal → Noun PP
       - Nominal → Det Noun
     - Re-write the above rule several times to eliminate the intermediate non-terminal:
       - NP → Noun PP
       - NP → Det Noun

3. Rules which have more than two items on the RHS
   - E.g. NP → Det Noun PP

Solution:
- Introduce new non-terminals to spread the sequence on the RHS over more than 1 rule.
  - Nominal → Noun PP
  - NP → Det Nominal

Example:
Check if grammar is in CNF or not, if not convert
S -> NP VP
NP -> ART ADJ N (Rule 3 more than 2 non terminal)
NP -> a ART N (Rule 1 mix of term & non term)
NP -> ADJ N
P -> VP (Rule 2 unit productions)
VP -> AUX VP
VP -> V NP

(Rule 3) let { let DUM1 -> ART ADJ } then
NP -> DUM1 N
(Rule 1) let { let DUM2 -> a } then
NP -> DUM2 ART N
(Rule 3) NP -> DUM3 N,
DUM3 -> DUM2 ART
(Rule 2) { P -> VP -> AUX VP == P -> AUX VP }
{ P -> AUX VP }
{ P -> V NP }

CNF Grammar

If we parse a sentence with a CNF grammar, we know that:
- Every phrase-level non-terminal (above the part of speech level) will have exactly 2 daughters.
  - NP → Det N
- Every part-of-speech level non-terminal will have exactly 1 daughter, and that daughter is a terminal:
  - N → lady

S -> NP VP
DUM1 -> ART ADJ
NP -> DUM1 N
DUM2 -> a
DUM3 -> DUM2 ART
NP -> DUM3 N
NP -> ADJ N
P -> AUX VP
P -> V NP

CYK Algorithm

function CKY-Parse(*words*, *grammar*) **returns** *table* **for**

j ←**from** 1 **to** LENGTH(*words*) **do** — loop over the columns

  table[*j* − 1, *j*] ← {*A* | *A* → *words*[*j*] ∈ *grammar*} — fill bottom cell

  for *i* ←**from** *j* − 2 **downto** 0 **do** — fill row *i* in column *j*

    for *k* ← *i* + 1 **to** *j* − 1 **do** — loop over split locations between *i* and *j*

      table[*i*, *j*] ← *table*[*i*, *j*] ∪ {*A* | *A* → *BC* ∈ *grammar*, *B* ∈ *table*[*i*, *k*], *C* ∈ *table*[*k*, *j*]}

Check the grammar for rules that link the constituents in [*i*, *k*] with those in [*k*, *j*]. For each rule found store LHS in cell [*i*, *j*].

S → NP VP
NP → Det N
VP → V NP
V → includes
Det → the
Det → a
N → meal
N → flight

CKY: lexical step (j = 1)

	1	2	3	4	5
0	Det				
1					
2					
3					
4					
5					

Lexical lookup
- Matches Det → the

The flight includes a meal.

CKY: lexical step (j = 2)

	1	2	3	4	5
0	Det				
1		N			
2					
3					
4					
5					

Lexical lookup
- Matches N → flight

The flight includes a meal.

CKY: syntactic step (j = 2)

	1	2	3	4	5
0	Det	NP			
1		N			
2					
3					
4					
5					

Syntactic lookup:
- look backwards and see if there is any rule that will cover what we've done so far.

The flight includes a meal.

CKY: lexical step (j = 3)

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		
3					
4					
5					

Lexical lookup
- Matches V → includes

The flight includes a meal.

CKY: lexical step (j = 3)

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		
3					
4					
5					

Syntactic lookup
- There are no rules in our grammar that will cover Det, NP, V

The flight includes a meal.

CKY: lexical step (j = 4)

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		
3				Det	
4					
5					

Lexical lookup
- Matches Det → a

The flight includes a meal.

CKY: lexical step (j = 5)

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		
3				Det	
4					N

Lexical lookup
- Matches N → meal

The flight includes a meal.

CYK (contd.)

CKY: syntactic step (j = 5)

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		
3				Det	NP
4					N

Syntactic lookup
- We find that we have NP → Det N

The flight includes a meal.

CKY: syntactic step (j = 5)

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		VP
3				Det	NP
4					N

Syntactic lookup
- We find that we have VP → V NP

The flight includes a meal.

CKY: syntactic step (j = 5)

	1	2	3	4	5
0	Det	NP			S
1		N			
2			V		VP
3				Det	NP
4					N

Syntactic lookup
- We find that we have S → NP VP

The flight includes a meal.

PCFG

A PCFG consists of:

- A set of terminals, $\{w^k\}$, k= 1,...,V
- A set of nonterminals, N^i, i= 1,..., n
- A designated start symbol N^1
- A set of rules, $\{N^i \rightarrow \xi^j\}$, (where ξ^j is a sequence of terminals and nonterminals)
- A corresponding set of probabilities on rules such that: $\forall i \ \Sigma_j P(N^i \rightarrow \xi^j) = 1$

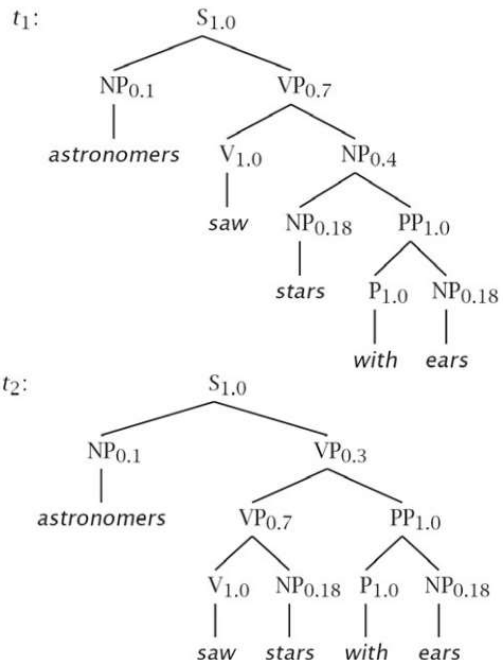
S → NP VP	1.0	NP → NP PP	0.4
PP → P NP	1.0	NP → *astronomers*	0.1
VP → V NP	0.7	NP → *ears*	0.18
VP → VP PP	0.3	NP → *saw*	0.04
P → *with*	1.0	NP → *stars*	0.18
V → *saw*	1.0	NP → *telescopes*	0.1

Terminals

with, saw, astronomers, ears, stars, telescopes

Nonterminals S, PP, P, NP, VP, V

Start symbol S



$P(t_1) = 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \times 0.18 \times 1.0 \times 1.0 \times 0.18$

$P(t_2) = 0.0009072$

$P(w_{15}) = 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \times 0.18 \times 1.0 \times 1.0 \times 0.18$

$= 0.0006804$

$= P(t_1) + P(t_2) = 0.0015876$

Probabilistic CYK

	1	2	3	4	5
0	Det (.4)				
1					
2					
3					
4					
5					

- S → NP VP [.80]
- NP → Det N [.30]
- VP → V NP [.20]
- V → includes [.05]
- Det → the [.4]
- Det → a [.4]
- N → meal [.01]
- N → flight [.02]

The flight includes a meal.

S → NP VP [.80]
NP → Det N [.30]
VP → V NP [.20]
V → includes [.05]
Det → the [.4]
Det → a [.4]
N → meal [.01]
N → flight [.02]

	1	2	3	4	5
0	Det (.4)				
1		N .02			
2					
3					
4					
5					

- S → NP VP [.80]
- NP → Det N [.30]
- VP → V NP [.20]
- V → includes [.05]
- Det → the [.4]
- Det → a [.4]
- N → meal [.01]
- N → flight [.02]

The flight includes a meal.

	1	2	3	4	5
0	Det (.4)	NP .0024			
1		N .02			
2					
3					
4					
5					

- S → NP VP [.80]
- NP → Det N [.30]
- VP → V NP [.20]
- V → includes [.05]
- Det → the [.4]
- Det → a [.4]
- N → meal [.01]
- N → flight [.02]

The flight includes a meal.

Note: probability of NP in [0,2]
P(Det → the) * P(N → flight) * P(NP → Det N)

0.4 x 0.02 x 0.3 = 0.0024

S → NP VP [.80]
NP → Det N [.30]
VP → V NP [.20]
V → includes [.05]
Det → the [.4]
Det → a [.4]
N → meal [.01]
N → flight [.02]

	1	2	3	4	5
0	Det (.4)	NP .0024			
1		N .02			
2			V .05		
3					
4					
5					

The flight includes a meal.

S → NP VP [.80]
NP → Det N [.30]
VP → V NP [.20]
V → includes [.05]
Det → the [.4]
Det → a [.4]
N → meal [.01]
N → flight [.02]

	1	2	3	4	5
0	Det (.4)	NP .0024			
1		N .02			
2			V .05		
3				Det .4	
4					N .01

The flight includes a meal.

S → NP VP [.80]
NP → Det N [.30]
VP → V NP [.20]
V → includes [.05]
Det → the [.4]
Det → a [.4]
N → meal [.01]
N → flight [.02]

	1	2	3	4	5
0	Det (.4)	NP .0024	X		
1		N .02	X	X	
2		{N V} no production	V .05	X	VP 0.0001
3			Det .4	NP .001	
4			{V Det}	N .01	

no production

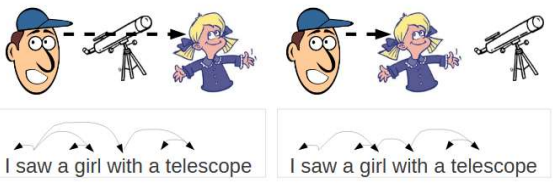
The flight includes a meal.

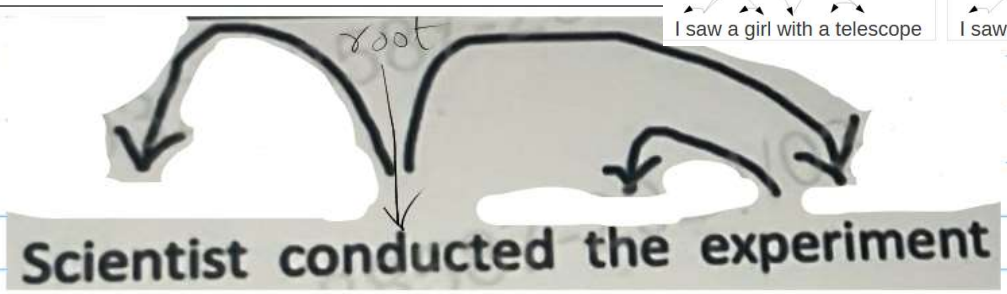
P(Det -> a) x P(N->meal) x P(NP->Det N)
0.4 x 0.01 x 0.3 = 0.0012

The flight includes a meal.

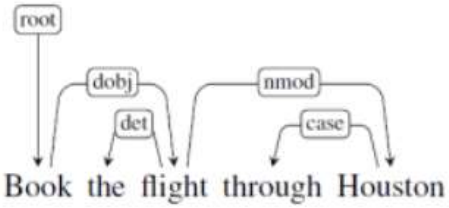
For cell [0,3] check pairs of non terms from cells [0,1] & [1,3] as [1,3] is empty check next
[0,2] & [2,3] is NP V there is no production
For cell [1,4] check pairs of non terms from cells [1,2] & [2,4] as [2,4] is empty check next
[1,3] & [3,4] as [1,3] is empty check next
For cell [2,5] check pairs of non terms from cells [2,3] & [3,5] is V NP exists VP -> V NP
P([2,5]) = P(V->includes)xP(N->meal)xP(VP->V NP) = 0.5x0.0012x0.2=0.0001
[2,4] & [4,5] as [2,4] is empty check next
If there were multiple probabilities then take P([2,5]) = max{P([2,5])_1, P([2,5])_2}

Dependency Parsing/Arc Standard





transition	stack	Buffer	Action	Relation
0	[root]	[scientist, conducted, the, experiment]	shift	
1	[root, scientist]	[conducted, the, experiment]	shift	
2	[root, scientist, conducted]	[the, experiment]	Left-Arc	scientist ← conducted
3	[root, conducted]	[the, experiment]	shift	
4.	[root, conducted, the]	[experiment]	shift	
5.	[root, conducted, the, experiment]	[ ]	Left-Arc	the ← experiment
6.	[root, conducted, experiment]	[ ]	right Arc	conducted → experiment
7	[root, conducted]	[ ]	right Arc	root → conducted
8	[root]	[ ]	Done	



Step	Stack	Word List	Action	Relation Added
0	[root]	[book, the, flight, through, houston]	RIGHTARC	(root → book)
1	[root, book]	[the, flight, through, houston]	SHIFT	
2	[root, book, the]	[flight, through, houston]	LEFTARC	(the ← flight)
3	[root, book]	[flight, through, houston]	RIGHTARC	(book → flight)
4	[root, book, flight]	[through, houston]	SHIFT	
5	[root, book, flight, through]	[houston]	LEFTARC	(through ← houston)
6	[root, book, flight]	[houston]	RIGHTARC	(flight → houston)
7	[root, book, flight, houston]	[]	REDUCE	
8	[root, book, flight]	[]	REDUCE	
9	[root, book]	[]	REDUCE	
10	[root]	[]	Done	

Oracle for Arc Eager parsing

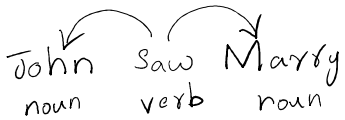
```
LEARN(\{T_1, \ldots, T_N\})
1   w \leftarrow 0.0
2   for i in 1..K
3     for j in 1..N
4       c \leftarrow ([]_S, [w_1, \ldots, w_{n_j}]_B, \{\})
5       while B_c \neq []
6         t^* \leftarrow \arg\max_t w.f(c,t)
7         t_o \leftarrow o(c, T_i)
8         if t^* \neq t_o
9           w \leftarrow w + f(c,t_o) - f(c,t^*)
10        c \leftarrow t_o(c)
11  return w
```

Oracle $o(c, T_i)$ returns the optimal transition of c and T_i

Consider the sentence, 'John saw Mary'. Draw a dependency graph for this sentence.

Assume that you are learning a classifier for the data-driven deterministic parsing and the above sentence is a gold-standard parse in your training data. You are also given that *John* and *Mary* are 'Nouns', while the POS tag of *saw* is 'Verb'. Assume that your features correspond to the following conditions:

- c_0 The stack is empty
- c_1 Top of stack is Noun and Top of buffer is Verb
- c_2 Top of stack is Verb and Top of buffer is Noun



Initialize the weights of all your features to *5.0*, except that in all of the above cases, you give a weight of *5.5* to *Left-Arc*. Define your feature vector and the initial weight vector.

Use this gold standard parse during online learning and report the weights after completing one full iteration of Arc-Eager parsing over this sentence.

Operations: Left-Arc LA, Right-Arc RA, Shift SH, Reduce RE

F(c,t) = [(c0,LA), (c1,LA), (c2,LA)|(c0,RA), (c1,RA), (c2,RA)|(c0,SH), (c1,SH), (c2,SH)|(c0,RE), (c1,RE), (c2,RE)]

W = [5.5,5.5,5.5|5.0,5.0,5.0|5.0,5.0,5.0|5.0,5.0,5.0]

First iteration:

Stack	Buffer
[ ]	[John, Saw, Marry]

stack is empty which corresponds to feature c0 being true, thus

LA RA SH RE

F(c, LA) = [1,0,0 | 0,0,0 | 0,0,0 | 0,0,0]

F(c, RA) = [0,0,0 | 1,0,0 | 0,0,0 | 0,0,0]

F(c, SH) = [0,0,0 | 0,0,0 | 1,0,0 | 0,0,0]

F(c, RE) = [0,0,0 | 0,0,0 | 0,0,0 | 1,0,0]

t* = argmax(W * F(c,t))

W * F(c,LA) = sum([5.5 * 1,5.5 * 0,5.5 * 0|5.0 * 0,5.0 * 0,5.0 * 0|5.0 * 0,5.0 * 0,5.0 * 0|5.0 * 0,5.0 * 0,5.0 * 0])
= sum([5.5,0,0,0,0,0,0,0,0,0,0,0]) = 5.5

W * F(c,RA) = sum([5.5 * 0,5.5 * 0,5.5 * 0|5.0 * 1,5.0 * 0,5.0 * 0|5.0 * 0,5.0 * 0,5.0 * 0|5.0 * 0,5.0 * 0,5.0 * 0])
= sum([0,0,0,5.0,0,0,0,0,0,0,0,0]) = 5.0

W * F(c,SH) = sum([5.5 * 0,5.5 * 0,5.5 * 0|5.0 * 0,5.0 * 0,5.0 * 0|5.0 * 1,5.0 * 0,5.0 * 0|5.0 * 0,5.0 * 0,5.0 * 0])
= sum([0,0,0,0,0,0,5.0,0,0,0,0,0]) = 5.0

W * F(c,RE) = sum([5.5 * 0,5.5 * 0,5.5 * 0|5.0 * 0,5.0 * 0,5.0 * 0|5.0 * 0,5.0 * 0,5.0 * 0|5.0 * 1,5.0 * 0,5.0 * 0])
= sum([0,0,0,0,0,0,0,0,0,5.0,0,0]) = 5.0

It was LA operation which gave us the maximum value of 5.5 therefore t* = LA

As per oracle the optimal transition t0 is SH (when len(stack) > 0 else if len(stack) == 0 then t0 is RE)

Updating weights

W = W + F(c,t0) - F(c,t*)

= [5.5,5.5,5.5|5.0,5.0,5.0|5.0,5.0,5.0|5.0,5.0,5.0] + [0,0,0 | 0,0,0 | 1,0,0 | 0,0,0] - [1,0,0 | 0,0,0 | 0,0,0 | 0,0,0]

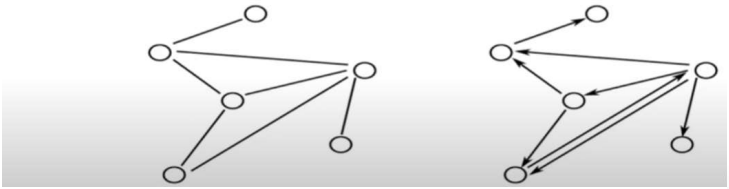
W = [4.5,5.5,5.5|5.0,5.0,5.0|6.0,5.0,5.0|5.0,5.0,5.0]

Iteration 1 done.

Graph-Based Dependency Parsing

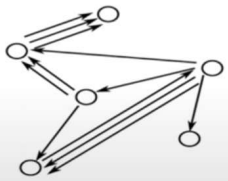
Some Graph Theory Reminders

- A graph $G = (V, A)$ is a set of vertices V and arcs $(i,j) \in A$ where $i, j \in V$.
- Undirected graphs: $(i,j) \in A \Leftrightarrow (j,i) \in A$
- Directed graphs (digraphs) : $(i,j) \in A \nRightarrow (j,i) \in A$



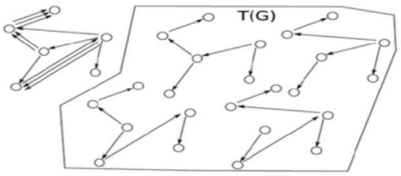
Multi-Digraphs

- A multi-digraph is a digraph where multiple arcs between vertices are possible
- $(i,j,k) \in A$ represents the k^{th} arc from vertex i to vertex j.



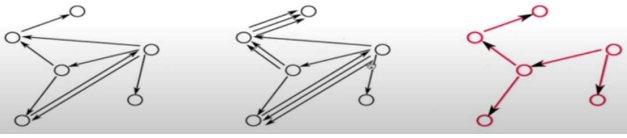
Maximum Spanning Trees (MST)

Let $T(G)$ be the set of all spanning trees for graph G



Directed Spanning Trees

- A directed spanning tree of a (multi-)digraph $G = (V, A)$ is a subgraph $G' = (V', A')$ such that:
  - $V' = V$
  - $A' \subseteq A$, and $|A'| = |V'| - 1$
  - G' is a tree (acyclic)
- A spanning tree of the following (multi-)digraphs

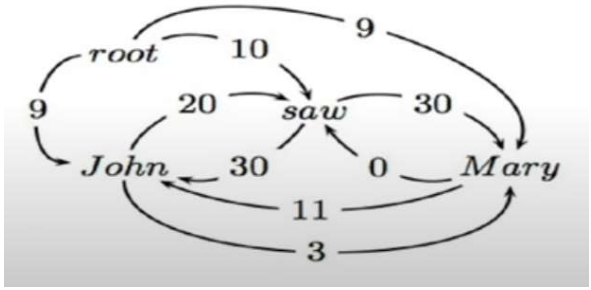


The MST problem

Find the spanning tree G' of the graph G that has the highest weight

$$G' = \arg\max_{G' \in T(G)} w(G') = \arg\max_{G' \in T(G)} \sum_{(i,j,k) \in G'} w_{ij}{}^{k}$$

x = John saw Mary



Chu-Liu-Edmonds algorithm

function MAXSPANNINGTREE(G=(V,E), $root$, $score$) **returns** *spanning tree*

```
F ← []
T' ← []
score' ← []
for each v ∈ V do
    bestInEdge ← argmax_{e=(u,v)∈E} score[e]
    F ← F ∪ bestInEdge
    for each e=(u,v) ∈ E do
        score'[e] ← score[e] − score[bestInEdge]
if T=(V,F) is a spanning tree then return it
else
    C ← a cycle in F
    G' ← CONTRACT(G, C)
    T' ← MAXSPANNINGTREE(G', root, score')
    T ← EXPAND(T', C)
    return T
```

select best incoming edge for each node

subtract its score from all incoming edges

stopping condition

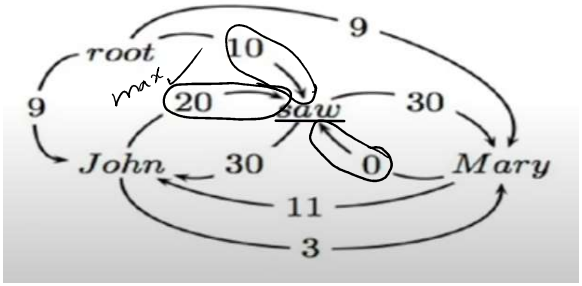
contract nodes if there are cycles

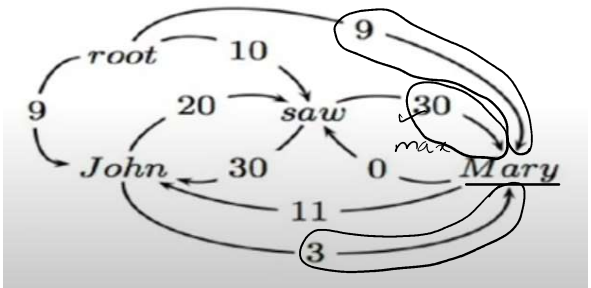
recursively compute MST

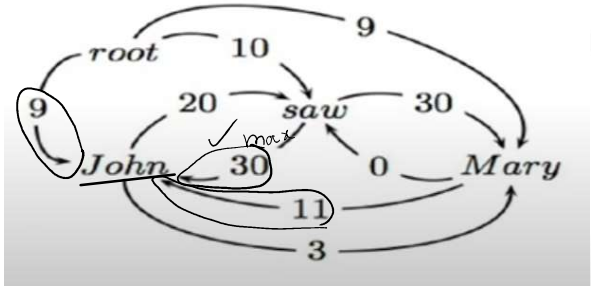
expand contracted nodes

function CONTRACT(G, C) **returns** *contracted graph*

function EXPAND(T, C) **returns** *expanded graph*

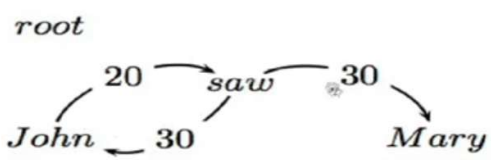




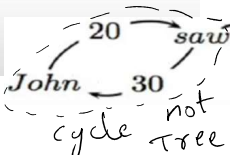


Find the highest scoring incoming arc for each vertex

no incoming edges to root

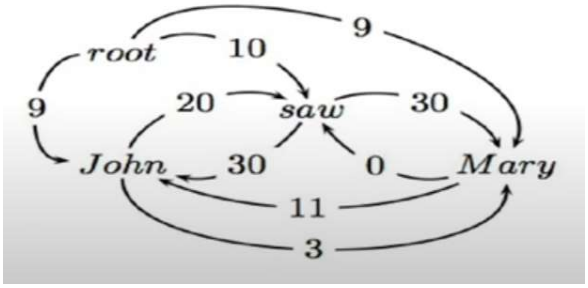


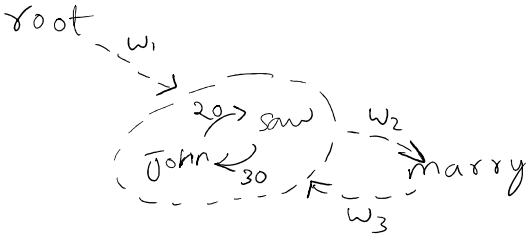
If this is a tree, then we have found MST.



cycle not tree

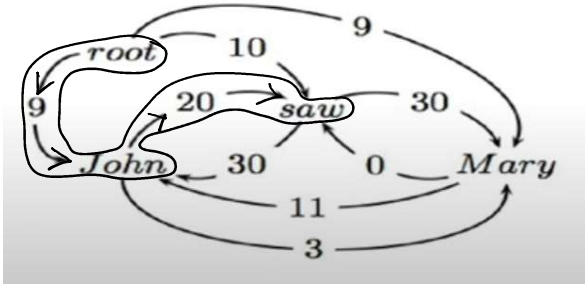
since there is cycle contract i.e make nodes john & saw as one new temparary node



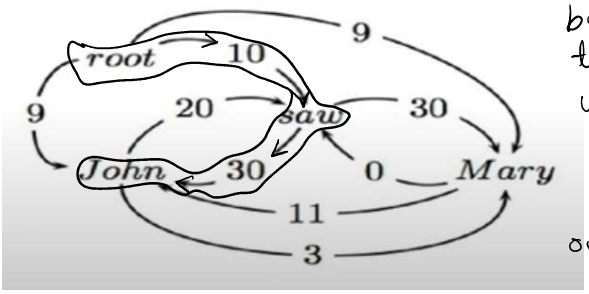


since its a new node it may have incoming & outgoing edges to every other non-root nodes.

if we consider unique paths b/w root, john, saw with no cycles & in single direction there are 2 ways



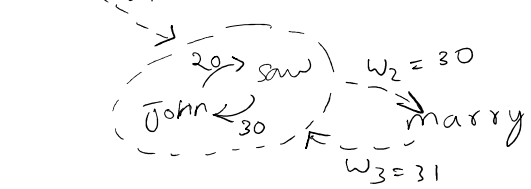
root → john → saw
9 20
29



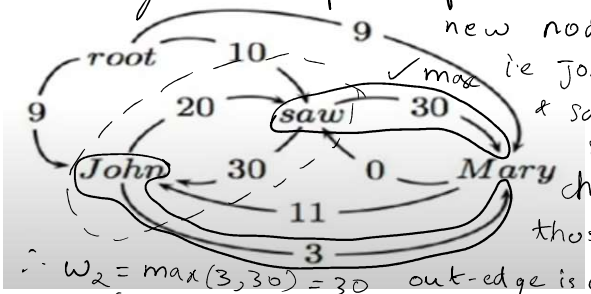
root → saw → john
10 30
40

both these ways ends up in our new node

weight for root to new node w_1 = max (29, 40) = 40

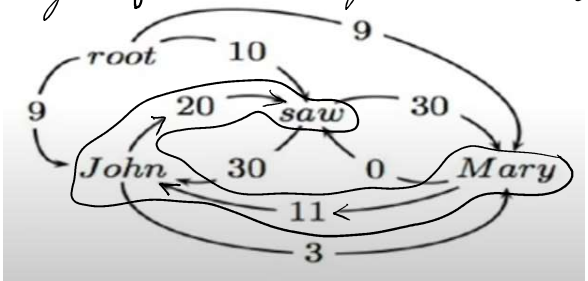


to find w_2 i.e outgoing edge from irrespective of unique paths there are only 2 exit points from this new node ie john & saw so checking those 2

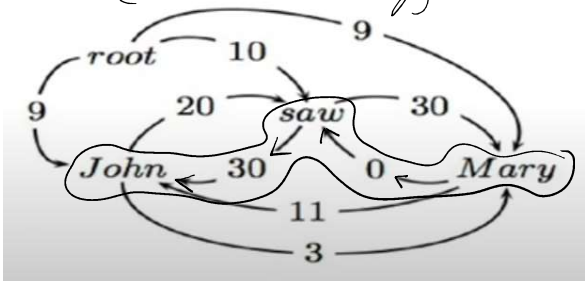


$\therefore w_2 = \max(3, 30) = 30$ out-edge is enough {saw ⟶ marry}

to find w_3 weight of incoming edge from marry to new node



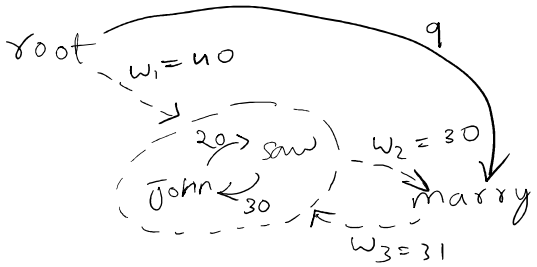
marry → john → saw
11 20
31

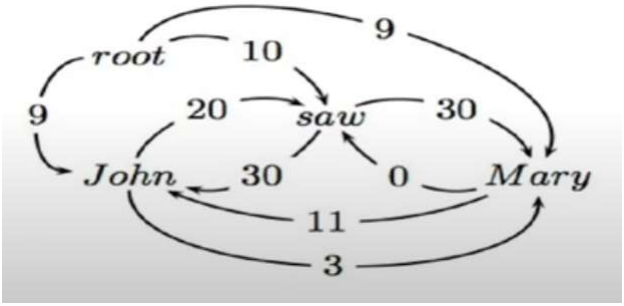


marry → saw → john
0 30
30

$w_3 = \max(30, 31) = 31$

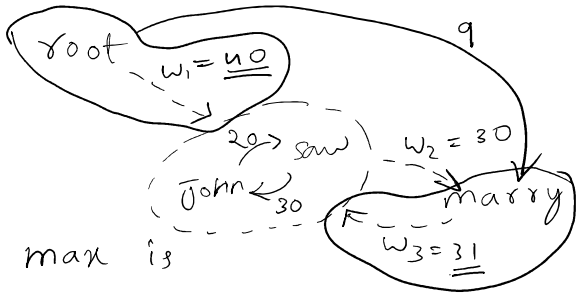
Its still has cycles thus not a tree yet





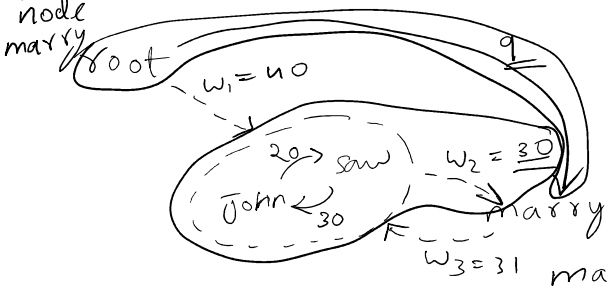
To get max spanning tree MST choose max weighted incoming edge for every node

for new node



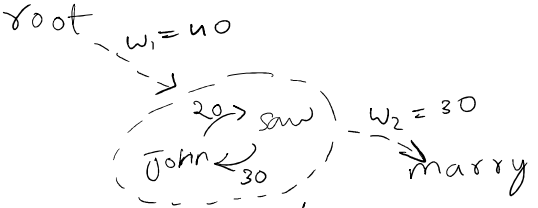
max is $w_1 = 40$

for node marry



max is $w_2 = 30$

after preserving max In edges



no further contraction possible

we still have 1 contracted node so start expanding

$w_2 = 30$ is contributed by original edge saw $\xrightarrow{30}$ marry

$w_1 = 40$ contributed by root $\xrightarrow{10}$ saw $\xrightarrow{30}$ John

by combing original edges we get root $\xrightarrow{10}$ saw $\xrightarrow{30}$ marry, saw $\xrightarrow{30}$ John

our final dependency graph

Graph-Based MST Learning

G_t : Gold standard parse tree

Training data: $T = \{(x_t, G_t)\}_{t=1}^{|T|}$

1. $w^{(0)} = 0; i = 0$
2. for $n : 1..N$
3. for $t : 1..|T|$
4. Let $G' = argmax_{G'} w^{(i)} . f(G')$
5. if $G' \neq G_t$
6. $w^{(i+1)} = w^{(i)} + f(G_t) - f(G')$
7. $i = i + 1$
8. return w^i

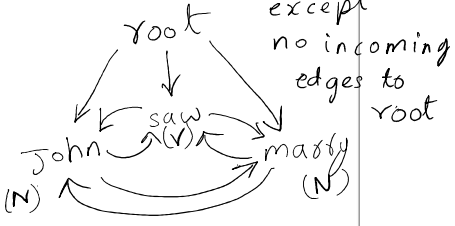
Suppose you are training MST Parser for dependency and the sentence, "John saw Mary" occurs in the training set. Also, for simplicity, assume that there is only one dependency relation, "rel". Thus, for every arc from word w_i to w_j, your features may be simplified to depend only on words w_i and w_j and not on the relation label.

Below is the set of features

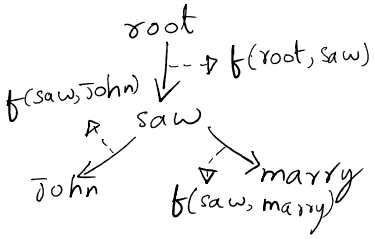
- f_1: pos(w_i) = Noun and pos(w_j) = Noun
- f_2: pos(w_i) = Verb and pos(w_j) = Noun
- f_3: w_i = Root and pos(w_j) = Verb
- f_4: w_i = Root and pos(w_j) = Noun
- f_5: w_i = Root and w_j occurs at the end of sentence
- f_6: w_i occurs before w_j in the sentence
- f_7: pos(w_i) = Noun and pos(w_j) = Verb

The feature weights before the start of the iteration are: {3, 20, 15, 12, 1, 10, 20}. Determine the weights after an iteration over this example.

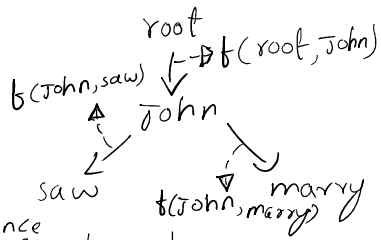
fully connected graph except no incoming edges to root



let G_t Gold standard parse tree



from chu-Liu edmond we got G'



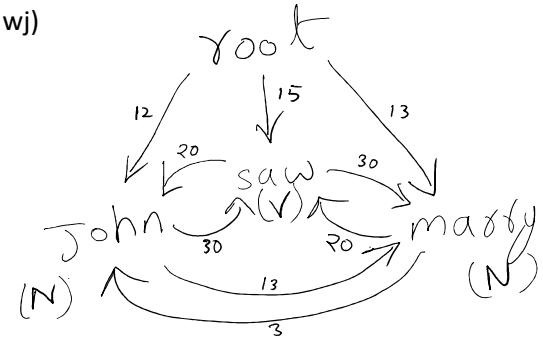
since $G_t \neq G'$ have to update weights

Edge f(w_i,w_j)	f1	f2	f3	f4	f5	f6	f7
f(root, john)	0	0	0	1	0	0	0
f(root, saw)	0	0	1	0	0	0	0
f(root, marry)	0	0	0	1	1	0	0
f(john, saw)	0	0	0	0	0	1	1
f(john, marry)	1	0	0	0	0	1	0
f(saw, john)	0	1	0	0	0	0	0
f(saw, marry)	0	1	0	0	0	1	0
f(marry, john)	1	0	0	0	0	0	0
f(marry, saw)	0	0	0	0	0	0	1

w=[3, 20, 15, 12, 1, 10, 20]

Edge weight = dot product of w & f(wi, wj)

for edge ('root', 'john') weight: 12
for edge ('root', 'saw') weight: 15
for edge ('root', 'marry') weight: 13
for edge ('john', 'saw') weight: 30
for edge ('john', 'marry') weight: 13
for edge ('saw', 'john') weight: 20
for edge ('saw', 'marry') weight: 30
for edge ('marry', 'john') weight: 3
for edge ('marry', 'saw') weight: 20



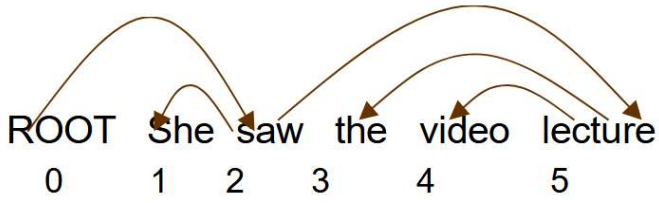
$f(G_t)$ = f(root, saw) + f(saw, John) + f(saw, marry) = [0, 2, 1, 0, 0, 1, 0]

w_new = w + f(Gt) - f(G`)
= [2, 22, 16, 11, 1, 9, 19]

$f(G')$ = f('root','john') + f('john','saw') + f('john','marry') = [1, 0, 0, 1, 0, 2, 1]

Evaluation

- Unlabeled Attachment Score (UAS), which corresponds to the number of correctly predicted dependencies over the number of possibilities;
  - Labeled Attachment Score (LAS), which corresponds to the number of correctly predicted dependencies and relations over the number of possibilities.



$$\text{Acc} = \frac{\text{\# correct deps}}{\text{\# of deps}}$$

UAS(unlabeled attachment score) = 4 / 5
= 80%
LAS (Labeled Attachment Score) = 2 / 5
= 40%

Gold				Parsed			
1	2	She	nsubj	1	2	She	nsubj
2	0	saw	root	2	0	saw	root
3	5	the	det	3	4	the	det
4	5	video	nn	4	5	video	nsubj
5	2	lecture	dobj	5	2	lecture	ccomp

4 UAS

2 LAS

out of no. of UAS

how many matching relations

Word Sense Disambiguation

What's a word sense?

- **A sense (or word sense) is a discrete representation of one aspect of the meaning of a word**
- **Example**
  - We went to see the play Romeo and Juliet at the theater.
  - The children went out to play in the park.

Relationships between word senses

- Homonymy
- Polysemy
- Synonymy
- Antonymy
- Hypernomy
- Hyponomy

Homonymy

Lexemes that share a form
- Phonological, orthographic or both

But have unrelated, distinct meanings
- Examples
  - bat (wooden stick-like thing) vs bat (flying scary mammal thing)
  - bank (financial institution) versus bank (riverside)
- Can be homophones, homographs, or both:
  - Homophones:
    - Write and right
    - Piece and peace

Homonymy causes problems for NLP applications

- Text-to-Speech
  - Bass vs bass
- Information retrieval
  - Bat care
- Machine Translation
  - I went to bank
- Speech recognition
  - Piece and peace

Polysemy

The **bank** is constructed from red brick

I withdrew the money from the **bank**
- Which sense of bank is this?
  - Is it distinct from the river bank sense?
  - How about the savings bank sense?

A single lexeme with multiple related meanings (bank the building, bank the financial institution)

Most non-rare words have multiple meanings
- The number of meanings is related to its frequency
- Verbs tend more to polysemy
- Distinguishing polysemy from homonymy isn't always easy (or necessary)

Another example:

His cottage is near a small **wood**.

The statue was made out of a block of **wood**.

Are those the same sense?

Synonyms

Word that have the same meaning in some or all contexts.
- filbert / hazelnut
- couch / sofa
- big / large
- automobile / car
- vomit / throw up
- Water / H_20

Two lexemes are synonyms if they can be successfully substituted for each other in all situations

But

There are no examples of perfect synonymy
- Why should that be?
- Even if many aspects of meaning are identical
- Still may not preserve the acceptability based on notions of politeness, slang, register, genre, etc.

Example:
- Water and H_20

Word sense disambiguation (contd.)

Synonymy is a relation between senses rather than words

Consider the words *big* and *large*

Are they synonyms?

- How **big** is that plane?
- Would I be flying on a **large** or small plane?

How about here:

- Miss Nelson, for instance, became a kind of **big** sister to Benjamin.
- ?Miss Nelson, for instance, became a kind of **large** sister to Benjamin.

Why?

- *big* has a sense that means being older, or grown up
- *large* lacks this sense

Antonyms

Senses that are opposites with respect to one feature of their meaning

Otherwise, they are very similar!

- dark / light
- short / long
- hot / cold
- up / down
- in / out

More formally: antonyms can

- define a binary opposition or at opposite ends of a scale (*long/short, fast/slow*)
- Be **reverses**: *rise/fall, up/down*

Hyponymy

- One sense is a **hyponym** of another if the first sense is more specific, denoting a subclass of the other
  - *car* is a hyponym of *vehicle*
  - *dog* is a hyponym of *animal*
  - *mango* is a hyponym of *fruit*
- Conversely
  - *vehicle* is a hypernym/superordinate of *car*
  - *animal* is a hypernym of *dog*
  - *fruit* is a hypernym of *mango*

superordinate	vehicle	fruit	furniture	mammal
hyponym	car	mango	chair	dog

Wordnet

- The set of near-synonyms for a WordNet sense is called a **synset** (**synonym set**); it's their version of a sense or a concept

Example: chump as a noun to mean

'a person who is gullible and easy to take advantage of'

{chump[1], fool[2], gull[1], mark[9], patsy[1], fall guy[1], sucker[1], soft touch[1], mug[2]}

- Each of these senses share this same gloss
- Thus for WordNet, the meaning of this sense of chump *is* this list.

WordNet Noun Relations

Relation	Also called	Definition	Example
Hypernym	Superordinate	From concepts to superordinates	*breakfast*[1] → *meal*[1]
Hyponym	Subordinate	From concepts to subtypes	*meal*[1] → *lunch*[1]
Member Meronym	Has-Member	From groups to their members	*faculty*[2] → *professor*[1]
Has-Instance		From concepts to instances of the concept	*composer*[1] → *Bach*[1]
Instance		From instances to their concepts	*Austen*[1] → *author*[1]
Member Holonym	Member-Of	From members to their groups	*copilot*[1] → *crew*[1]
Part Meronym	Has-Part	From wholes to parts	*table*[2] → *leg*[3]
Part Holonym	Part-Of	From parts to wholes	*course*[7] → *meal*[1]
Antonym		Opposites	*leader*[1] → *follower*[1]

WordNet Verb Relations

Relation	Definition	Example
Hypernym	From events to superordinate events	*fly*[9] → *travel*[5]
Troponym	From a verb (event) to a specific manner elaboration of that verb	*walk*[1] → *stroll*[1]
Entails	From verbs (events) to the verbs (events) they entail	*snore*[1] → *sleep*[1]
Antonym	Opposites	*increase*[1] ⟺ *decrease*[1]

WordNet Hierarchies

```
Sense 3
bass, basso --
(an adult male singer with the lowest voice)
=> singer, vocalist, vocalizer, vocaliser
   => musician, instrumentalist, player
      => performer, performing artist
         => entertainer
            => person, individual, someone...
               => organism, being
                  => living thing, animate thing,
                     => whole, unit
                        => object, physical object
                           => physical entity
                              => entity
               => causal agent, cause, causal agency
                  => physical entity
                     => entity
```

```
Sense 7
bass --
(the member with the lowest range of a family of
musical instruments)
=> musical instrument, instrument
   => device
      => instrumentality, instrumentation
         => artifact, artefact
            => whole, unit
               => object, physical object
                  => physical entity
                     => entity
```

Word sense disambiguation (contd.)

Word Sense Disambiguation (WSD)

The task of selecting the correct sense for a word is called word sense disambiguation, or WSD

Given
- a word in context,
- a fixed inventory of potential word sense

Decide which sense of the word this is

Examples
- English-to-Spanish MT
  - Inventory is set of Spanish translations
- Speech Synthesis
  - Inventory is homogrpahs with different pronunciations like *bass* and *bow*

Two variants of WSD task

Lexical sample task
- Small pre-selected set of target words
- And inventory of senses for each word
- Since these words and the set of senses are small, simple supervised classification approaches work very well

All-words task
- In this all-words task, the system is given an all-words entire texts and
- lexicon with an inventory of senses for each entry
- we have to disambiguate every word in the text (or sometimes just every content word).

WSD Methods Supervised Machine Learning Approaches

- Supervised
- Dictionary based
- Semi supervised

Supervised machine learning approach:
- a training corpus of words tagged in context with their sense
- used to train a classifier that can tag words in new text

Summary of what we need:
- the **tag set** ("sense inventory")
- the **training corpus**
- A set of **features** extracted from the training corpus
- A **classifier**

Supervised WSD 1: WSD Tags

What's a tag?
- A dictionary sense?

For example, for WordNet an instance of "bass" in a text has 8 possible tags or labels (bass1 through bass8).

WordNet Bass

The noun ``bass'' has 8 senses in WordNet

1. bass - (the lowest part of the musical range)
2. bass, bass part - (the lowest part in polyphonic music)
3. bass, basso - (an adult male singer with the lowest voice)
4. sea bass, bass - (flesh of lean-fleshed saltwater fish of the family Serranidae)
5. freshwater bass, bass - (any of various North American lean-fleshed freshwater fishes especially of the genus Micropterus)
6. bass, bass voice, basso - (the lowest adult male singing voice)
7. bass - (the member with the lowest range of a family of musical instruments)
8. bass -(nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes)

Word sense disambiguation (contd.)

Supervised WSD 2: Get a corpus

Lexical sample task:

- *Line-hard-serve* corpus - 4000 examples of each
- *Interest* corpus - 2369 sense-tagged examples

All words:

- **Semantic concordance**: a corpus in which each open-class word is labeled with a sense from a specific dictionary/thesaurus.
  - SemCor: 234,000 words from Brown Corpus, manually tagged with WordNet senses
  - SENSEVAL-3 competition corpora - 2081 tagged word tokens

Example of SemCor with wordnet sense numbers

You will find$^{9}_{v}$ that avocado$^{1}_{n}$ is$^{1}_{v}$ unlike$^{1}_{j}$ other$^{1}_{j}$ fruit$^{1}_{n}$ you have ever$^{1}_{r}$ tasted$^{2}_{v}$

Given each noun, verb, adjective, or adverb word in the hand-labeled test set. Ex: fruit$^{1}_{n}$ (the ripened reproductive body of a seed plant), and the other two senses fruit$^{2}_{n}$ (yield;an amount of a product) and fruit$^{3}_{n}$ (the consequence of some effort or action).

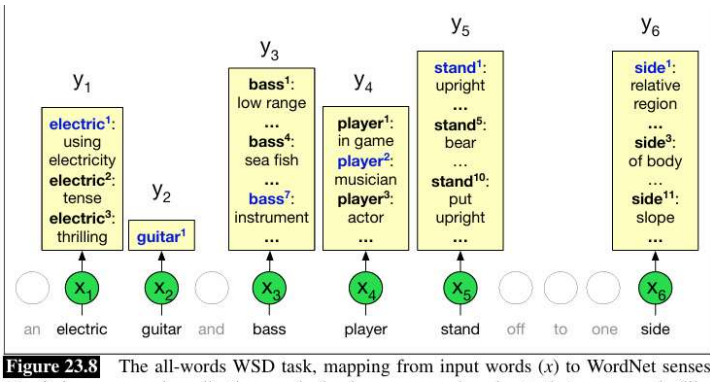


Figure 23.8 The all-words WSD task, mapping from input words (x) to WordNet senses (y). Only nouns, verbs, adjectives, and adverbs are mapped, and note that some words (like *guitar* in the example) only have one sense in WordNet. Figure inspired by Chaplot and Salakhutdinov (2018).

Supervised WSD 3: Extract feature vectors

A simple representation for each observation (each instance of a target word)

- Vectors of sets of feature/value pairs
  - I.e. files of comma-separated values
- These vectors should represent the window of words around the target

Two kinds of features in the vectors

Collocational features and **bag-of-words** features

- **Collocational**
  - Features about words at **specific** positions near target word
    - Often limited to just word identity and POS
- **Bag-of-words**
  - Features about words that occur anywhere in the window (regardless of position)
    - Typically limited to frequency counts

Collocational

Position-specific information about the words in the window

guitar and bass player stand

- [guitar, NN, and, CC, player, NN, stand, VB]
- $Word_{n-2}$, POS_{n-2}, $word_{n-1}$, POS_{n-1}, $Word_{n+1}$ POS_{n+1}…
- In other words, a vector consisting of
- [position n word, position n part-of-speech…]

Example text (WSJ)

An electric guitar and bass player stand off to one side not really part of the scene,

Assume a window of +/- 2 from the target

Bag-of-words

Words that occur within the window, regardless of specific position

First derive a set of terms to place in the vector

Then note how often each of those terms occurs in a given window

Assume we've settled on a possible vocabulary of 12 words that includes guitar and player but not and and stand

[*fishing, big, sound, player, fly, rod, pound, double, runs, playing, guitar, band*]

- The vector for:

  guitar and bass player stand

  [0,0,0,1,0,0,0,0,0,0,1,0]

Supervised WSD4:Classifier

Input:

- a word *w in a text window d (which we'll call a "document")*
- a fixed set of classes $C = \{c_1, c_2, \ldots, c_j\}$
- A training set of *m* hand-labeled text windows again called "documents" $(d_1,c_1),\ldots,(d_m,c_m)$

Output:

- a learned classifier $\gamma:d \rightarrow c$

Any kind of classifier

- Naive Bayes
- Logistic regression
- Neural Networks
- Support-vector machines
- k-Nearest Neighbors

Lesk Algorithm Example

- Consider disambiguating the word bank in the following context:

  The bank can guarantee deposits will eventually cover future tuition costs because it invests in adjustable-rate mortgage securities.

- *Wordnet senses of "bank":*

bank[1]	Gloss:	a financial institution that accepts deposits and channels the money into lending activities
	Examples:	"he cashed a check at the bank", "that bank holds the mortgage on my home"
bank[2]	Gloss:	sloping land (especially the slope beside a body of water)
	Examples:	"they pulled the canoe up on the bank", "he sat on the bank of the river and watched the currents"

- Sense bank1 has two non-stopwords overlapping with the context in deposits and mortgage, while sense bank2 has zero words, so bank1 is chosen.

Word Sense Disambiguation Naïve Bayes Example

Using the Naïve Bayes classifier, we need to disambiguate the word "bank" in the Test Document #1 given in the table below. The Training Documents and their respective classifications are given below: [6 Marks]

	Doc	Sentence	Tokenized Words	Class
Test	1	I went to the bank to deposit my money.	Bank, deposit, money	?
Training	2	The fish swam in the river bank.	Fish, swim, river, bank	R=River_Bank
Training	3	We need to take out a loan from the bank	Loan, bank	F=Financial_Institution
Training	4	The boat was moored at the river bank	Boat, river, bank	R=River_Bank
Training	5	I deposited some money in the bank	deposit, money, bank	F=Financial_Institution

$P(R) = \frac{2}{4} = 0.5$

$P(F) = 2/4 = 0.5$

no. of unique words in vocabulary = 8

no. of words in class R = 5

class F = 5

Test: Bank, deposit, money

$P(Bank|R) = \frac{2+1}{5+8} = \frac{3}{13} = 0.23$

$P(deposit|R) = \frac{0+1}{5+8} = \frac{1}{13} = 0.0769$

$P(money|R) = \frac{0+1}{5+8} = \frac{1}{13} = 0.0769$

$P(Bank|F) = \frac{2+1}{5+8} = \frac{3}{13} = 0.23$

$P(deposit|F) = \frac{1+1}{5+8} = \frac{2}{13} = 0.1538$

$P(money|F) = \frac{1+1}{5+8} = \frac{2}{13} = 0.1538$

$P(Test|R) \propto P(R) \times P(Bank|R) \times P(deposit|R) \times P(money|R)$

$= 0.5 \times 0.23 \times (0.0769)^2 = 0.00068$

$P(Test|F) \propto P(F) \times P(Bank|F) \times P(deposit|F) \times P(money|F)$

$= 0.5 \times 0.23 \times (0.1538)^2 = 0.0027$

as $P(Test|F) > P(Test|R)$ the test doc belongs to class financial_institution (F)

Self attention Example

3 input vectors
[1, 0, 1, 0]
[0, 2, 0, 2]
[1, 1, 1, 1]

$q_i = Q x_i$ (queries)
$k_i = K x_i$ (keys)
$v_i = V x_i$ (values)

Weights for Key
[[0, 0, 1],
[1, 1, 0],
[0, 1, 0],
[1, 1, 0]]

Key representation for Input

$$[1, 0, 1, 0] \cdot \begin{bmatrix} 0, 0, 1 \\ 1, 1, 0 \\ 0, 1, 0 \\ 1, 1, 0 \end{bmatrix} = [0, 1, 1]$$

Key for all inputs
[0 1 1]
[4 4 0]
[2 3 1]

Weights for query
[[1, 0, 1],
[1, 0, 0],
[0, 0, 1],
[0, 1, 1]]

Query representation for Input

$$[1, 0, 1, 0] \cdot \begin{bmatrix} 1, 0, 1 \\ 1, 0, 0 \\ 0, 0, 1 \\ 0, 1, 1 \end{bmatrix} = [1, 0, 2]$$

Query for all inputs
[1, 0, 2]
[2, 2, 2]
[2, 1, 3]

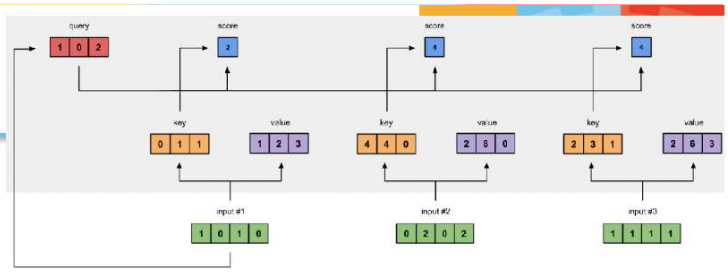
Weights for Value
[[0, 2, 0],
[0, 3, 0],
[1, 0, 3],
[1, 1, 0]]

Value representation for Input

$$[1, 0, 1, 0] \cdot \begin{bmatrix} 0, 2, 0 \\ 0, 3, 0 \\ 1, 0, 3 \\ 1, 1, 0 \end{bmatrix} = [1, 2, 3]$$

Value for all inputs
[1, 2, 3]
[2, 8, 0]
[2, 6, 3]

CONTD...



$$e_{ij} = q_i^T k_j \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

3 attention scores for first input *i* : (e_{ij})

$$[1, 0, 2] \cdot \begin{bmatrix} 0, 1, 1 \\ 4, 4, 0 \\ 2, 3, 1 \end{bmatrix} = [2, 4, 4]$$

3 softmax attention scores (α_{ij})
softmax([2, 4, 4]) = [0.0, 0.5, 0.5]
weighted values
1: 0.0 * [1, 2, 3] = [0.0, 0.0, 0.0]
2: 0.5 * [2, 8, 0] = [1.0, 4.0, 0.0]
3: 0.5 * [2, 6, 3] = [1.0, 3.0, 1.5]

Output 1, which is based on the query representation from Input 1 interacting with all other keys, including itself

$$o_i = \sum_j \alpha_{ij} v_i$$

[0.0, 0.0, 0.0]
+ [1.0, 4.0, 0.0]
+ [1.0, 3.0, 1.5]

= [2.0, 7.0, 1.5]

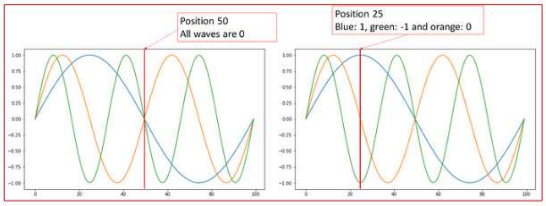
All the outputs after first iteration
[2.0, 7.0, 1.5], # Output 1
[2.0000, 8.0, 0.0], # Output 2
[2.0, 7.8, 0.3] # Output 3

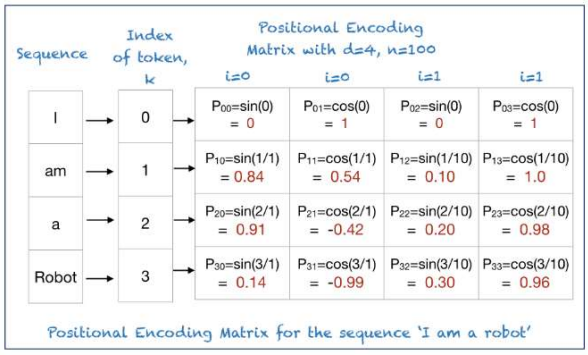
Position representation vectors through sinusoids

Sinusoidal position representations: concatenate sinusoidal functions of varying periods

$$p_i = \begin{bmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{bmatrix}$$

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$





WordPiece tokenization algorithm

WordPiece is the tokenization algorithm Google developed to pretrain BERT.

Starts from a small vocabulary : special tokens used by the model and the initial alphabet.

Identifies subwords by adding a prefix (like ## for BERT),

"word" gets split like this: w ##o ##r ##d

Initial vocabulary contains all the characters present at the beginning of a word and the characters present inside a word preceded by the WordPiece prefix.

WordPiece tokenization algorithm

Instead of selecting the most frequent pair (BPE), WordPiece computes a score for each pair

score=(freq_of_pair)/(freq_of_first_element × freq_of_second_element)

Algorithm prioritizes the merging of pairs where the individual parts are less frequent in the vocabulary.

- It won't necessarily merge ("un", "##able") even if that pair occurs very frequently in the vocabulary, because the two pairs "un" and "##able" will likely each appear in a lot of other words and have a high frequency.
- ("hu", "##gging") will probably be merged faster (assuming the word "hugging" appears often in the vocabulary) since "hu" and "##gging" are likely to be less frequent individually.

Tokenization example using wordpiece

Corpus uses these five words: **"hug", "pug", "pun", "bun", "hugs"**

Assume the words had the following frequencies: **("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)**

Training: token **learner** : splitting words and adding a prefix ## for subwords

- ("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12),
- ("b" "##u" "##n", 4), ("h" "##u" "##g" "##s", 5)

First iteration:

initial vocabulary : ["b", "h", "p", "##g", "##n", "##s", "##u"] (ignoring special tokens)

- Score ("##u", "##g") : pair freq/individual freq = 20/36*20=1/36
- Score ("##g", "##s"): pair freq/individual freq = 5/20*5=1/20
- Likewise compute scores for all the pairs
- first merge learned is ("##g", "##s") -> **("##gs")**

Vocabulary after first iteration: ["b", "h", "p", "##g", "##n", "##s", "##u", **"##gs"**]

Repeat iterations till you get a desired vocab size.

Token **segmenter:** New inputs are tokenized by applying the following steps:

- Pre-tokenize it
- Split it
- Apply the tokenization algorithm on each word.
  - Look for the biggest subword starting at the beginning of the first word and split it
  - Repeat the process on the second part
  - And so on for the rest of that word and the following words in the text

Continuing the example: suppose the final vocab learnt:

- Vocabulary: **["b", "h", "p", "##g", "##n", "##s", "##u", "##gs", "hu", "hug"]**
- Corpus: ("hug", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("hu" "##gs", 5)
- **Test word : "hugs"**
- **Tokenization of "hugs" is ["hug", "##s"]**

Evaluating Summaries: ROUGE

ROUGE (Recall Oriented Understudy for Gisting Evaluation)

- Intrinsic metric for atomically evaluating summaries
- Based on BLEU (a metric used for machine translation)
- Not as good as human evaluation ("Did this answer the user's question?")
- But much more convenient

ROUGE-2

Given a document D, and an automatic summary X:

1. Have N humans produce a set of reference summaries of D
2. Run system, giving automatic summary X
3. What percentage of the bigrams from the reference summaries appear in X?

$$ROUGE-2 = \frac{\sum_{s \in \{RefSummaries\}} \sum_{\text{bigrams } i \in S} \min(count(i,X), count(i,S))}{\sum_{s \in \{RefSummaries\}} \sum_{\text{bigrams } i \in S} count(i,S)}$$

ROUGE-2 Example

Q: "What is water spinach?"

Human 1: Water spinach is a green leafy vegetable grown in the tropics.

Human 2: Water spinach is a semi-aquatic tropical plant grown as a vegetable.

Human 3: Water spinach is a commonly eaten leaf vegetable of Asia.

- System answer: Water spinach is a leaf vegetable commonly eaten in tropical areas of Asia.

Rouge-2 score= $\frac{3+3+6}{10+9+9}$ = 12/28 = .43

Log-Likelihood Ratio (LLR)

Centroid Based Summarization

innovate achieve lead

Water spinach (ipomoea aquatica) is a semiaquatic leafy green plant with long hollow stems and spear or heart shaped leaves. It is widely grown throughout Asia as a leaf vegetable. The leaves and stems are often eaten fried flavored with salt or in soups. Other common names include morning glory vegetable, kangkong (Malay). It is not related to spinach, but is closely related to sweet potato and convolvulus.

1. Tokenize the input sentence Sj into set of non-stop words : Wi's
2. Compute LLR for every Wi's
3. Find the weights of every Wi's
4. Using weight(Wi) compute weight(S) for every Sj's
5. Rank the Sj's in descending order of weight or compare with threshold θ
6. Pick top "k" sentences for summary

S1 : Water spinach ipomoea aquatica semiaquatic leafy green plant long hollow stems spear heart shaped leaves = 8/15 = 0.53
S2 : grown Asia leaf vegetable = 2/4 = 0.5
S3 : leaves stems eaten fried flavored salt soups = 2/7 = 0.28
S4 : common names morning glory vegetable kangkong Malay = 3/7 = 0.42
S5 : related spinach closely related sweet potato convolvulus = 4/7 = 0.57

λ(*w*) = log likelihood ratio for a word is given by

$$\lambda = \frac{b(k, N, p)}{b(k_I, N_I, p_I).b(k_B, N_B, p_B)}$$

Assume Pi(W | I) = 1/24 , Pb(W | B) = 50/10000000
P = (1+50)/(24+10000000)
L(W) = 0.000877 → -2log(L(w)) = 14.8

$$weight(w_i) = \begin{cases} 1 & \text{if } -2\log\lambda(w_i) > 10 \\ 1 & \text{if } w_i \in question \\ 0 & \text{otherwise} \end{cases}$$

Weight(vegetable) = 1
Weight(S2) = ¼ * sum_of_weights{grown, asia, leaf, vegetable)

$$weight(s) = \frac{1}{|S|}\sum_{w \in S} weight(w) \qquad weight(s_i) = \sum_{w \in s_i} \frac{weight(w)}{|\{w | w \in s_i\}|}$$

Water spinach (ipomoea aquatica) is a semiaquatic leafy green plant with long hollow stems and spear or heart shaped leaves. It is not related to spinach, but is closely related to sweet potato and convolvulus.