Finite state Transducers (FSTs)

- FSA
  Accept/Reject | spell checking
- Mapping b/w upper language and lower language
- If successful
  → returns the upper side symbols on the path as result.
- If unsuccessful
  → returns nothing (reject)

- FSA have some input labels.
- FST have **input : output** pairs on labels.

Q → Finite set of states

Σ

Δ → output alphabet

$q_0 \in Q$

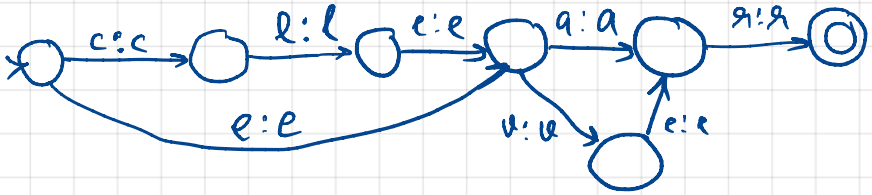
$F \subseteq Q$

$\delta(q, w) : Q \times \Sigma^* \rightarrow 2^Q$

$\delta(q, w) : Q \times \Sigma^* \rightarrow 2^{\Delta^*}$

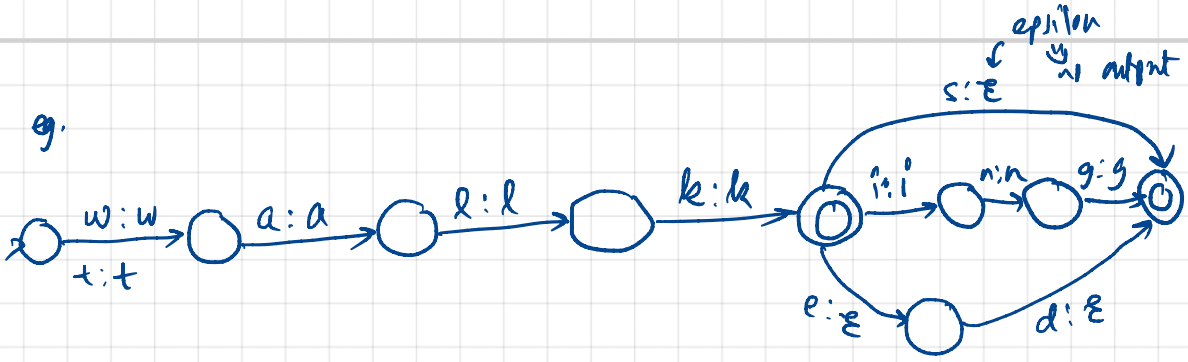
surface form ← input → cats
lexical form ← output → cat + N + P

eg.



input → clear, output → clear
input → ever, output → ever

epsilon = no output



e.g.

can map

talk, talks, talked, talking to talk

and vice versa.

Operations on FST:

1. Inversion

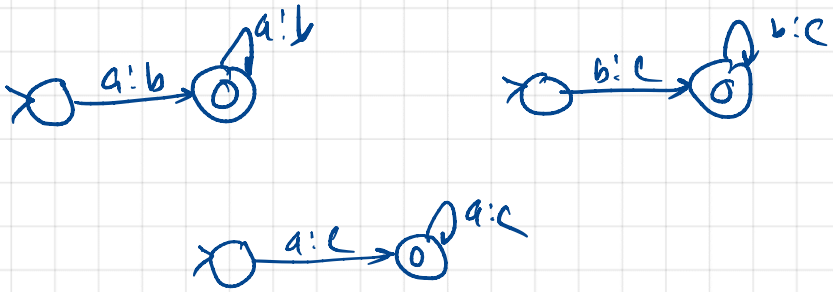
$$T \rightarrow T^{-1}$$

$T = \{(q, q_1), (q, q_2), (b, b_1), (c, a)\}$

$T^{-1} = \{(q_1, q), (q_2, q), (b_1, b), (a, c)\}$

The inversion of transducer $T \rightarrow T^{-1}$ simply switches the input and output labels

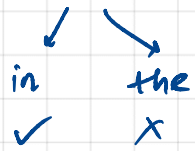
2. Composition



$$T_1 \circ T_2(x) = T_1(T_2(x))$$

Probabilistic Language Modeling

Please turn your homework ____?



N-gram model

↳ predicts next word from previous N-1 words

An N-gram is a N-token sequence of words.

Bigram is a sequence of 2 words.
("please turn", "turn your", "your homework")

Smoothing
- Add one
- Witten Bell
- Good Turing
- Interpolation
- Back off

Unsmoothing??

Witten Bell — shared probability mass

↳ unseen → $\frac{T(c)}{N(c)+T(c)}$

seen → $\frac{N(c)}{N(c)+T(c)}$

} Bigrams starting with c, for example in slide

→ To get counts from probabilities:

$P(w_2/w_1) = \frac{c(w_2/w_1)}{N(w_1)}$ ← needed

↓ known from Witten-Bell

$c(w_2/w_1) = P(w_2/w_1) * N(w_1)$

$= \frac{1}{Z(w_1)} \cdot \frac{T(w_1)}{N(w_1)+T(w_2)} \cdot N(w_1)$

↑ counts of unseen types

$= \frac{T(w_1)}{Z(w_1)} \cdot \frac{N(w_1)}{N(w_1)+T(w_1)}$

Good Turing Discounting

N-grams which occur once in the corpus

use the frequency of singletons to re-estimate the frequency of zero-count bigrams, trigrams, ...

N_c = Number of N-grams that occur c times
("frequency of frequency")

N_0 ⇒ Number of bigrams with count 0.
N_1 ⇒ Number of bigrams with count 1.

N_c can be considered as bin → stores N-grams occurring with frequency c.

$$N_c = \sum_{x.count(x)=c} 1$$

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$ } formula to re-estimate counts

$$P^*_{GT}(\text{things with zero freq in training}) = \frac{N_1}{N}$$ } Good Turing probability

N_1 → number of items in bin 1
N → total no. of items in training set

Eg. Fishing in a lake with 8 species

bass - 0
carp - 16
catfish - 0
eel - 1
perch - 3
salmon - 1
trout - 1
whitefish - 2
total - 18

unseen species - bass, catfish

probability that next fish caught is bass or catfish

$= P^*_{GT}(\text{bass, catfish}) = \frac{N_1}{N} = \frac{3}{18}$

$N_0 = 2$
$N_1 = 3$
$N_2 = 1$
$N_3 = 1$
$N_4 = 0$
$N_5 = 0$
$N_6 = 0$
$N_7 = 0$
$N_8 = 0$
$N_9 = 0$
$N_{10} = 1$

	unseen (Bass or Catfish)	Trout
c	0	1
MLE	$P = \frac{0}{18}$	$\frac{1}{18}$
c^*		$(1+1) * \frac{N_2}{N_1} = \frac{2}{3}$
p^*_{GT}	$\frac{3}{18}$ ↳ divided among all unseen events	$\frac{(2/3)}{18}$

Revised count c^* for trout was discounted from $c = 1$ to $c^* = 0.67$

Interpolation (Linear)

$P(w_n | w_{n-2}\, w_{n-1})$

If there is no example of the trigram $w_n\, w_{n-1}\, w_{n-2}$ in the corpus, we use bigram to estimate its probability.

↓

$P(w_n | w_{n-1})$

If bigram is not present, then do unigram.

$P(w_n)$

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

$$\boxed{\sum_i \lambda_i = 1}$$

The λ values need to be learned from a defined corpus.

Back off N-gram Modelling

→ Katz Backoff

If N gram we need has 0 count, we approximate it by backing off to N-1 gram, recursively.

discounted probability

$$P_{katz}\left(w_n \mid w_{n-N+1}^{n-1}\right) = \begin{cases} P^*\left(w_n \mid w_{n-N+1}^{n-1}\right) & \text{if } c\left(w_{n-N+1}^{n}\right) > 0 \\ \alpha\left(w_{n-N+1}^{n-1}\right) \cdot P_{katz}\left(w_n \mid w_{n-N+2}^{n-1}\right) & \text{otherwise} \end{cases}$$

For trigram x, y, z

$$P_{katz}(z \mid x, y) = \begin{cases} P^*(z \mid x, y) & \text{if } c(x,y,z) > 0 \\ \alpha(x,y)\, P_{katz}(z \mid y) & \text{else if } c(x,y) > 0 \\ P^*(z) & \text{otherwise} \end{cases}$$

For bigram y, z

$$P_{katz}(z \mid y) = \begin{cases} P^*(z \mid y) & \text{if } c(y,z) > 0 \\ \alpha(y)\, P^*(z) & \text{otherwise} \end{cases}$$

Discounted probability

$$P^*\left(w_n \mid w_{n-N+1}^{n-1}\right) = \frac{c^*\left(w_{n-N+1}^{n}\right)}{c\left(w_{n-N+1}^{n-1}\right)}$$

original MLE $= \dfrac{c\left(w_{n-N+1}^{n}\right)}{c\left(w_{n-N+1}^{n-1}\right)}$

P^* is slightly smaller than the original MLE.
This leftover probability mass is distributed to lower order n-grams using the parameter α.

Left over probability mass → sum of all patterns for which count is non-zero.

$$\beta\left(w_{n-N+1}^{n-1}\right) = 1 - \sum_{w_n : c(w_{n-N+1}^{n}) > 0} p^*\left(w_n \mid w_{n-N+1}^{n-1}\right)$$

Distribute β among patterns with zero counts

$$\alpha\left(w_{n-N+1}^{n-1}\right) = \frac{\beta\left(w_{n-N+1}^{n-1}\right)}{\sum_{w_n : c(w_{n-N+1}^{n}) = 0} P_{katz}\left(w_n \mid w_{n-N+2}^{n-1}\right)}$$

$$= \frac{1 - \sum_{w_n : c(w_{n-N+1}^{n}) > 0} p^*\left(w_n \mid w_{n-N+1}^{n-1}\right)}{1 - \sum_{w_n : c(w_{n-N+1}^{n}) > 0} p^*\left(w_n \mid w_{n-N+2}^{n-1}\right)}$$

Evaluating N-Gram models

(a) Extrinsic evaluation

- embed it in an application and measure performance of the application.

(b) Intrinsic evaluation

- measure quality of model irrespective of application

eg. perplexity (PP)

For a test set $w = w_1 w_2 \dots w_n$

$$PP(w) = P(w_1 w_2 \dots w_n)^{-1/N}$$

↳ probability (calculated using any language model)

POS tagging

Problem: Determining best tag sequence for a sentence.

$T = t_1 t_2 t_3 \dots t_n$ (tags)
$W = w_1 w_2 w_3 \dots w_n$ (sentence)

Input: The Lead paint is unsafe
output: The/Det Lead/N Paint/N is/V unsafe/adj

Approach: 1) Rule based
~~2)~~ 2) Hidden Markov Model based tagger

HMM

↳ Components of the Model:

$Q = \{q_1, q_2, q_3, \dots, q_N\}$ → set of N states

$A = \{a_{i1}, a_{i2}, a_{i3}, \dots a_{ij} \dots, a_{NN}\}$ → Transition probability matrix
↳ represents probability of moving from state i to state j.
$\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$

$O = \{O_1, O_2, \dots, O_T\}$ → sequence of T observations each drawn from a vocabulary $V = \{v_1, v_2, \dots, v_k\}$

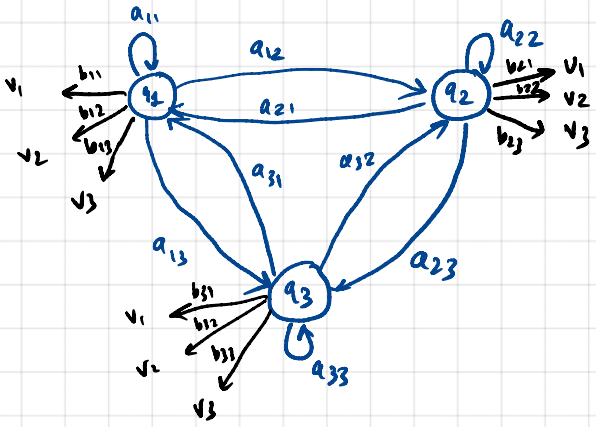
$B = b_i(O_t)$ → Emission probabilities
↳ probability of observation O_t being generated from state i.

$\Pi = \{\pi_1, \pi_2, \dots \pi_N\}$ → Initial probability distribution
π_i → probability that Markov chain starts from state q_i

$\sum_{i=1}^{N} \pi_i = 1$

q_1, q_2, q_3: POS tags

v_1, v_2, v_3: input sentences

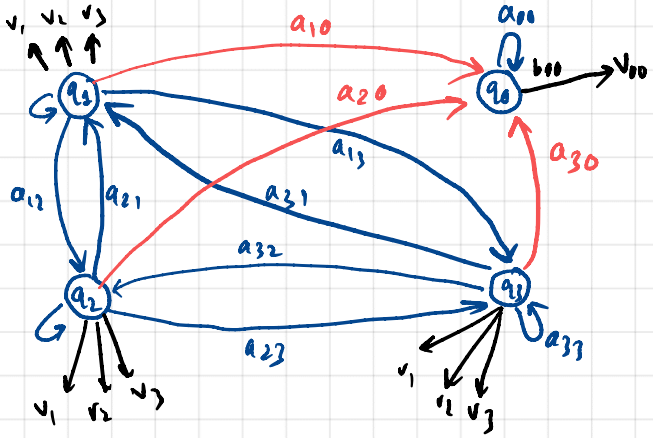


$q_i(t-1) \xrightarrow{a_{ij}} q_j(t)$

$\sum_j a_{ij} = 1 \quad \forall i$ → sum of all transition probabilities of a state is 1.

$\sum_k b_{jk} = 1 \quad \forall j$ → sum of all emission probabilities for all hidden states is 1.

Once HMM reaches accepting state, it emits v_0.



$a_{01} = a_{02} = a_{03} = 0$

$b_{00} = 1$

$\pi_0 = 0$

→ Central Issues / Basic Problems

(a) Evaluation Problem

(b) Decoding Problem

(c) Learning Problem

Evaluation Problem → forward Algorithm

Given a HMM θ and observation sequence V^T, determine the likelihood

$$P(V^T \mid \theta)$$

↳ probability of emitting V^T from machine θ.

Known quantities: $\mathcal{S}, \mathcal{O}, a_{ij}, b_{ik}$

Decoding Problem → Viterbi Algorithm

Given an observation sequence and HMM θ, find the best hidden state sequences $\mathcal{S}$.

once we know this, we can find the tags.

Learning Problem

Estimate a_{ij}, b_{ik} from a large annotated training set

Evaluation Problem

Given: Q, O, a_{ij}, b_{ik}

$$P(V^T | \theta) = \sum_{r=1}^{r_{max}} P(V^T | Q_r^T) * P(Q_r^T) \rightarrow (1)$$

$Q_{(r)}^T \rightarrow$ denotes 1 sequence from Q_T

$r_{max} \rightarrow$ no. of all possible sequences

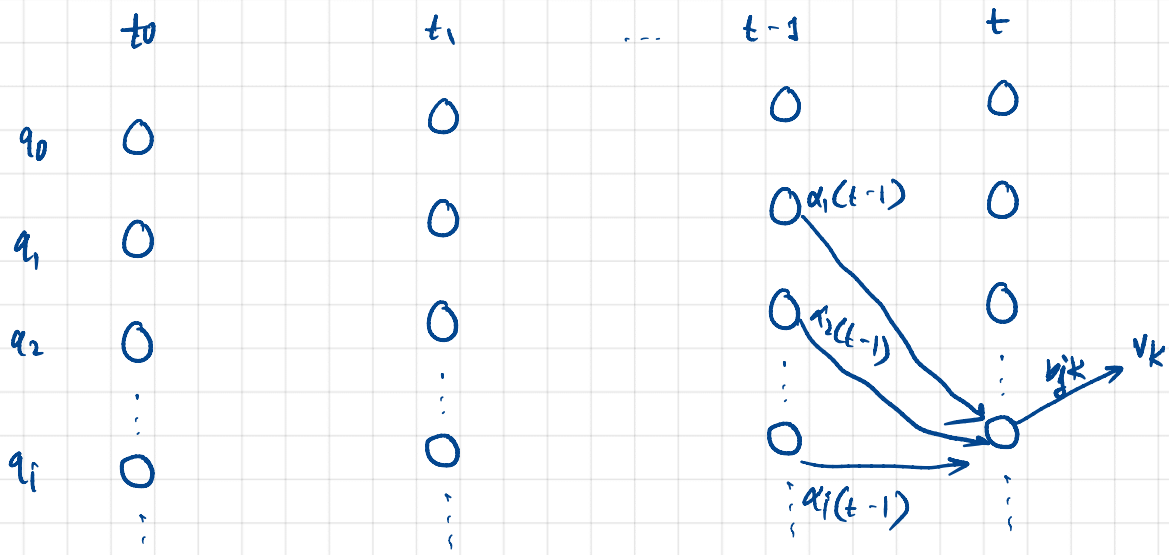
$$Q_r^T = \{ q(1), q(2), \ldots, q(T) \}$$

T can be thought of as time-stamps.

$$P(Q_r^T) = \prod_{t=1}^{T} P(q(t) | q(t-1)) \rightarrow (2)$$

$$P(V^T | Q_r^T) = \prod_{t=1}^{T} P(v(t) | q(t)) \rightarrow (3)$$

$$\therefore \boxed{P(V^T | \theta) = \sum_{r=1}^{r_{max}} \prod_{t=1}^{T} P(v(t) | q(t)) \cdot P(q(t) | q(t-1))}$$



q_0 is accepting state.

At time t, the probability of the previous state being q_i is $\alpha_i(t-1)$

Note that $\alpha_i(t-1)$ is not the same as a_{ij} which is the transition probability.

$$\alpha_j(t) = \begin{cases} 0 & t=0,\ q_j \neq \text{initial state} \\ 1 & t=0,\ q_j = \text{initial state} \\ \left[\sum_{i}^{N} \alpha_i(t-1)\, a_{ij}\right] b_{jk}\, v(t) & \text{otherwise} \end{cases}$$

probability that machine will be in state q_j after generating t symbols

Forward algorithm:

1. Initialization
   $t \leftarrow 0$, a_{ij}, b_{jk}, v^T, $\alpha_j(0)$
2. for $t \leftarrow t+1$
   $$\alpha_j(t) = b_{jk}\, v(t) \sum_{i=1}^{N} \alpha_i(t-1) \cdot a_{ij}$$
3. until $t = T$
4. Return $P(v^T|\theta) \leftarrow \alpha_0(T)$

Probability that it reaches final state q_0 and emits v_0 at time T.

Consider an example.

V^5 → sequence of five symbols = $\{v_3, v_3, v_2, v_1, v_0\}$

$\alpha_3(3)$ → what is the probability that HMM will be in state q_3 after generating first 3 symbols $\langle v_3, v_1, v_2 \rangle$

$\alpha_2(4)$ → in state 2 after generating 4 symbols $\langle v_3, v_1, v_2, v_1 \rangle$

Eg. Hidden state : q_1, q_2, q_3 and final state (q_0)

Visible state : $(v_0), v_1, v_2, v_3, v_4$

$a_{ij} =$

	q_0	q_1	q_2	q_3	
	1	0	0	0	q_0
	0.2	0.3	0.1	0.4	q_1
	0.2	0.5	0.2	0.1	q_2
	0.7	0.1	0.1	0.1	q_3

$\sum_j a_{ij} = 1 \quad \forall i$

$b_{jk} =$ (k → , j ↓)

	v_0	v_1	v_2	v_3	v_4	
	1	0	0	0	0	q_0
	0	0.3	0.4	0.1	0.2	q_1
	0	0.1	0.1	0.7	0.1	q_2
	0	0.5	0.2	0.1	0.2	q_3

$\sum_k b_{jk} = 1 \quad \forall j$

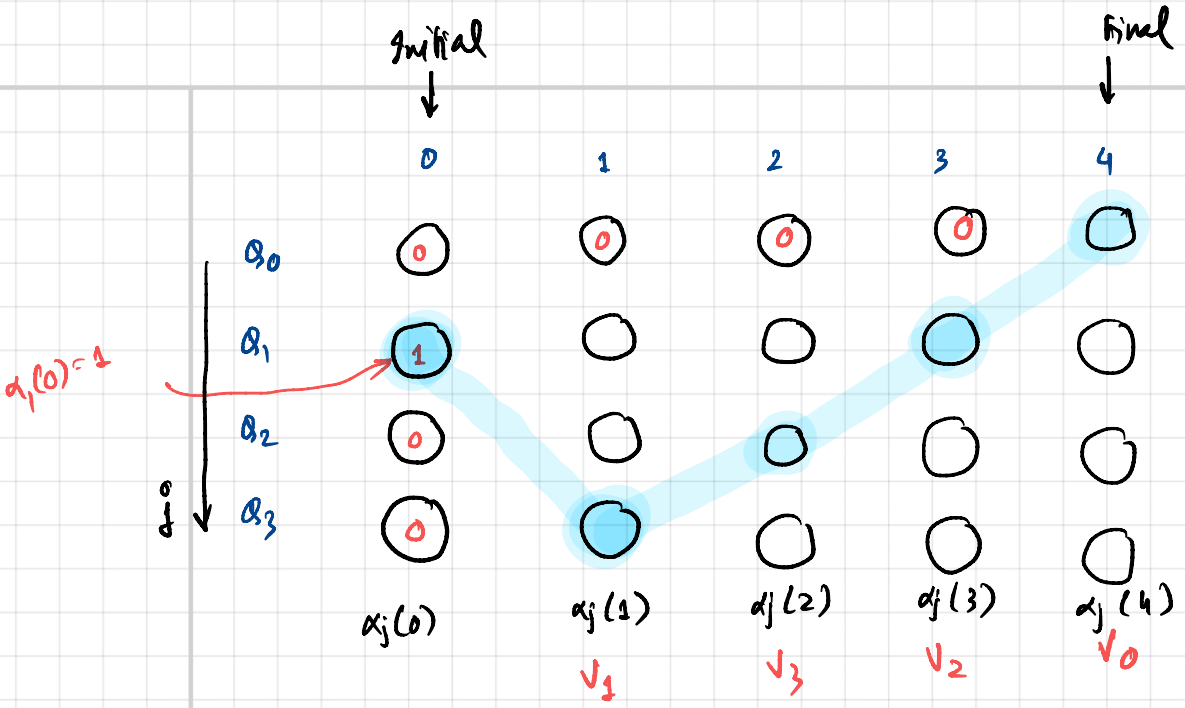
$V^4 = \{v_1, v_3, v_2, v_0\}$

what is $P(V^4 | 0)$?

Assume, at t_0, initial state q_1

⇓

$\alpha_1(0) = 1$



probability of going from q_1 to q_1 — probability of emitting v_3 from q_1

$$\alpha_1(1) = [\alpha_1(0) * a_{11} * b_{11}] + [\alpha_2(0) * a_{21} * b_{21}] + [\alpha_3(1) * a_{31} * b_{11}]$$

$$= 0.09 + 0 + 0 = 0.09$$

$$\alpha_2(1) = [1 \times 0.1 \times 0.1] + 0 + 0 = 0.01$$

$$\alpha_3(1) = 0.2$$

$$\alpha_1(2) = 0.0052$$
$$\alpha_2(2) = 0.0343$$
$$\alpha_3(2) = 0.0057$$

$$\alpha_1(3) = 0.0019$$
$$\alpha_2(3) = 0.0008$$
$$\alpha_3(3) = 0.0012$$

$$\alpha_1(4) = 0$$
$$\alpha_2(4) = 0$$
$$\alpha_3(4) = 0$$

$$\alpha_0(4) = 0.00138$$

Path → $q_1 \to q_3 \to q_2 \to q_1 \to q_0$

[by finding max probability in each time step]

↑ decoding problem

$$P(v_4|0) = 0.00138$$

↖ evaluation problem

At time $t=0$,

most probable state q_1

At time $t=1$

most probable state q_3

$t=2 \rightarrow q_2$

$t=3 \rightarrow q_1$

$t=4 \rightarrow q_0$

This can be considered as sentence

↳ Output sequence: v_1, v_3, v_2, v_0

↱ Hidden state sequence: q_1, q_3, q_2, q_1, q_0

This can be considered as POS tags

→ **Algorithm for Decoding:**

Given v^T, find most probable Hidden state sequences.

1. Initialization: $Path = \{0\}$, $t \leftarrow 0$, $j \leftarrow 0$, a_{ij}, b_{jk}

2. for $t \leftarrow t+1$

   for $j \leftarrow j+1$

   $$\alpha_j(t) \leftarrow b_{jk=v(t)} \sum_{i=1}^{N} \alpha_i(t-1) \cdot a_{ij}$$

   until $j = N$

   $j' \leftarrow \arg\max_j \alpha_j(t)$

   Append $q_{j'}$ to Path

   until $t = T$

3. Return Path

Assumption:
All states are interconnected and reachable except q_0 (with each other)

Because otherwise when we choose max, that path may not exist.

HMM Learning Problem

Forward-Backward algorithm
(Baumen Welch Algorithm)

→ Evaluation Problem

$$P(v^T|\theta) = \sum_{n=1}^{r_{max}} P(v^T|q_n^T) * P(q_n^T)$$

Estimate a_{ij} and b_{jk}

→ Forward Algorithm

$$\alpha_j = \begin{cases} 0 & t=0 \ \& \ q_j \neq \text{initial state} \\ 1 & t=0 \ \& \ q_j = \text{initial state} \\ [\Sigma \alpha_i(t-1).a_{ij}]\, b_{jk}\, v(t) & \text{otherwise} \end{cases}$$

→ Backward Algorithm

$$B_i(t) = \begin{cases} 0 & q_j(t) \neq q_0 \text{ at } t=T \\ 1 & q_i(t) = q_0 \text{ at } t=T \\ [\Sigma \beta_j(t+1)\, a_{ij}].\, b_{jk}\, v(t+1) & \text{otherwise} \end{cases}$$

Define $\gamma_{ij}(t)$
↳ Probability of transition from $q_i(t-1)$ to $q_j(t)$

$$\gamma_{ij}(t) = \frac{\alpha_i(t-1)\, a_{ij}\, b_{jk}\, \beta_j(t)}{P(v^T|\theta)}$$

$$\hat{a}_{ij} = \frac{\text{Expected no. of transitions from state } i \text{ to } j}{\text{expected no. of transitions from state } i}$$

Expected no. of transitions from $q_i(t-1)$ to $q_j(t) = \sum_{t=1}^{T} \gamma_{ij}(t)$

Expected no. of transitions from q_i to any state $= \sum_{t=1}^{T} \sum_{k} \gamma_{ik}(t)$

∴ Refined transition probability

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T} \gamma_{ij}(t)}{\sum_{t=1}^{T} \sum_{k} \gamma_{ik}(t)}$$

$\hat{b}_{jk}$ = Expected no. of times in state j and observing the symbol v_k / Expected no. of times in state j

$$\hat{b}_{jk} = \frac{\sum_{t=1,\, k}^{T} \sum_{v(t)=v(k)} \sum_{l} \gamma_{jl}(t)}{\sum_{t=1}^{T} \sum_{l} \gamma_{jl}(t)}$$

↳ Refined emission probability

Forward-Backward algorithm

1. Initialize A & B.
2. Iterate until convergence
   - Define $\gamma_{ij}(t)$
   - Compute $\hat{a}_{ij}$
   - Compute $\hat{b}_{jk}$
3. Return A & B

Syntactic Analysis

(a) Top-down Parsers

(b) Bottom-up Parser

(c) Dynamic Programming Parsing algorithm (Early, CYK)

(d) Probabilistic Parsing

Context-Free Grammar → $G = (V, \Sigma, P, S)$

For CYK, CFG has to be converted to its chomsky Normal form.

eg. $A \rightarrow BCD$ | $A \rightarrow BX$, $X \rightarrow CD$

Some rules for English:

- I prefer a morning flight.
  Subject Noun Verb

- Show the latest fare.
  verb

- Yes/No questions
- "wh.." questions

S → NP VP
S → X1 VP
X1 → Aux NP

S → book | include | prefer
S → Verb NP
S → X2 PP
S → Verb PP

VP → book | include | prefer
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP

NP → I | she | me
NP → Houston
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun

Det → the | a

Noun → book | flight | meal | money
Verb → book | include | prefer
proper noun → Houston
Aux → Does
Preposition → from | to | on | through

Using the rules above, something like
"Book the flight through Houston" → sentence length = n = 5
is derivable with multiple parse trees.

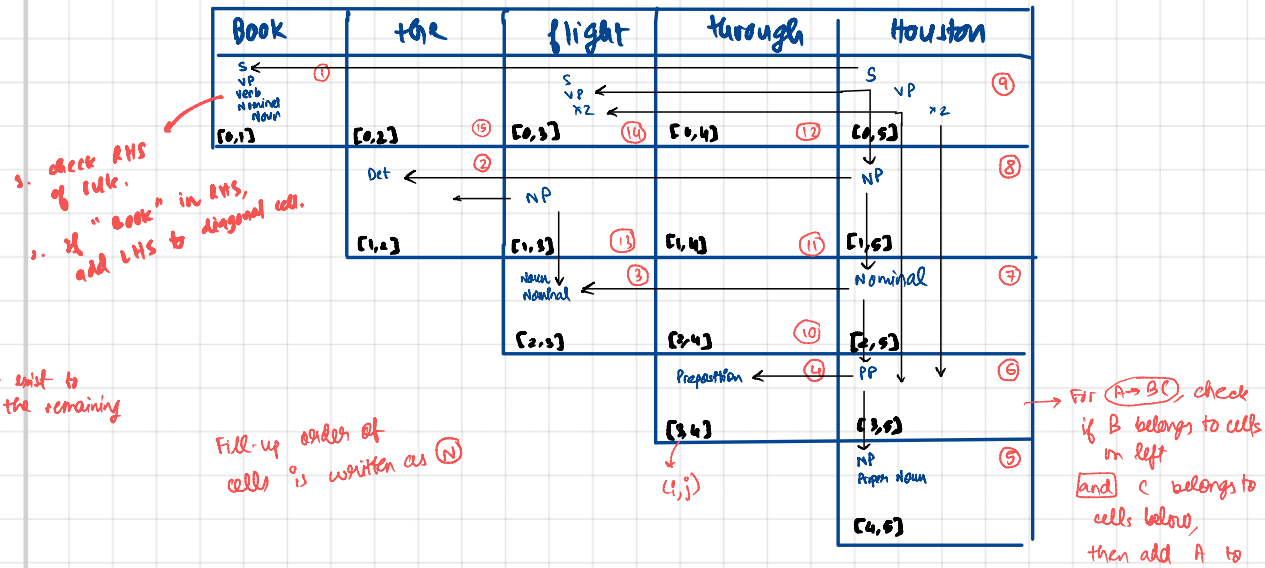
We develop the CKY algorithm below where we fill the upper triangular portion of a (n+1) (n+1) matrix using dynamic programming, where n = sentence length

CKY - Cocke Kasami Younger Algorithm

```
function CKY-Parse (words, grammar)

    for  j ← from 1 to length(words) do                    // diagonal
        for all rules {A | A → words[j] ∈ Grammar}
            table[j-1, j] ← table[j-1, j] ∪ A

    for  j ← length(words) down to 0  do
        for i ← from j-2 down to 0 do
            from  k ← i+1  to  j-1  do
                for  all { A | A → BC ∈ grammar and
                            B ∈ table[i, k]  and
                            C ∈ table[k, j] }

                    table[i,j] ← table[i,j] ∪ A

    return table
```

from this table, multiple parse trees are possible if grammar is ambiguous. →

eg. sentence : Book the flight through houston → len(words) = 5
grammar : defined in previous page.



Probabilistic CFG Parsing (PCFG)

$G = (V, \Sigma, P, S)$

V = set of non-terminals
Σ = set of terminals
P = set of production rules

$A \rightarrow \beta [p]$

$\beta \in \{ \Sigma \cup V \}^*$

S = start symbol

$P(A \rightarrow \beta)$ or $P(A \rightarrow \beta | A)$: p is probability of β being derivable from A.

$\sum_{\beta} P(A \rightarrow \beta) = 1$

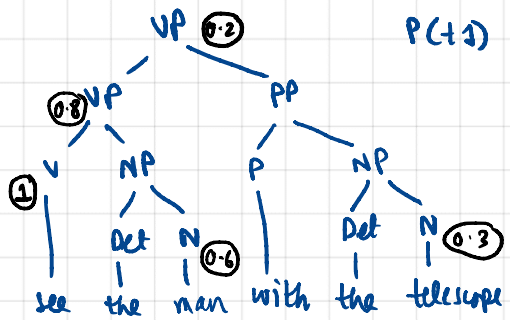
Example PCFG:

VP → V NP	[0.8]
VP → VP PP	[0.2]
NP → Det N	[1]
Det → the	[1]
P → with	[1]
N → man	[0.6]
N → telescope	[0.3]
PP → P NP	[1]
N → N PP	[0.1]
V → sees	[1]

VP is the start symbol

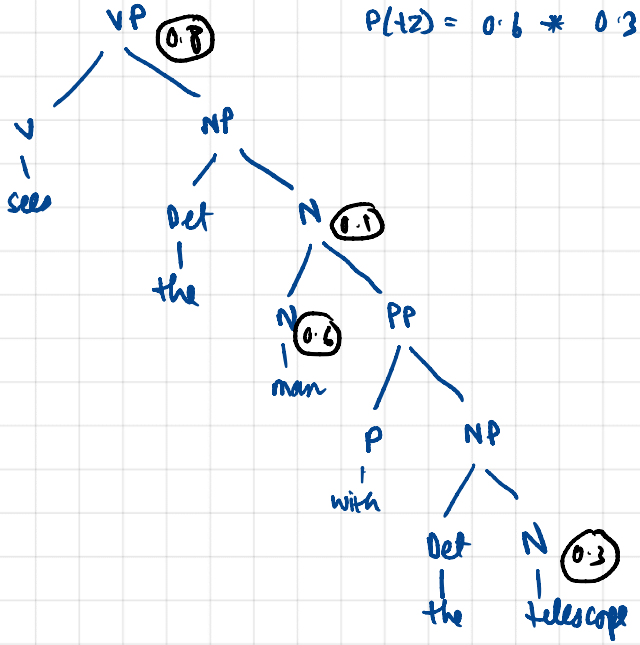
Probability of a Parse tree
= Product of the probabilities of the rules used to generate the parse tree.

Eg. sentence → see the man with the telescope



tree t1

$P(t1) = 0.1 * 0.8 * 0.3 * 0.2 = 0.02\ 88$



$P(t2) = 0.6 * 0.3 * 0.1 * 0.8 = 0.0144$

Tree t_2

consistent grammar → sum of all probabilities of all sentences in the language is equal to 1.

$P(VP, \text{sentence}) = P(t1) + P(t2) + \ldots = 1$

Example PCFG:

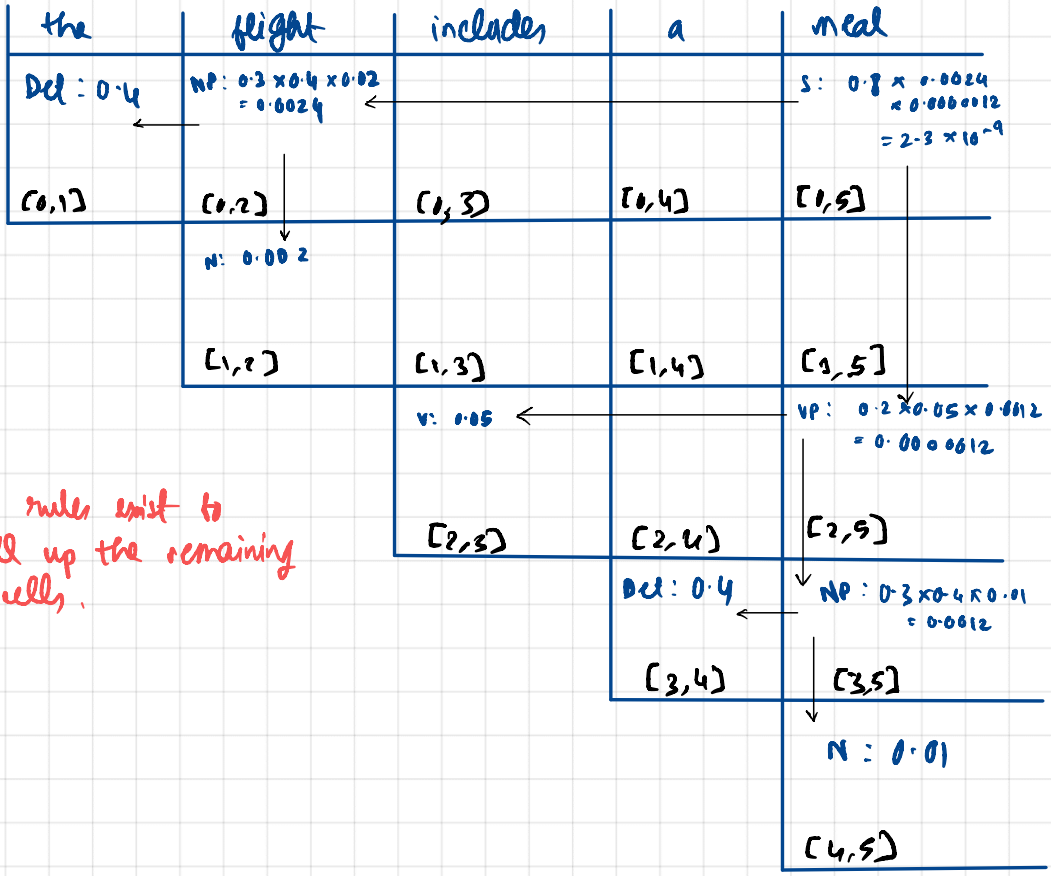
S → A [0.4]
S → B [0.6]
A → a [1]
B → B [1]

} → when infinite derivations are possible, grammar is inconsistent

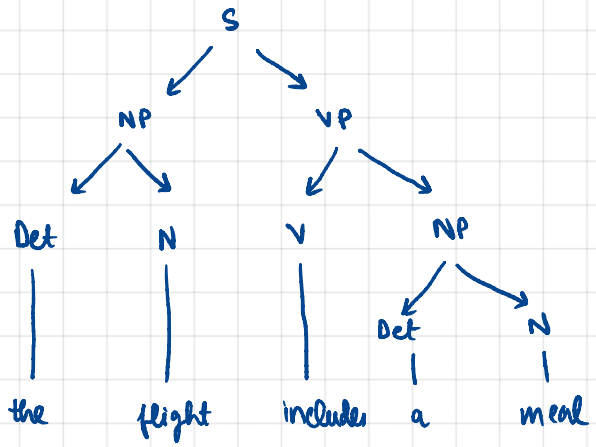
Probabilistic CKY parsing of PCFG:

S → NP VP [0.8]
NP → Det N [0.3]
VP → V NP [0.2]
V → includes [0.05]
Det → the [0.4]
Det → a [0.4]
N → meal [0.01]
N → flight [0.02]

sentence: the flight includes a meal



Homework Assignment → Write the algorithm for backtrack to find the most probable parse tree.



Learn PCFG rule probabilities

$$P(\alpha \to \beta | \alpha) = \frac{\text{count}(\alpha \to \beta)}{\sum_{\gamma} \text{count}(\alpha \to \gamma)} = \frac{\text{count}(\alpha \to \beta)}{\text{count}(\alpha)}$$

A corpus has to be used where there are annotated parse trees for each sentence.

3/9/22

input → Syntactic Analysis → syntactic structure → Semantic analysis → Meaning representation

Properties of Knowledge Base:

- verifiability

  eg. Does Spice Island serve veg food?
  ↳ serve (Spice Island, veg food)

- ambiguity

  eg. Give me the book?
  ↳ which book?

  – canonical form

- inference and variables

  eg. serves (x, vegfood).

Meaning representation can be in terms of FOPL.
↳ can be represented using Lambda notation.

Examples:

Every one likes chocolate

$\forall x \ \text{Person}(x) \Rightarrow \text{Likes}(x, \text{chocolate})$

Someone likes chocolate

$\exists x \ \text{Person}(x) \wedge \text{Likes}(x, \text{chocolates})$

Everyone likes chocolate unless they are allergic to it.

$\forall x \ (\text{Person}(x) \wedge \neg \text{allergic}(x, \text{chocolate})) \Rightarrow \text{likes}(x, \text{chocolate})$

A restaurant near ICSI serves mess food.

$\exists x \ (\text{Restaurant}(x) \wedge \text{serves}(x, \text{Messfood}) \wedge \text{near}(\text{Location}(x), \text{Location}(\text{ICSI}))$.

Lambda notation:

$\lambda x\ P(x)$

Lambda Reduction:

$\lambda x\ P(x)\ (A)$

↳ $P(A)$

eg. $\lambda x\ \lambda y$ Near (x, y) (Kolkata)

↳ λy Near (Kolkata, y)

eg. λy Near (Kolkata, y) (Howrah)

↳ Near (Kolkata, Howrah)

} Howrah is near Kolkata.

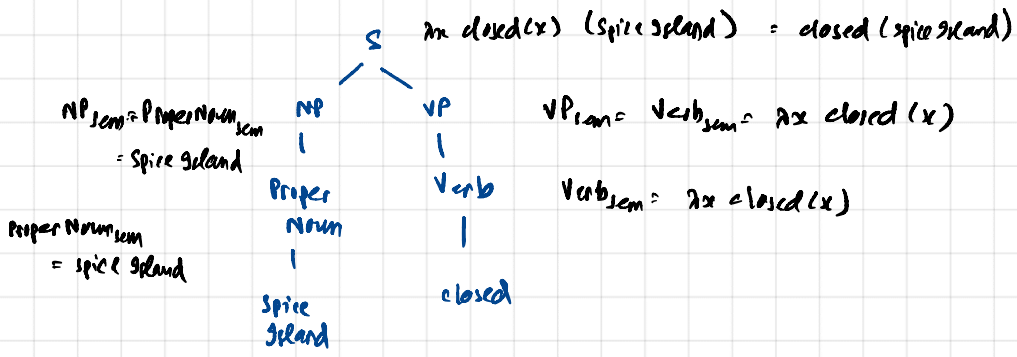
Syntax driven Semantic Analysis:

$A \rightarrow \alpha_1\ \alpha_2 \ldots \alpha_n$ { $f(\alpha_1.sem, \ldots, \alpha_k.sem)$ }

Example:

S → NP VP	{ VP$_{sem}$ (NP$_{sem}$) }
NP → ProperNoun	{ ProperNoun$_{sem}$ }
ProperNoun → spicesland	{ spicesland }
VP → Verb	{ Verb$_{sem}$ }
Verb → closed	{ λx closed (x) }

eg. Spice Island closed.



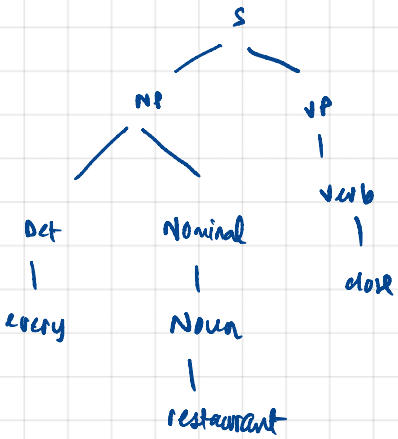
Example:

S → NP VP { $VP_{sem}(NP_{sem})$ }
NP → Det Nominal { $Det_{sem}(Nominal_{sem})$ }
Det → every { $\lambda P\ \lambda Q\ \forall x\ P(x) \Rightarrow Q(x)$ }
Nominal → Noun { $Noun_{sem}$ }
Noun → restaurant { λx Restaurant(x) }
Verb → closed { $\lambda x\ \exists e$ closed(e) $\wedge$ closedthing(e,x) }

eg. Every restaurant is closed

Ans: $\forall x$ Restaurant(x) $\Rightarrow \exists e$ closed(e) $\wedge$ closedthing(e,x)

H.W. →



Det.sem = $\lambda P\ \lambda Q\ \forall x\ P(x) \Rightarrow Q(x)$
Noun.sem = λx Restaurant(x)
Nominal.sem = λx Restaurant(x)
NP.sem = $\lambda P\ \lambda Q\ \forall x\ P(x) \Rightarrow Q(x)$ $(\lambda(x)$ Restaurant$(x))$
= $\lambda Q\ \forall x\ \lambda(x)$ Restaurant$(x) \Rightarrow Q(x)$

Verb.sem = $\lambda x\ \exists e$ closed(e) $\wedge$ closedthing(e,x)

5/5/23

df_t = Number of documents in the collection that contain the term t.

N = Total number of documents in the collection.

$idf_t = \log \frac{N}{df_t}$